Table of Contents

OrientDB Manual - version 2.2.x

Quick Navigation

Getting Started	Main Topics	Developers
Introduction to OrientDB	Basic Concepts	SQL
Installation	Supported Data Types	Gremlin
First Steps	Inheritance	HTTP API
Troubleshooting	Security	Java API
Enterprise Edition	Indexes	NodeJS
	ACID Transactions	PHP
	Functions	Python
	Caching Levels	.NET
	Common Use Cases	Other Drivers
		Network Binary Protocol
		Javadocs

Operations

- Installation
- 3rd party Plugins
- Upgrade
- Configuration
- Distributed Architecture (replication, sharding and high-availability)
- Performance Tuning
- ETL to Import any kind of data into OrientDB
- Import from Relational DB
- Backup and Restore
- Export and Import

Quick References

- Console
- Studio web tool
- Workbench (Enterprise Edition)
- OrientDB Server
- Network-Binary-Protocol
- Gephi Graph Analysis Visual tool
- Rexster Support and configuration
- Continuous integration

Resources

- User Group - Have question, troubles, problems?
- #orientdb IRC channel on freenode
- Professional Support
- Training - Training and classes.
- Events - Follow OrientDB at the next event!
- Team - Meet the team behind OrientDB
- Contribute - Contribute to the project.
- Who is using OrientDB? - Clients using OrientDB in production.

Questions or Need Help?

Check out our Get in Touch page for different ways of getting in touch with us.

PDF

This documentation is also available in PDF format.

Welcome to **OrientDB** - the first Multi-Model Open Source NoSQL DBMS that brings together the power of graphs and the flexibility of documents into one scalable high-performance operational database.

Getting Started

Over the past few years, there has been an explosion of many NoSQL database solutions and products. The meaning of the word "NoSQL" is not a campaign against the SQL language. In fact, OrientDB allows for SQL syntax! NoSQL is probably best described by the following:

> NoSQL, meaning "not only SQL", is a movement encouraging developers and business people to open their minds and consider new possibilities beyond the classic relational approach to data persistence.

Alternatives to relational database management systems have existed for many years, but they have been relegated primarily to niche use cases such as telecommunications, medicine, CAD and others. Interest in NoSQL alternatives like OrientDB is increasing dramatically. Not surprisingly, many of the largest web companies like Google, Amazon, Facebook, Foursquare and Twitter are using NoSQL based solutions in their production environments.

What motivates companies to leave the comfort of a well established relational database world? It is basically the great need to better solve today's data problems. Specifically, there are a few key areas:

- Performance
- Scalability (often huge)
- Smaller footprint
- Developer productivity and friendliness
- Schema flexibility

Most of these areas also happen to be the requirements of modern web applications. A few years ago, developers designed systems that could handle hundreds of concurrent users. Today it is not uncommon to have a potential target of thousands or millions of users connected and served at the same time.

Changing technology requirements have been taken into account on the application front by creating frameworks, introducing standards and leveraging best practices. However, in the database world, the situation has remained more or less the same for over 30 years. From the 1970s until recently, relational DBMSs have played the dominant role. Programming languages and methodologies have evolved, but the concept of data persistence and the DBMS have remained unchanged for the most part: it is all still tables, records and joins.

NoSQL Models

NoSQL-based solutions in general provide a powerful, scalable, and flexible way to solve data needs and use cases, which have previously been managed by relational databases. To summarize the NoSQL options, we'll examine the most common models or categories:

- **Key / Value databases**: where the data model is reduced to a simple hash table, which consists of key / value pairs. It is often easily distributed across multiple servers. The most recognized products of this group include Redis, Dynamo, and Riak.

- **Column-oriented databases**: where the data is stored in sections of columns offering more flexibility and easy aggregation. Facebook's Cassandra, Google's BigTable, and Amazon's SimpleDB are some examples of column-oriented databases.

- **Document databases**: where the data model consists of document collections, in which each individual document can have multiple fields without necessarily having a defined schema. The best known products of this group are MongoDB and CouchDB.

- **Graph databases**: where the domain model consists of vertices interconnected by edges creating rich graph structures. The best known products of this group are OrientDB, Neo4j and Titan.

> OrientDB is a document-graph database, meaning it has full native graph capabilities coupled with features normally only found in document databases.

Each of these categories or models has its own peculiarities, strengths and limitations. There is no single category or model, which is better than the others. However, certain types of databases are better at solving specific problems. This leads to the motto of NoSQL: choose the best tool for your specific use case.

The goal of Orient Technologies in building OrientDB was to create a robust, highly scalable database that can perform optimally in the widest possible set of use cases. Our product is designed to be a fantastic "go to" solution for practically all of your data persistence needs. In the following parts of this tutorial, we will look closely at **OrientDB**, one of the best open-source, multi-model, next generation NoSQL products on the market today.

Installation

OrientDB is available in two editions:

- **Community Edition** is released as an open source project under the Apache 2 license. This license allows unrestricted free usage for both open source and commercial projects.
- **Enterprise Edition** is commercial software built on top of the Community Edition. Enterprise is developed by the same team that developed the OrientDB engine. It serves as an extension of the Community Edition, providing Enterprise features, such as:
  - Non-Stop Backup and Restore
  - Scheduled FULL and Incremental Backups
  - Query Profiler
  - Distributed Clustering configuration
  - Metrics Recording
  - Live Monitoring with configurable Alerts

The Community Edition is available as a binary package for download or as source code on GitHub. The Enterprise Edition license is included with Support purchases.

Use Docker

If you have Docker installed in your computer, this is the easiest way to run OrientDB. From the command line type:

```
$ docker run -d --name orientdb -p 2424:2424 -p 2480:2480 \
   -e ORIENTDB_ROOT_PASSWORD=root orientdb:latest
```

Where instead of "root", type the root's password you want to use.

Use Ansible

If you manage your servers through Ansible, you can use the following role : https://galaxy.ansible.com/migibert/orientdb which is highly customizable and allows you to deploy OrientDB as a standalone instance or multiple clusterized instances.

For using it, you can follow these steps :

Install the role

```
ansible-galaxy install migibert.orientdb
```

Create an Ansible inventory

Assuming you have one two servers with respective IPs fixed at 192.168.10.5 and 192.168.10.6, using ubuntu user.

```
[orientdb-servers]
192.168.20.5 ansible_ssh_user=ubuntu
192.168.20.6 ansible_ssh_user=ubuntu
```

Create an Ansible playbook

In this example, we provision a two node cluster using multicast discovery mode. Please note that this playbook assumes java is already installed on the machine so you should have one step before that install Java 8 on the servers

```
- hosts: orientdb-servers
  become: yes
  vars:
    orientdb_version: 2.0.5
    orientdb_enable_distributed: true
    orientdb_distributed:
      hazelcast_network_port: 2434
      hazelcast_group: orientdb
      hazelcast_password: orientdb
      multicast_enabled: True
      multicast_group: 235.1.1.1
      multicast_port: 2434
      tcp_enabled: False
      tcp_members: []
    orientdb_users:
      - name: root
        password: root
         tasks:
  - apt:
      name: openjdk-8-jdk
      state: present
  roles:
  - role: orientdb-role
```

Run the playbook `ansible-playbook -i inventory playbook.yml`

Prerequisites

Both editions of OrientDB run on any operating system that implements the Java Virtual machine (JVM). Examples of these include:

- Linux, all distributions, including ARM (Raspberry Pi, etc.)
- Mac OS X
- Microsoft Windows, from 95/NT and later
- Solaris
- HP-UX
- IBM AIX

OrientDB requires Java, version 1.7 or higher.

> **Note**: In OSGi containers, OrientDB uses a `ConcurrentLinkedHashMap` implementation provided by concurrentlinkedhashmap to create the LRU based cache. This library actively uses the sun.misc package which is usually not exposed as a system package. To overcome this limitation you should add property `org.osgi.framework.system.packages.extra` with value `sun.misc` to your list of framework properties.
>
> It may be as simple as passing an argument to the VM starting the platform:
>
> ```
> $ java -Dorg.osgi.framework.system.packages.extra=sun.misc
> ```

Binary Installation

OrientDB provides a pre-compiled binary package to install the database on your system. Depending on your operating system, this is a tarred or zipped package that contains all the relevant files you need to run OrientDB. For desktop installations, go to OrientDB Downloads and select the package that best suits your system.

On server installations, you can use the `wget` utility:

```
$ wget https://s3.us-east-2.amazonaws.com/orientdb3/releases/2.2.37/orientdb-community-2.2.37.zip -O orientdb-community-2.2.37
.zip
```

Whether you use your web browser or `wget`, unzip or extract the downloaded file into a directory convenient for your use, (for example, `/opt/orientdb/` on Linux). This creates a directory called orientdb-community-2.2.37 with relevant files and scripts, which you will need to run OrientDB on your system.

Source Code Installation

In addition to downloading the binary packages, you also have the option of compiling OrientDB from the Community Edition source code, available on GitHub. This process requires that you install Git and Apache Maven on your system.

To compile OrientDB from source code, clone the Community Edition repository, then run Maven (`mvn`) in the newly created directory:

```
$ git clone https://github.com/orientechnologies/orientdb
$ git checkout develop
$ cd orientdb
$ mvn clean install
```

It is possible to skip tests:

```
$ mvn clean install -DskipTests
```

The develop branch contains code for the next version of OrientDB. Stable versions are tagged on master branch. For each maintained version OrientDB has its own `hotfix` branch. As the time of writing this notes, the state of branches is:

- develop: work in progress for next 3.0.x release (3.0.x-SNAPSHOT)
- 2.2.x: hot fix for next 2.2.x stable release (2.2.x-SNAPSHOT)
- 2.1.x: hot fix for next 2.1.x stable release (2.1.x-SNAPSHOT)
- 2.0.x: hot fix for next 2.0.x stable release (2.0.x-SNAPSHOT)
- last tag on master is 2.2.0

The build process installs all jars in the local maven repository and creates archives under the `distribution` module inside the `target` directory. At the time of writing, building from branch 2.1.x gave:

```
$ls -l distribution/target/
total 199920
    1088 26 Jan 09:57 archive-tmp
     102 26 Jan 09:57 databases
     102 26 Jan 09:57 orientdb-community-2.2.1-SNAPSHOT.dir
48814386 26 Jan 09:57 orientdb-community-2.2.1-SNAPSHOT.tar.gz
53542231 26 Jan 09:58 orientdb-community-2.2.1-SNAPSHOT.zip
$
```

The directory `orientdb-community-2.2.1-SNAPSHOT.dir` contains the OrientDB distribution uncompressed. Take a look to Contribute to OrientDB if you want to be involved.

Update Permissions

For Linux, Mac OS X and UNIX-based operating system, you need to change the permissions on some of the files after compiling from source.

```
$ chmod 755 bin/*.sh
$ chmod -R 777 config
```

These commands update the execute permissions on files in the `config/` directory and shell scripts in `bin/` , ensuring that you can run the scripts or programs that you've compiled.

Post-installation Tasks

For desktop users installing the binary, OrientDB is now installed and can be run through shell scripts found in the package `bin` directory of the installation. For servers, there are some additional steps that you need to take in order to manage the database server for OrientDB as a service. The procedure for this varies, depending on your operating system.

- Install as Service on Unix, Linux and Mac OS X
- Install as Service on Microsoft Windows

Upgrading

When the time comes to upgrade to a newer version of OrientDB, the methods vary depending on how you chose to install it in the first place. If you installed from binary downloads, repeat the download process above and update any symbolic links or shortcuts to point to the new directory.

For systems where OrientDB was built from source, pull down the latest source code and compile from source.

```
$ git pull origin master
$ mvn clean install
```

Bear in mind that when you build from source, you can switch branches to build different versions of OrientDB using Git. For example,

```
$ git checkout 2.2.x
$ mvn clean install
```

builds the `2.2.x` branch, instead of `master`.

Building a single executable jar with OrientDB

OrientDB for internal components like engines, operators, factories uses Java SPI Service Provider Interface. That means that the jars of OrientDB are shipped with files in `META-INF/services` that contains the implementation of components. Bear in mind that when building a single executable jar, you have to concatenate the content of files with the same name in different orientdb-*.jar . If you are using Maven Shade Plugin you can use Service Resource Transformer to do that.

Other Resources

To learn more about how to install OrientDB on specific environments, please refer to the guides below:

- Install with Docker
- Install with Ansible
- Install on Linux Ubuntu
- Install on JBoss AS
- Install on GlassFish
- Install on Ubuntu 12.04 VPS (DigitalOcean)
- Install on Vagrant

Running the OrientDB Server

When you finish installing OrientDB, whether you build it from source or download the binary package, you are ready to launch the database server. You can either start it through the system daemon or through the provided server script. This article only covers the latter.

> **Note**: If you would like to run OrientDB as a service on your system, there are some additional steps that you need to take. This provides alternate methods for starting the server and allows you to launch it as a daemon when your system boots. For more information on this process see:
>
> - Install OrientDB as a Service on Unix, Linux and Mac OS X
> - Install OrientDB as a Service on Microsoft Windows

Starting the Database Server

While you can run the database server as system daemon, you also have the option of starting it directly. In the OrientDB installation directory, (that is `$ORIENTDB_HOME`), under `bin` , there is a file named `server.sh` on Unix-based systems and `server.bat` on Windows. Executing this file starts the server.

To launch the OrientDB database server, run the following commands:

```
$ cd $ORIENTDB_HOME/bin
$ ./server.sh

           .
          .`        `
       ,      `:.
         `,`    ,:`
         .,.   :,,
         .,,  ,,,
    .    .,.:::::  ````
    ,`   .::,,,,::.,,,,,,`;;                      .:
    `,.  ::,,,,,,,:.,,.`  `                       .:
     ,,:,:,,,,,,,,::.   `        `         ``     .:
      ,,:.,,,,,,,,,: `::, ,,   ::,::`   : :,::`  ::::
       ,:,,,,,,,,,,::,:   ,,  :.    :   ::    :   .:
        :,,,,,,,,,,:,::   ,,  :      :  :     :   .:
  `     :,,,,,,,,,,:,::,  ,, .::::::::  :     :   .:
  `,...,,:,,,,,,,,,: .:,. ,, ,,         :     :   .:
    .,,,,::,,,,,,,:  `: , ,,  :     `   :     :   .:
      ...,::,,,,::.. `:  .,,  :,    :   :     :   .:
           ,::::,,,. `:   ,,   :::::    :     :   .:
           ,,:` `,,.
          ,,,    .,`
         ,,.     `,                      S E R V E R
       ``        `.
                 ``
                 `

2012-12-28 01:25:46:319 INFO Loading configuration from: config/orientdb-server-config.xml... [OServerConfigurationLoaderXml]
2012-12-28 01:25:46:625 INFO OrientDB Server v1.6 is starting up... [OServer]
2012-12-28 01:25:47:142 INFO -> Loaded memory database 'temp' [OServer]
2012-12-28 01:25:47:289 INFO Listening binary connections on 0.0.0.0:2424 [OServerNetworkListener]
2012-12-28 01:25:47:290 INFO Listening http connections on 0.0.0.0:2480 [OServerNetworkListener]
2012-12-28 01:25:47:317 INFO OrientDB Server v1.6 is active. [OServer]
```

The database server is now running. It is accessible on your system through ports `2424` and `2480` . At the first startup the server will ask for the root user password. The password is stored in the config file.

Stop the Server

On the console where the server is running a simple CTRL+c will shutdown the server.

The shutdown.sh (shutdown.bat) script could be used to stop the server:

```
$ cd $ORIENTDB_HOME/bin
$ ./shutdown.sh -p ROOT_PASSWORD
```

On ***nix** systems a simple call to shutdown.sh will stop the server running on localhost:

```
$ cd $ORIENTDB_HOME/bin
$ ./shutdown.sh
```

It is possible to stop servers running on remote hosts or even on different ports on localhost:

```
$ cd $ORIENTDB_HOME/bin
$ ./shutdown.sh -h odb1.mydomain.com -P 2424-2430 -u root -p ROOT_PASSWORD
```

List of params

- -h | --host **HOSTNAME or IP ADDRESS** : the host or ip where OrientDB is running, default to **localhost**
- -P | --ports **PORT or PORT RANGE** : single port value or range of ports; default to **2424-2430**
- -u | --user **ROOT USERNAME** : root's username; deafult to **root**
- -p | --password **ROOT PASSWORD** : root's user password; **mandatory**

NOTE On Windows systems password is always **mandatory** because the script isn't able to discover the pid of the OrientDB's process.

Server Log Messages

Following the masthead, the database server begins to print log messages to standard output. This provides you with a guide to what OrientDB does as it starts up on your system.

1. The database server loads its configuration file from the file `$ORIENTDB_HOME/config/orientdb-server-config.xml` .

   > For more information on this step, see OrientDB Server.

2. The database server loads the `temp` database into memory. You can use this database for storing temporary data.
3. The database server begins listening for binary connections on port `2424` for all configured networks, (`0.0.0.0`).
4. The database server begins listening for HTTP connections on port `2480` for all configured networks, (`0.0.0.0`).

Accessing the Database Server

By default, OrientDB listens on two different ports for external connections.

- **Binary**: OrientDB listens on port `2424` for binary connections from the console and for clients and drivers that support the Network Binary Protocol.
- **HTTP**: OrientDB listens on port `2480` for HTTP connections from OrientDB Studio Web Tool and clients and drivers that support the HTTP/REST protocol, or similar tools, such as cURL.

If you would like the database server to listen at different ports or IP address, you can define these values in the configuration file `config/orientdb-server-config.xml` .

Running the OrientDB Console

Once the server is running there are various methods you can use to connect to your database server to an individual databases. Two such methods are the Network Binary and HTTP/REST protocols. In addition to these OrientDB provides a command-line interface for connecting to and working with the database server.

Starting the OrientDB Console

In the OrientDB installation directory (that is, `$ORIENTDB_HOME`, where you installed the database) under `bin`, there is a file called `console.sh` for Unix-based systems or `console.bat` for Windows users.

To launch the OrientDB console, run the following command after you start the database server:

```
$ cd $ORIENTDB_HOME/bin
$ ./console.sh

OrientDB console v.X.X.X (build 0) www.orientdb.com
Type 'HELP' to display all the commands supported.
Installing extensions for GREMLIN language v.X.X.X

orientdb>
```

The OrientDB console is now running. From this prompt you can connect to and manage any remote or local databases available to you.

Using the `HELP` Command

In the event that you are unfamiliar with OrientDB and the available commands, or if you need help at any time, you can use the `HELP` command, or type `?` into the console prompt.

```
orientdb> HELP

AVAILABLE COMMANDS:
 * alter class <command-text>   Alter a class in the database schema
 * alter cluster <command-text> Alter class in the database schema
 ...                            ...
 * help                         Print this help
 * exit                         Close the console
```

For each console command available to you, `HELP` documents its basic use and what it does. If you know the particular command and need details on its use, you can provide arguments to `HELP` for further clarification.

```
orientdb> HELP SELECT

COMMAND: SELECT
- Execute a query against the database and display the results.
SYNTAX: select <query-text>
WHERE:
- <query-text>: The query to execute
```

Connecting to Server Instances

There are some console commands, such as `LIST DATABASES` or `CREATE DATABASE`, which you can only run while connected to a server instance. For other commands, however, you must also connect to a database, before they run without error.

> Before you can connect to a fresh server instance and fully control it, you need to know the root password for the database. The root password is located in the configuration file at `config/orientdb-server-config.xml`. You can find it by searching for the `<users>` element. If you want to change it, edit the configuration file and restart the server.
>
> ```
> ...
> <users>
>    <user resources="*"
>          password="my_root_password"
>          name="root"/>
>    <user resources="connect,server.listDatabases,server.dblist"
>          password="my_guest_password"
>          name="guest"/>
> </users>
> ...
> ```

With the required credentials, you can connect to the database server instance on your system, or establish a remote connection to one running on a different machine.

```
orientdb> CONNECT remote:localhost root my_root_password

Connecting to remote Server instance [remote:localhost] with user 'root'...OK
```

Once you have established a connection to the database server, you can begin to execute commands on that server, such as `LIST DATABASES` and `CREATE DATABASE`.

```
orientdb> LIST DATABASES

Found 1 databases:
* GratefulDeadConcerts (plocal)
```

To connect to this database or to a different one, use the `CONNECT` command from the console and specify the server URL, username, and password. By default, each database has an `admin` user with a password of `admin`.

> **Warning**: Always change the default password on production databases.

The above `LIST DATABASES` command shows a `GratefulDeadConcerts` installed on the local server. To connect to this database, run the following command:

```
orientdb> CONNECT remote:localhost/GratefulDeadConcerts admin admin

Connecting to database [remote:localhost/GratefulDeadConcerts] with user 'admin'...OK
```

The `CONNECT` command takes a specific syntax for its URL. That is, `remote:localhost/GratefulDeadConcerts` in the example. It has three parts:

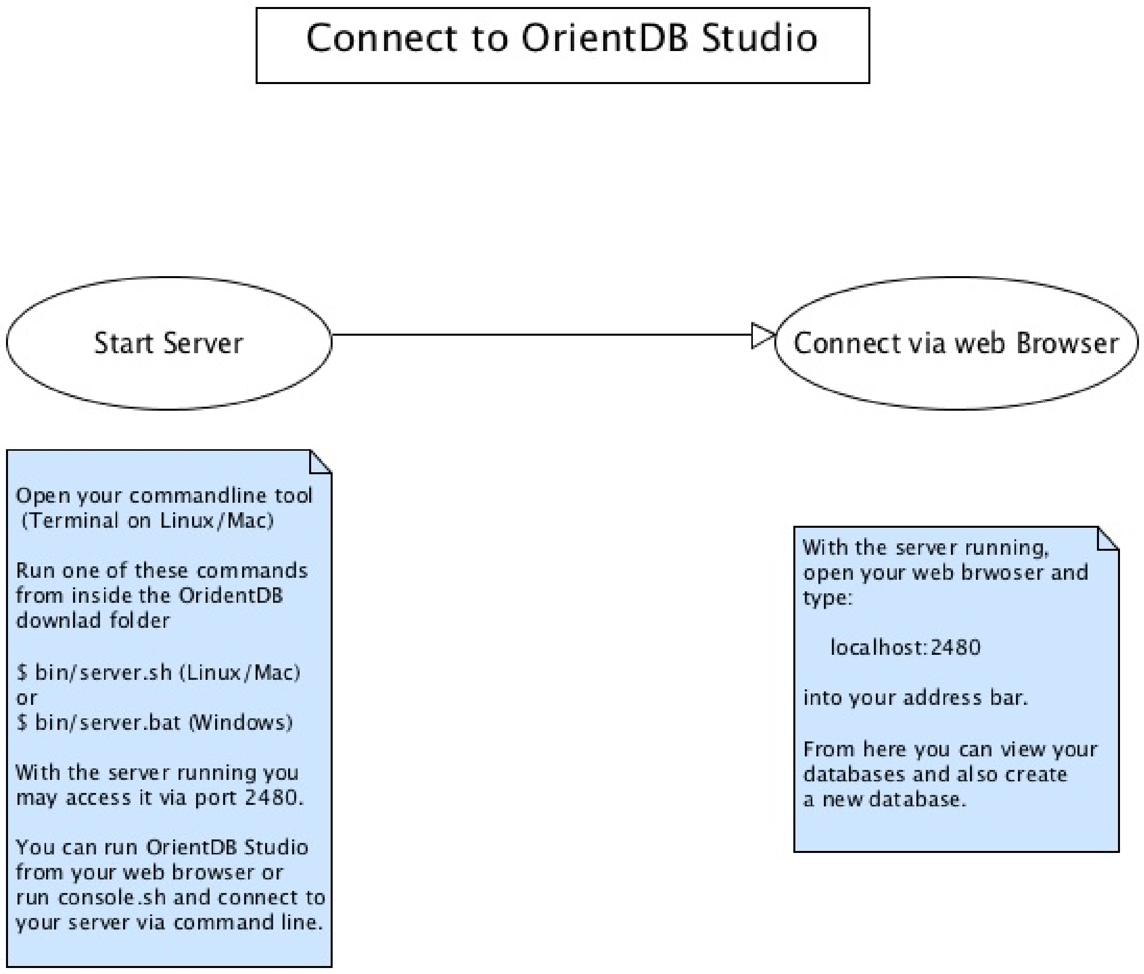
- **Protocol**: The first part of the database address is the protocol the console should use in the connection. In the example, this is `remote`, indicating that it should use the TCP/IP protocol.
- **Address**: The second part of the database address is hostname or IP address of the database server that you want the console to connect to. In the example, this is `localhost`, since the connection is made to a server instance running on the local file system.
- **Database**: The third part of the address is the name of the database that you want to use. In the case of the example, this is `GratefulDeadConcerts`.

For more detailed information about the commands, see Console Commands.

> **Note**: The OrientDB distribution comes with the bundled database `GratefulDeadConcerts` which represents the Graph of the Grateful Dead's concerts. This database can be used by anyone to start exploring the features and characteristics of OrientDB.

Run the Studio

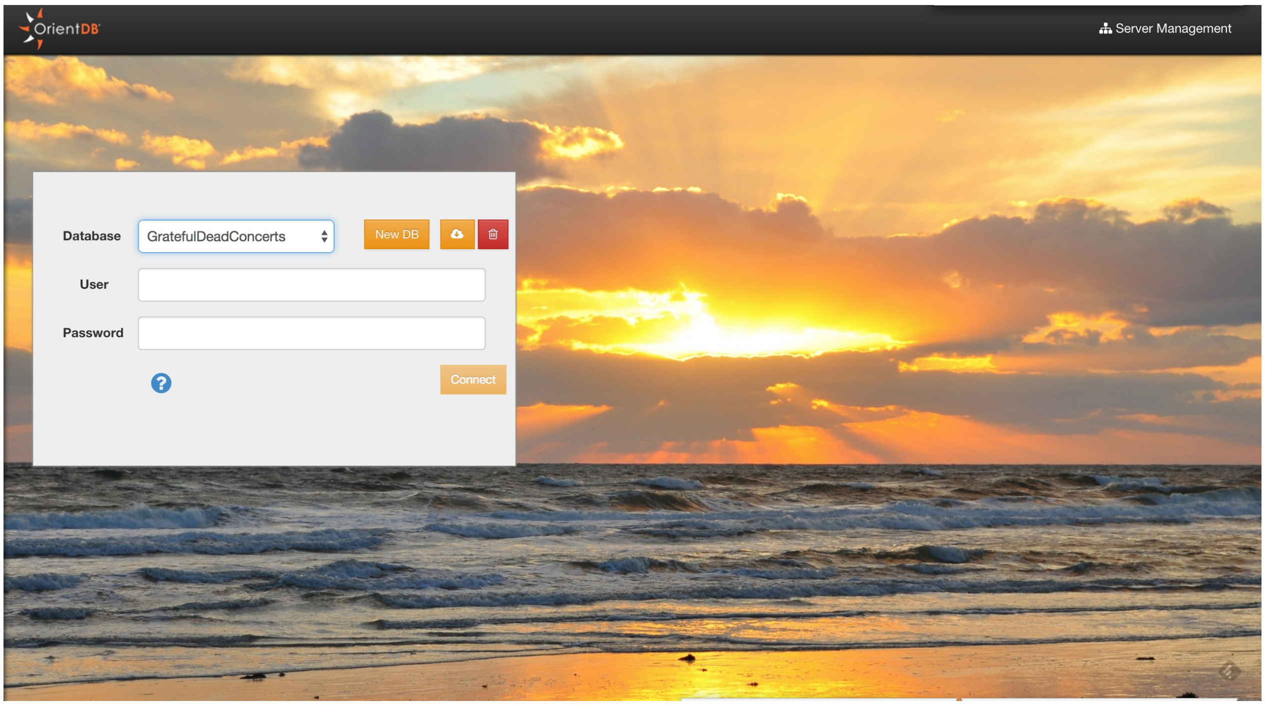
If you're more comfortable interacting with database systems through a graphical interface then you can accomplish the most common database tasks with OrientDB Studio, the web interface.


Connect to OrientDB Studio
Start Server
Connect via web Browser
Open your commandline tool (Terminal on Linux/Mac)
Run one of these commands from inside the OridentDB downlad folder
$ bin/server.sh (Linux/Mac) or
$ bin/server.bat (Windows)
With the server running you may access it via port 2480.
You can run OrientDB Studio from your web browser or run console.sh and connect to your server via command line.
With the server running, open your web brwoser and type:
localhost:2480
into your address bar.
From here you can view your databases and also create a new database.

Connecting to Studio

By default, there are no additional steps that you need to take to start OrientDB Studio. When you launch the Server, whether through the start-up script `server.sh` or as a system daemon, the Studio web interface opens automatically with it.

```
$ firefox http://localhost:2480
```



From here you can create a new database, connect to or drop an existing database, import a public database and navigate to the Server management interface.

> For more information on the OrientDB Studio, see Studio.

Classes

Multi-model support in the OrientDB engine provides a number of ways in approaching and understanding its basic concepts. These concepts are clearest when viewed from the perspective of the Document Database API. Like many database management systems, OrientDB uses the Record as an element of storage. There are many types of records, but with the Document Database API, records always use the Document type. Documents are formed by a set of key/value pairs, referred to as fields and properties, and can belong to a class.

The Class is a concept drawn from the Object-oriented programming paradigm. It is a type of data model that allows you to define certain rules for records that belong to it. In the traditional Document database model, it is comparable to the collection, while in the Relational database model it is comparable to the table.

> For more information on classes in general, see Wikipedia.

To list all the configured classes on your system, use the `LIST CLASSES` command in the console:

```
orientdb> LIST CLASSES

CLASSES:
-------------------+------------+----------+-----------+
 NAME              | SUPERCLASS |CLUSTERS  | RECORDS   |
-------------------+------------+----------+-----------+
 AbstractPerson    |            | -1       |         0 |
 Account           |            | 11       |      1126 |
 Actor             |            | 91       |         3 |
 Address           |            | 19       |       166 |
 Animal            |            | 17       |         0 |
 ....              | ....       | ....     |      .... |
 Whiz              |            | 14       |      1001 |
-------------------+------------+----------+-----------+
 TOTAL                                           22775 |
-------------------------------------------------------+
```

Working with Classes

In order to start using classes with your own applications, you need to understand how to create and configure a class for use. The class in OrientDB is similar to the table in relational databases, but unlike tables, classes can be schema-less, schema-full or mixed. A class can inherit properties from other classes thereby creating trees of classes (though the super-class relationship).

Each class has its own cluster or clusters, (created by default, if none are defined). For now we should know that a cluster is a place where a group of records are stored. We'll soon see how `clustering` improves performance of querying the database.

> For more information on classes in OrientDB, see Class.

To create a new class, use the `CREATE CLASS` command:

```
orientdb> CREATE CLASS Student

Class created successfully. Total classes in database now: 92
```

This creates a class called `Student`. Given that no cluster was defined in the `CREATE CLASS` command, OrientDB creates a default cluster called `student`, to contain records assigned to this class. For the moment, the class has no records or properties tied to it. It is now displayed in the `CLASSES` listings.

Adding Properties to a Class

As mentioned above, OrientDB does allow you to work in a schema-less mode. That is, it allows you to create classes without defining their properties. However, in the event that you would like to define indexes or constraints for your class, properties are mandatory. Following the comparison to relational databases, if classes in OrientDB are similar to tables, properties are the columns on those tables.

To create new properties on `Student`, use the `CREATE PROPERTY` command in the console:

```
orientdb> CREATE PROPERTY Student.name STRING

Property created successfully with id=1


orientdb> CREATE PROPERTY Student.surname STRING

Property created successfully with id=2


orientdb> CREATE PROPERTY Student.birthDate DATE

Property created successfully with id=3
```

These commands create three new properties on the `Student` class to provide you with areas to define the individual student's name, surname and date of birth.

Displaying Class Information

On occasion, you may need to reference a particular class to see what clusters it belongs to and any properties configured for its use. Using the `INFO CLASS` command, you can display information on the current configuration and properties of a class.

To display information on the class `Student`, use the `INFO CLASS` command:

```
orientdb> INFO CLASS Student

Class................: Student
Default cluster......: student (id=96)
Supported cluster ids: [96]
Properties:
-----------+--------+--------------+-----------+----------+----------+-----+-----+
 NAME      | TYPE   | LINKED TYPE/ | MANDATORY | READONLY | NOT NULL | MIN | MAX |
           |        | CLASS        |           |          |          |     |     |
-----------+--------+--------------+-----------+----------+----------+-----+-----+
 birthDate | DATE   | null         | false     | false    | false    |     |     |
 name      | STRING | null         | false     | false    | false    |     |     |
 surname   | STRING | null         | false     | false    | false    |     |     |
-----------+--------+--------------+-----------+----------+----------+-----+-----+
```

Adding Constraints to Properties

Constraints create limits on the data values assigned to properties. For instance, the type, the minimum or maximum size of, whether or not a value is mandatory or if null values are permitted to the property.

To add a constraint, use the `ALTER PROPERTY` command:

```
orientdb> ALTER PROPERTY Student.name MIN 3

Property updated successfully
```

This command adds a constraint to `Student` on the `name` property. It sets it so that any value given to this class and property must have a minimum of three characters.

Viewing Records in a Class

Classes contain and define records in OrientDB. You can view all records that belong to a class using the `BROWSE CLASS` command and data belonging to a particular record with the `DISPLAY RECORD` command.

In the above examples, you created a `Student` class and defined the schema for records that belong to that class, but you did not create these records or add any data. As a result, running these commands on the `Student` class returns no results. Instead, for the examples below, consider the `OUser` class.

```
orientdb> INFO CLASS OUser

CLASS 'OUser'

Super classes........: [OIdentity]
Default cluster......: ouser (id=5)
Supported cluster ids: [5]
Cluster selection....: round-robin
Oversize.............: 0.0

PROPERTIES
----------+---------+-------------+-----------+----------+----------+-----+-----+
 NAME     | TYPE    | LINKED TYPE/ | MANDATORY | READONLY | NOT NULL | MIN | MAX |
          |         | CLASS        |           |          |          |     |     |
----------+---------+-------------+-----------+----------+----------+-----+-----+
 password | STRING  | null         | true      | false    | true     |     |     |
 roles    | LINKSET | ORole        | false     | false    | false    |     |     |
 name     | STRING  | null         | true      | false    | true     |     |     |
 status   | STRING  | null         | true      | false    | true     |     |     |
----------+---------+-------------+-----------+----------+----------+-----+-----+

INDEXES (1 altogether)
-------------------------------+----------------+
 NAME                          | PROPERTIES     |
-------------------------------+----------------+
 OUser.name                    | name           |
-------------------------------+----------------+
```

OrientDB ships with a number of default classes, which it uses in configuration and in managing data on your system, (the classes with the `O` prefix shown in the `CLASSES` command output). The `OUser` class defines the users on your database.

To see records assigned to the `OUser` class, run the `BROWSE CLASS` command:

```
orientdb> BROWSE CLASS OUser

---+------+-------+--------+-----------------------------------+--------+-------+
 # | @RID | @Class| name   | password                          | status | roles |
---+------+-------+--------+-----------------------------------+--------+-------+
 0 | #5:0 | OUser | admin  | {SHA-256}8C6976E5B5410415BDE90... | ACTIVE | [1]   |
 1 | #5:1 | OUser | reader | {SHA-256}3D0941964AA3EBDCB00EF... | ACTIVE | [1]   |
 2 | #5:2 | OUser | writer | {SHA-256}B93006774CBDD4B299389... | ACTIVE | [1]   |
---+------+-------+--------+-----------------------------------+--------+-------+
```

!	In the example, you are listing all of the users of the database. While this is fine for your initial setup and as an example, it is not particularly secure. To further improve security in production environments, see Security.

When you run `BROWSE CLASS`, the first column in the output provides the identifier number, which you can use to display detailed information on that particular record.

To show the first record browsed from the `OUser` class, run the `DISPLAY RECORD` command:

```
orientdb> DISPLAY RECORD 0

------------------------------------------------------------------------------+
 Document - @class: OUser                      @rid: #5:0      @version: 1    |
----------+-------------------------------------------------------------------+
     Name | Value                                                             |
----------+-------------------------------------------------------------------+
     name | admin                                                             |
 password | {SHA-256}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873F8A81F6F2AB... |
   status | ACTIVE                                                            |
    roles | [#4:0=#4:0]                                                       |
----------+-------------------------------------------------------------------+
```

Bear in mind that this command references the last call of `BROWSE CLASS`. You can continue to display other records, but you cannot display records from another class until you browse that particular class.

Clusters

The Cluster is a place where a group of records are stored. Like the Class, it is comparable with the collection in traditional document databases, and in relational databases with the table. However, this is a loose comparison given that unlike a table, clusters allow you to store the data of a class in different physical locations.

To list all the configured clusters on your system, use the `LIST CLUSTERS` command in the console:

```
orientdb> LIST CLUSTERS

CLUSTERS:
-------------+------+-----------+-----------+
 NAME        | ID   | TYPE      | RECORDS   |
-------------+------+-----------+-----------+
 account     | 11   | PHYSICAL  |      1107 |
 actor       | 91   | PHYSICAL  |         3 |
 address     | 19   | PHYSICAL  |       166 |
 animal      | 17   | PHYSICAL  |         0 |
 animalrace  | 16   | PHYSICAL  |         2 |
 ....        | .... | ....      |      .... |
-------------+------+-----------+-----------+
 TOTAL                                23481 |
---------------------------------------------+
```

Understanding Clusters

By default, OrientDB creates one cluster for each Class. Starting from v2.2, OrientDB automatically creates multiple clusters per each class (the number of clusters created is equals to the number of CPU's cores available on the server) to improve using of parallelism. All records of a class are stored in the same cluster, which has the same name as the class. You can create up to 32,767 (or, 2^{15} - 1) clusters in a database. Understanding the concepts of classes and clusters allows you to take advantage of the power of clusters in designing new databases.

While the default strategy is that each class maps to one cluster, a class can rely on multiple clusters. For instance, you can spawn records physically in multiple locations, thereby creating multiple clusters.

Here, you have a class `Customer` that relies on two clusters:

- `USA_customers`, which is a cluster that contains all customers in the United States.
- `China_customers`, which is a cluster that contains all customers in China.

In this deployment, the default cluster is `USA_customers`. Whenever commands are run on the `Customer` class, such as `INSERT` statements, OrientDB assigns this new data to the default cluster.

The new entry from the `INSERT` statement is added to the `USA_customers` cluster, given that it's the default. Inserting data into a non-default cluster would require that you specify the cluster you want to insert the data into in your statement.

When you run a query on the `Customer` class, such as `SELECT` queries, for instance:

OrientDB scans all clusters associated with the class in looking for matches.

In the event that you know the cluster in which the data is stored, you can query that cluster directly to avoid scanning all others and optimize the query.

Here, OrientDB only scans the `China_customers` cluster of the `Customer` class in looking for matches

> **Note**: The method OrientDB uses to select the cluster, where it inserts new records, is configurable and extensible. For more information, see Cluster Selection.

Working with Clusters

While running in HA mode, upon the creation of a new record (document, vertex, edge, etc.) the coordinator server automatically assigns the cluster among the list of local clusters for the current server. For more information look at HA: Cluster Ownership.

You may also find it beneficial to locate different clusters on different servers, physically separating where you store records in your database. The advantages of this include:

- **Optimization** Faster query execution against clusters, given that you need only search a subset of the clusters in a class.
- **Indexes** With good partitioning, you can reduce or remove the use of indexes.
- **Parallel Queries**: Queries can be run in parallel when made to data on multiple disks.
- **Sharding**: You can shard large data-sets across multiple instances.

Adding Clusters

When you create a class, OrientDB creates a default cluster of the same name. In order for you to take advantage of the power of clusters, you need to create additional clusters on the class. This is done with the `ALTER CLASS` statement in conjunction with the `ADDCLUSTER` parameter.

To add a cluster to the `Customer` class, use an `ALTER CLASS` statement in the console:

```
orientdb> ALTER CLASS Customer ADDCLUSTER UK_Customers

Class updated successfully
```

You now have a third cluster for the `Customer` class, covering those customers located in the United Kingdom.

Viewing Records in a Cluster

Clusters store the records contained by a class in OrientDB. You can view all records that belong to a cluster using the `BROWSE CLUSTER` command and the data belonging to a particular record with the `DISPLAY RECORD` command.

In the above example, you added a cluster to a class for storing records customer information based on their locations around the world, but you did not create these records or add any data. As a result, running these commands on the `Customer` class returns no results. Instead, for the examples below, consider the `ouser` cluster.

OrientDB ships with a number of default clusters to store data from its default classes. You can see these using the `CLUSTERS` command. Among these, there is the `ouser` cluster, which stores data of the users on your database.

To see records stored in the `ouser` cluster, run the `BROWSE CLUSTER` command:

```
orientdb> BROWSE CLUSTER OUser

---+------+--------+--------+----------------------------------+--------+-------+
 # | @RID | @CLASS | name   | password                         | status | roles |
---+------+-------+--------+----------------------------------+--------+-------+
 0 | #5:0 | OUser | admin  | {SHA-256}8C6976E5B5410415BDE90... | ACTIVE | [1]   |
 1 | #5:1 | OUser | reader | {SHA-256}3D0941964AA3EBDCB00CC... | ACTIVE | [1]   |
 2 | #5:2 | OUser | writer | {SHA-256}B93006774CBDD4B299389... | ACTIVE | [1]   |
---+------+--------+--------+----------------------------------+--------+-------+
```

The results are identical to executing `BROWSE CLASS` on the `OUser` class, given that there is only one cluster for the `OUser` class in this example.

❗	In the example, you are listing all of the users of the database. While this is fine for your initial setup and as an example, it is not particularly secure. To further improve security in production environments, see Security.

When you run `BROWSE CLUSTER`, the first column in the output provides the identifier number, which you can use to display detailed information on that particular record.

To show the first record browsed from the `ouser` cluster, run the `DISPLAY RECORD` command:

```
orientdb> DISPLAY RECORD 0

------------------------------------------------------------------------------+
 Document - @class: OUser                      @rid: #5:0      @version: 1    |
----------+-------------------------------------------------------------------+
     Name | Value                                                             |
----------+-------------------------------------------------------------------+
     name | admin                                                             |
 password | {SHA-256}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873F8A81F6F2AB... |
   status | ACTIVE                                                            |
    roles | [#4:0=#4:0]                                                       |
----------+-------------------------------------------------------------------+
```

Bear in mind that this command references the last call of `BROWSE CLUSTER`. You can continue to display other records, but you cannot display records from another cluster until you browse that particular cluster.

Record ID

In OrientDB, each record has its own self-assigned unique ID within the database called Record ID or RID. It is composed of two parts:

```
#<cluster-id>:<cluster-position>
```

That is,

- `<cluster-id>` The cluster identifier.
- `<cluster-position>` The position of the data within the cluster.

Each database can have a maximum of 32,767 clusters, or 2^{15} - 1. Each cluster can handle up to 9,223,372,036,780,000 records, or 2^{63}, namely 9,223,372 trillion records.

> The maximum size of a database is 2^{78} records, or 302,231,454,903 trillion records. Due to limitations in hardware resources, OrientDB has not been tested at such high numbers, but there are users working with OrientDB in the billions of records range.

Loading Records

Each record has a Record ID, which notes the physical position of the record inside the database. What this means is that when you load a record by its RID, the load is significantly faster than it would be otherwise.

In document and relational databases, the more data that you have, the slower the database responds. OrientDB handles relationships as physical links to the records. The relationship is assigned only once, when the edge is created `O(1)`. You can compare this to relational databases, which compute the relationship every time the database is run `O(log N)`. In OrientDB, the size of a database does not effect the traverse speed. The speed remains constant, whether for one record or one hundred billion records. This is a critical feature in the age of Big Data.

To directly load a record, use the `LOAD RECORD` command in the console.

```
orientdb> LOAD RECORD #12:4

--------------------------------------------------------
 ODocument - @class: Company  @rid: #12:4  @version: 8
-------------+------------------------------------------
        Name | Value
-------------+------------------------------------------
   addresses | [NOT LOADED: #19:159]
      salary | 0.0
   employees | 100004
          id | 4
        name | Microsoft4
 initialized | false
     salary2 | 0.0
  checkpoint | true
     created | Sat Dec 29 23:13:49 CET 2012
-------------+------------------------------------------
```

The `LOAD RECORD` command returns some useful information about this record. It shows:

- that it is a document. OrientDB supports different types of records, but document is the only type covered in this chapter.
- that it belongs to the `Company` class.
- that its current version is `8`. OrientDB uses an MVCC system. Every time you update a record, its version increments by one.

- that we have different field types: floats in `salary` and `salary2`, integers for `employees` and `id`, string for `name`, booleans for `initialized` and `checkpoint`, and date-time for `created`.
- that the field `addresses` has been `NOT LOADED`. It is also a `LINK` to another record, `#19:159`. This is a relationship. For more information on this concept, see Relationships.

Relationships

One of the most important features of Graph databases lies in how they manage relationships. Many users come to OrientDB from MongoDB due to OrientDB having more efficient support for relationships.

Relations in Relational Databases

Most database developers are familiar with the Relational model of databases and with relational database management systems, such as MySQL and MS-SQL. Given its more than thirty years of dominance, this has long been thought the best way to handle relationships. By contrast, Graph databases suggest a more modern approach to this concept.

Consider, as an example, a database where you need to establish relationships between `Customer` and `Address` tables.

1-to-1 Relationship

Relational databases store the value of the target record in the `address` row of the `Customer` table. This is the Foreign Key. The foreign key points to the Primary Key of the related record in the `Address` table.

Consider a case where you want to view the address of a customer named Luca. In a Relational database, like MySQL, this is how you would query the table:

```
mysql> SELECT B.location FROM Customer A, Address B
         WHERE A.name='Luca' AND A.address=B.id;
```

What happens here is a `JOIN`. That is, the contents of two tables are joined to form the results. The database executes the `JOIN` every time you retrieve the relationship.

1-to-Many Relationship

Given that Relational databases have no concept of a collections, the `Customer` table cannot have multiple foreign keys. The only way to manage a 1-to-Many Relationship in databases of this kind is to move the Foreign Key to the `Address` table.

For example, consider a case where you want to return all addresses connected to the customer Luca, this is how you would query the table:

```
mysql> SELECT B.location FROM Customer A, Address B
         WHERE A.name='Luca' AND B.customer=A.id;
```

Many-to-Many relationship

The most complicated case is the Many-to-Many relationship. To handle associations of this kind, Relational databases require a separate, intermediary table that matches rows from both `Customer` and `Address` tables in all required combinations. This results in a double `JOIN` per record at runtime.

For example, consider a case where you want to return all address for the customer Luca, this is how you would query the table:

```
mysql> SELECT C.location FROM Customer A, CustomerAddress B, Address C
         WHERE A.name='Luca' AND B.id=A.id AND B.address=C.id;
```

Understanding `JOIN`

In document and relational database systems, the more data that you have, the slower the database responds and `JOIN` operations have a heavy runtime cost.

For relational database systems, the database computes the relationship every time you query the server. That translates to `O(log N / block_size)`. OrientDB handles relationships as physical links to the records and assigns them only once, when the edge is created. That is, `O(1)`.

In OrientDB, the speed of traversal is not affected by the size of the database. It is always constant regardless of whether it has one record or one hundred billion records. This is a critical feature in the age of Big Data.

Searching for an identifier at runtime each time you execute a query, for every record will grow very expensive. The first optimization with relational databases is the use of indexing. Indexes speed up searches, but they slow down `INSERT`, `UPDATE`, and `DELETE` operations. Additionally, they occupy a substantial amount of space on the disk and in memory.

Consider also whether searching an index is actually fast.

Indexes and `JOIN`

In the database industry, there are a number of indexing algorithms available. The most common in both relational and NoSQL database systems is the B+ Tree.

Balance trees all work in a similar manner. For example, consider a case where you're looking for an entry with the name `Luca`: after only five hops, the record is found.

While this is fine on a small database, consider what would happen if there were millions or billions of records. The database would have to go through many, many more hops to find `Luca`. And, the database would execute this operation on every `JOIN` per record. Picture: joining four tables with thousands of records. The number of `JOIN` operations could run in the millions.

Relations in OrientDB

There is no `JOIN` in OrientDB. Instead, it uses `LINK`. `LINK` is a relationship managed by storing the target Record ID in the source record. It is similar to storing the pointer between two objects in memory.

When you have `Invoice` linked to `Customer`, then you have a pointer to `Customer` inside `Invoice` as an attribute. They are exactly the same. In this way, it's as though your database was kept in memory: a memory of several exabytes.

Types of Relationships

In 1-to-N relationships, OrientDB handles the relationship as a collection of Record ID's, as you would when managing objects in memory.

OrientDB supports several different kinds of relationships:

- `LINK` Relationship that points to one record only.
- `LINKSET` Relationship that points to several records. It is similar to Java sets, the same Record ID can only be included once. The pointers have no order.
- `LINKLIST` Relationship that points to several records. It is similar to Java lists, they are ordered and can contain duplicates.
- `LINKMAP` Relationship that points to several records with a key stored in the source record. The Map values are the Record ID's. It is similar to Java `Map<?,Record>`.

SQL

Most NoSQL products employ a custom query language. In this, OrientDB differs by focusing on standards in query languages. That is, instead of inventing "Yet Another Query Language," it begins with the widely used and well-understood language of SQL. It then extends SQL to support more complex graphing concepts, such as Trees and Graphs.

Why SQL? Because SQL is ubiquitous in the database development world. It is familiar and more readable and concise than its competitors, such as Map Reduce scripts or JSON based querying.

`SELECT`

The `SELECT` statement queries the database and returns results that match the given parameters. For instance, earlier in Getting Started, two queries were presented that gave the same results: `BROWSE CLUSTER ouser` and `BROWSE CLASS OUser`. Here is a third option, available through a `SELECT` statement.

```
orientdb> SELECT FROM OUser
```

Notice that the query has no projections. This means that you do not need to enter a character to indicate that the query should return the entire record, such as the asterisk in the Relational model, (that is, `SELECT * FROM OUser`).

Additionally, OUser is a class. By default, OrientDB executes queries against classes. Targets can also be:

- **Clusters** To execute against a cluster, rather than a class, prefix `CLUSTER` to the target name.

  ```
  orientdb> SELECT FROM CLUSTER:Ouser
  ```

- **Record ID** To execute against one or more Record ID's, use the identifier(s) as your target. For example.

  ```
  orientdb> SELECT FROM #10:3
  orientdb> SELECT FROM [#10:1, #10:30, #10:5]
  ```

- **Indexes** To execute a query against an index, prefix `INDEX` to the target name.

  ```
  orientdb> SELECT VALUE FROM INDEX:dictionary WHERE key='Jay'
  ```

`WHERE`

Much like the standard implementation of SQL, OrientDB supports `WHERE` conditions to filter the returning records too. For example,

```
orientdb> SELECT FROM OUser WHERE name LIKE 'l%'
```

This returns all `OUser` records where the name begins with `l`. For more information on supported operators and functions, see `WHERE`.

`ORDER BY`

In addition to `WHERE`, OrientDB also supports `ORDER BY` clauses. This allows you to order the results returned by the query according to one or more fields, in either ascending or descending order.

```
orientdb> SELECT FROM Employee WHERE city='Rome' ORDER BY surname ASC, name ASC
```

The example queries the `Employee` class, it returns a listing of all employees in that class who live in Rome and it orders the results by surname and name, in ascending order.

`GROUP BY`

In the event that you need results of the query grouped together according to the values of certain fields, you can manage this using the `GROUP BY` clause.

```
orientdb> SELECT SUM(salary) FROM Employee WHERE age < 40 GROUP BY job
```

In the example, you query the `Employee` class for the sum of the salaries of all employees under the age of forty, grouped by their job types.

`LIMIT`

In the event that your query returns too many results, making it difficult to read or manage, you can use the `LIMIT` clause to reduce it to the top most of the return values.

```
orientdb> SELECT FROM Employee WHERE gender='male' LIMIT 20
```

In the example, you query the `Employee` class for a list of male employees. Given that there are likely to be a number of these, you limit the return to the first twenty entries.

`SKIP`

When using the `LIMIT` clause with queries, you can only view the topmost of the return results. In the event that you would like to view certain results further down the list, for instance the values from twenty to forty, you can paginate your results using the `SKIP` keyword in the `LIMIT` clause.

```
orientdb> SELECT FROM Employee WHERE gender='male' LIMIT 20
orientdb> SELECT FROM Employee WHERE gender='male' SKIP 20 LIMIT 20
orientdb> SELECT FROM Employee WHERE gender='male' SKIP 40 LIMIT 20
```

The first query returns the first twenty results, the second returns the next twenty results, the third up to sixty. You can use these queries to manage pages at the application layer.

`INSERT`

The `INSERT` statement adds new data to a class and cluster. OrientDB supports three forms of syntax used to insert new data into your database.

- The standard ANSI-93 syntax:

  ```
  orientdb> INSERT INTO    Employee(name, surname, gender)
            VALUES('Jay', 'Miner', 'M')
  ```

- The simplified ANSI-92 syntax:

  ```
  orientdb> INSERT INTO Employee SET name='Jay', surname='Miner', gender='M'
  ```

- The JSON syntax:

  ```
  orientdb> INSERT INTO Employee CONTENT {name : 'Jay', surname : 'Miner',
            gender : 'M'}
  ```

Each of these queries adds Jay Miner to the `Employee` class. You can choose whichever syntax that works best with your application.

UPDATE

The `UPDATE` statement changes the values of existing data in a class and cluster. In OrientDB there are two forms of syntax used to update data on your database.

- The standard ANSI-92 syntax:

  ```
  orientdb> UPDATE Employee SET local=TRUE WHERE city='London'
  ```

- The JSON syntax, used with the `MERGE` keyword, which merges the changes with the current record:

  ```
  orientdb> UPDATE Employee MERGE { local : TRUE } WHERE city='London'
  ```

Each of these statements updates the `Employee` class, changing the `local` property to `TRUE` when the employee is based in London.

DELETE

The `DELETE` statement removes existing values from your class and cluster. OrientDB supports the standard ANSI-92 compliant syntax for these statements:

```
orientdb> DELETE FROM Employee WHERE city <> 'London'
```

Here, entries are removed from the `Employee` class where the employee in question is not based in London.

See also:

- The SQL Reference
- The Console Command Reference

Working with Graphs

In graph databases, the database system graphs data into network-like structures consisting of vertices and edges. In the OrientDB Graph model, the database represents data through the concept of a property graph, which defines a vertex as an entity linked with other vertices and an edge, as an entity that links two vertices.

OrientDB ships with a generic vertex persistent class, called `V`, as well as a class for edges, called `E`. As an example, you can create a new vertex using the `INSERT` command with `V`.

```
orientdb> INSERT INTO V SET name='Jay'

Created record with RID #9:0
```

In effect, the Graph model database works on top of the underlying document model. But, in order to simplify this process, OrientDB introduces a new set of commands for managing graphs from the console. Instead of `INSERT`, use `CREATE VERTEX`

```
orientdb> CREATE VERTEX V SET name='Jay'

Created vertex with RID #9:1
```

By using the graph commands over the standard SQL syntax, OrientDB ensures that your graphs remain consistent. For more information on the particular commands, see the following pages:

- CREATE VERTEX
- DELETE VERTEX
- CREATE EDGE
- UPDATE EDGE
- DELETE EDGE

Use Case: Social Network for Restaurant Patrons

While you have the option of working with vertexes and edges in your database as they are, you can also extend the standard `V` and `E` classes to suit the particular needs of your application. The advantages of this approach are,

- It grants better understanding about the meaning of these entities.
- It allows for optional constraints at the class level.
- It improves performance through better partitioning of entities.
- It allows for object-oriented inheritance among the graph elements.

For example, consider a social network based on restaurants. You need to start with a class for individual customers and another for the restaurants they patronize. Create these classes to extend the `V` class.

```
orientdb> CREATE CLASS Person EXTENDS V

orientdb> CREATE CLASS Restaurant EXTENDS V
```

Doing this creates the schema for your social network. Now that the schema is ready, populate the graph with data.

```
orientdb> CREATE VERTEX Person SET name='Luca'

Created record with RID #11:0


orientdb> CREATE VERTEX Person SET name='Bill'

Created record with RID #11:1


orientdb> CREATE VERTEX Person SET name='Jay'

Created record with RID #11:2


orientdb> CREATE VERTEX Restaurant SET name='Dante', type='Pizza'

Created record with RID #12:0


orientdb> CREATE VERTEX Restaurant SET name='Charlie', type='French'

Created record with RID #12:1
```

This adds three vertices to the `Person` class, representing individual users in the social network. It also adds two vertices to the `Restaurant` class, representing the restaurants that they patronize.

Creating Edges

For the moment, these vertices are independent of one another, tied together only by the classes to which they belong. That is, they are not yet connected by edges. Before you can make these connections, you first need to create a class that extends `E`.

```
orientdb> CREATE CLASS Eat EXTENDS E
```

This creates the class `Eat`, which extends the class `E`. `Eat` represents the relationship between the vertex `Person` and the vertex `Restaurant`.

When you create the edge from this class, note that the orientation of the vertices is important, because it gives the relationship its meaning. For instance, creating an edge in the opposite direction, (from `Restaurant` to `Person`), would call for a separate class, such as `Attendee`.

The user Luca eats at the pizza joint Dante. Create an edge that represents this connection:

```
orientdb> CREATE EDGE Eat FROM ( SELECT FROM Person WHERE name='Luca' )
          TO ( SELECT FROM Restaurant WHERE name='Dante' )
```

Creating Edges from Record ID

In the event that you know the Record ID of the vertices, you can connect them directly with a shorter and faster command. For example, the person Bill also eats at the restaurant Dante and the person Jay eats at the restaurant Charlie. Create edges in the class `Eat` to represent these connections.

```
orientdb> CREATE EDGE Eat FROM #11:1 TO #12:0

orientdb> CREATE EDGE Eat FROM #11:2 TO #12:1
```

Querying Graphs

In the above example you created and populated a small graph of a social network of individual users and the restaurants at which they eat. You can now begin to experiment with queries on a graph database.

To cross edges, you can use special graph functions, such as:

- `OUT()` To retrieve the adjacent outgoing vertices
- `IN()` To retrieve the adjacent incoming vertices
- `BOTH()` To retrieve the adjacent incoming and outgoing vertices

For example, to know all of the people who eat in the restaurant Dante, which has a Record ID of `#12:0` , you can access the record for that restaurant and traverse the incoming edges to discover which entries in the `Person` class connect to it.

```
orientdb> SELECT IN() FROM Restaurant WHERE name='Dante'

-------+----------------+
 @RID  | in             |
-------+----------------+
 #-2:1 | [#11:0, #11:1] |
-------+----------------+
```

This query displays the record ID's from the `Person` class that connect to the restaurant Dante. In cases such as this, you can use the `EXPAND()` special function to transform the vertex collection in the result-set by expanding it.

```
orientdb> SELECT EXPAND( IN() ) FROM Restaurant WHERE name='Dante'

-------+-------------+------------+---------+
 @RID  | @CLASS      | Name       | out_Eat |
-------+-------------+------------+---------+
 #11:0 | Person      | Luca       | #12:0   |
 #11:1 | Person      | Bill       | #12:0   |
-------+-------------+------------+---------+
```

Creating Edge to Connect Users

Your application at this point shows connections between individual users and the restaurants they patronize. While this is interesting, it does not yet function as a social network. To do so, you need to establish edges that connect the users to one another.

To begin, as before, create a new class that extends `E` :

```
orientdb> CREATE CLASS Friend EXTENDS E
```

The users Luca and Jay are friends. They have Record ID's of `#11:0` and `#11:2` . Create an edge that connects them.

```
orientdb> CREATE EDGE Friend FROM #11:0 TO #11:2
```

In the `Friend` relationship, orientation is not important. That is, if Luca is a friend of Jay's then Jay is a friend of Luca's. Therefore, you should use the `BOTH()` function.

```
orientdb> SELECT EXPAND( BOTH( 'Friend' ) ) FROM Person WHERE name = 'Luca'

-------+-------------+-------------+---------+-----------+
 @RID  | @CLASS      | Name        | out_Eat | in_Friend |
-------+-------------+-------------+---------+-----------+
 #11:2 | Person      | Jay         | #12:1   | #11:0     |
-------+-------------+-------------+---------+-----------+
```

Here, the `BOTH()` function takes the edge class `Friend` as an argument, crossing only relationships of the Friend kind, (that is, it skips the `Eat` class, at this time). Note in the result-set that the relationship with Luca, with a Record ID of `#11:0` in the `in_` field.

You can also now view all the restaurants patronized by friends of Luca.

```
orientdb> SELECT EXPAND( BOTH('Friend').out('Eat') ) FROM Person
          WHERE name='Luca'

-------+-------------+-------------+-------------+--------+
 @RID  | @CLASS      | Name        | Type        | in_Eat |
-------+-------------+-------------+-------------+--------+
 #12:1 | Restaurant  | Charlie     | French      | #11:2  |
-------+-------------+-------------+-------------+--------+
```

Lightweight Edges

In version 1.4.x, OrientDB begins to manage some edges as Lightweight Edges. Lightweight Edges do not have Record ID's, but are physically stored as links within vertices. Note that OrientDB only uses a Lightweight Edge only when the edge has no properties, otherwise it uses the standard Edge.

From the logic point of view, Lightweight Edges are Edges in all effects, so that all graph functions work with them. This is to improve performance and reduce disk space.

Because Lightweight Edges don't exist as separate records in the database, some queries won't work as expected. For instance,

```
orientdb> SELECT FROM E
```

For most cases, an edge is used connecting vertices, so this query would not cause any problems in particular. But, it would not return Lightweight Edges in the result-set. In the event that you need to query edges directly, including those with no properties, disable the Lightweight Edge feature.

To disable the Lightweight Edge feature, execute the following command.

```
orientdb> ALTER DATABASE CUSTOM useLightweightEdges=FALSE
```

You only need to execute this command once. OrientDB now generates new edges as the standard Edge, rather than the Lightweight Edge. Note that this does not affect existing edges.

For troubleshooting information on Lightweight Edges, see Why I can't see all the edges. For more information in the Graph model in OrientDB, see Graph API.

Using Schema with Graphs

OrientDB, through the Graph API, offers a number of features above and beyond the traditional Graph Databases given that it supports concepts drawn from both the Document Database and the Object Oriented worlds. For instance, consider the power of graphs, when used in conjunction with schemas and constraints.

Use Case: Car Database

For this example, consider a graph database that maps the relationship between individual users and their cars. First, create the graph schema for the `Person` and `Car` vertex classes, as well as the `Owns` edge class to connect the two:

```
orientdb> CREATE CLASS Person EXTENDS V

orientdb> CREATE CLASS Car EXTENDS V

orientdb> CREATE CLASS Owns EXTENDS E
```

These commands lay out the schema for your graph database. That is, they define two vertex classes and an edge class to indicate the relationship between the two. With that, you can begin to populate the database with vertices and edges.

```
orientdb> CREATE VERTEX Person SET name = 'Luca'

Created vertex 'Person#11:0{name:Luca} v1' in 0,012000 sec(s).


orientdb> CREATE VERTEX Car SET name = 'Ferrari Modena'

Created vertex 'Car#12:0{name:Ferrari Modena} v1' in 0,001000 sec(s).


orientdb> CREATE EDGE Owns FROM ( SELECT FROM Person ) TO ( SELECT FROM Car )

Created edge '[e[#11:0->#12:0][#11:0-Owns->#12:0]]' in 0,005000 sec(s).
```

Querying the Car Database

In the above section, you create a car database and populated it with vertices and edges to map out the relationship between drivers and their cars. Now you can begin to query this database, showing what those connections are. For example, what is Luca's car? You can find out by traversing from the vertex Luca to the outgoing vertices following the `Owns` relationship.

```
orientdb> SELECT name FROM ( SELECT EXPAND( OUT('Owns') ) FROM Person
          WHERE name='Luca' )

----+-------+-----------------+
 #  | @RID  | name            |
----+-------+-----------------+
 0  | #-2:1 | Ferrari Modena  |
----+-------+-----------------+
```

As you can see, the query returns that Luca owns a Ferrari Modena. Now consider expanding your database to track where each person lives.

Adding a Location Vertex

Consider a situation, in which you might want to keep track of the countries in which each person lives. In practice, there are a number of reasons why you might want to do this, for instance, for the purposes of promotional material or in a larger database to analyze the connections to see how residence affects car ownership.

To begin, create a vertex class for the country, in which the person lives and an edge class that connects the individual to the place.

```
orientdb> CREATE CLASS Country EXTENDS V

orientdb> CREATE CLASS Lives EXTENDS E
```

This creates the schema for the feature you're adding to the cars database. The vertex class `Country` recording countries in which people live and the edge class `Lives` to connect individuals in the vertex class `Person` to entries in `Country` .

With the schema laid out, create a vertex for the United Kingdom and connect it to the person Luca.

```
orientdb> CREATE VERTEX Country SET name='UK'

Created vertex 'Country#14:0{name:UK} v1' in 0,004000 sec(s).


orientdb> CREATE EDGE Lives FROM ( SELECT FROM Person ) TO ( SELECT FROM Country )

Created edge '[e[#11:0->#14:0][#11:0-Lives->#14:0]]' in 0,006000 sec(s).
```

The second command creates an edge connecting the person Luca to the country United Kingdom. Now that your cars database is defined and populated, you can query it, such as a search that shows the countries where there are users that own a Ferrari.

```
orientdb> SELECT name FROM ( SELECT EXPAND( IN('Owns').OUT('Lives') )
          FROM Car WHERE name LIKE '%Ferrari%' )

---+-------+--------+
 # | @RID  | name   |
---+-------+--------+
 0 | #-2:1 | UK     |
---+-------+--------+
```

Using `in` and `out` Constraints on Edges

In the above sections, you modeled the graph using a schema without any constraints, but you might find it useful to use some. For instance, it would be good to require that an `Owns` relationship only exist between the vertex `Person` and the vertex `Car` .

```
orientdb> CREATE PROPERTY Owns.out LINK Person

orientdb> CREATE PROPERTY Owns.in LINK Car
```

These commands link outgoing vertices of the `Person` class to incoming vertices of the `Car` class. That is, it configures your database so that a user can own a car, but a car cannot own a user.

Using `MANDATORY` Constraints on Edges

By default, when OrientDB creates an edge that lacks properties, it creates it as a Lightweight Edge. That is, it creates an edge that has no physical record in the database. Using the `MANDATORY` setting, you can stop this behavior, forcing it to create the standard Edge, without outright disabling Lightweight Edges.

```
orientdb> ALTER PROPERTY Owns.out MANDATORY TRUE
orientdb> ALTER PROPERTY Owns.in MANDATORY TRUE
```

Using `UNIQUE` with Edges

For the sake of simplicity, consider a case where you want to limit the way people are connected to cars to where the user can only match to the car once. That is, if Luca owns a Ferrari Modena, you might prefer not to have a double entry for that car in the event that he buys a new one a few years later. This is particularly important given that our database covers make and model, but not year.

To manage this, you need to define a `UNIQUE` index against both the out and in properties.

```
orientdb> CREATE INDEX UniqueOwns ON Owns(out,in) UNIQUE

Created index successfully with 0 entries in 0,023000 sec(s).
```

The index returns tells us that no entries are indexed. You have already created the `Onws` relationship between Luca and the Ferrari Modena. In that case, however, OrientDB had created a Lightweight Edge before you set the rule to force the creation of documents for `Owns` instances. To fix this, you need to drop and recreate the edge.

```
orientdb> DELETE EDGE FROM #11:0 TO #12:0
orientdb> CREATE EDGE Owns FROM ( SELECT FROM Person ) TO ( SELECT FROM Car )
```

To confirm that this was successful, run a query to check that a record was created:

```
orientdb> SELECT FROM Owns

---+-------+-------+--------+
 # | @RID  | out   | in     |
---+-------+-------+--------+
 0 | #13:0 | #11:0 | #12:0  |
---+-------+-------+--------+
```

This shows that a record was indeed created. To confirm that the constraints work, attempt to create an edge in `Owns` that connects Luca to the United Kingdom.

```
orientdb> CREATE EDGE Owns FROM ( SELECT FROM Person ) TO ( SELECT FROM Country )

Error: com.orientechnologies.orient.core.exception.OCommandExecutionException:
Error on execution of command: sql.create edge Owns from (select from Person)...
Error: com.orientechnologies.orient.core.exception.OValidationException: The
field 'Owns.in' has been declared as LINK of type 'Car' but the value is the
document #14:0 of class 'Country'
```

This shows that the constraints effectively blocked the creation, generating a set of errors to explain why it was blocked.

You now have a typed graph with constraints. For more information, see Graph Schema.

Setting up a Distributed Graph Database

In addition to the standard deployment architecture, where it runs as a single, standalone database instance, you can also deploy OrientDB using Distributed Architecture. In this environment, it shares the database across multiple server instances.

Launching Distributed Server Cluster

There are two ways to share a database across multiple server nodes:

- Prior to startup, copy the specific database directory, under `$ORIENTDB_HOME/database` to all servers.
- Keep the database on the first running server node, then start every other server node. Under the default configurations, OrientDB automatically shares the database with the new servers that join.

This tutorial assumes that you want to start a distributed database using the second method.

NOTE: When you run in distributed mode, OrientDB needs more RAM. The minimum is 2GB of heap, but we suggest to use at least 4GB of heap memory. To change the heap modify the Java memory settings in the file `bin/dserver.sh` (or dserver.bat on Windows).

Starting the First Server Node

Unlike the standard standalone deployment of OrientDB, there is a different script that you need to use when launching a distributed server instance. Instead of `server.sh`, you use `dserver.sh`. In the case of Windows, use `dserver.bat`. Whichever you need, you can find it in the `bin` of your installation directory.

```
$ ./bin/dserver.sh
```

Bear in mind that OrientDB uses the same `orientdb-server-config.xml` configuration file, regardless of whether it's running as a server or distributed server. For more information, see Distributed Configuration.

The first time you start OrientDB as a distributed server, it generates the following output:

```
+---------------------------------------------------------------+
|         WARNING: FIRST DISTRIBUTED RUN CONFIGURATION          |
+---------------------------------------------------------------+
| This is the first time that the server is running as          |
| distributed. Please type the name you want to assign to the   |
| current server node.                                          |
|                                                               |
| To avoid this message set the environment variable or JVM     |
| setting ORIENTDB_NODE_NAME to the server node name to use.    |
+---------------------------------------------------------------+

Node name [BLANK=auto generate it]:
```

You need to give the node a name here. OrientDB stores it in the `nodeName` parameter of `OHazelcastPlugin`. It adds the variable to your `orientdb-server-config.xml` configuration file.

Distributed Startup Process

When OrientDB starts as a distributed server instance, it loads all databases in the `database` directory and configures them to run in distributed mode. For this reason, the first load, OrientDB copies the default distributed configuration, (that is, the `default-distributed-db-config.json` configuration file), into each database's directory, renaming it `distributed-config.json`. On subsequent starts, each database uses this file instead of the default configuration file. Since the shape of the cluster changes every time nodes join or leave, the configuration is kept up to date by each distributed server instance.

For more information on working with the `default-distributed-db-config.json` configuration file, see Distributed Configuration.

Starting Additional Server Nodes

When you have the first server node running, you can begin to start the other server nodes. Each server requires the same Hazelcast credentials in order to join the same cluster. You can define these in the `hazelcast.xml` configuration file.

The fastest way to initialize multiple server nodes is to copy the OrientDB installation directory from the first node to each of the subsequent nodes. For instance,

```
$ scp user@ip_address $ORIENTDB_HOME
```

This copies both the databases and their configuration files onto the new distributed server node.

> Bear in mind, if you run multiple server instances on the same host, such as when testing, you need to change the port entry in the `hazelcast.xml` configuration file.

For the other server nodes in the cluster, use the same `dserver.sh` command as you used in starting the first node. When the other server nodes come online, they begin to establish network connectivity with each other. Monitoring the logs, you can see where they establish connections from messages such as this:

```
WARN [node1384014656983] added new node id=Member [192.168.1.179]:2435 name=null
     [OHazelcastPlugin]
INFO [192.168.1.179]:2434 [orientdb] Re-partitioning cluster data... Migration
     queue size: 135 [PartitionService]
INFO [192.168.1.179]:2434 [orientdb] All migration tasks has been completed,
     queues are empty. [PartitionService]
INFO [node1384014656983] added node configuration id=Member [192.168.1.179]:2435
     name=node1384015873680, now 2 nodes are configured [OHazelcastPlugin]
INFO [node1384014656983] update configuration db=GratefulDeadConcerts
     from=node1384015873680 [OHazelcastPlugin]
WARN [node1383734730415]->[node1384015873680] deploying database
     GratefulDeadConcerts...[ODeployDatabaseTask]
WARN [node1383734730415]->[node1384015873680] sending the compressed database
     GratefulDeadConcerts over the network, total 339,66Kb [ODeployDatabaseTask]
```

In the example, two server nodes were started on the same machine. It has an IP address of 10.37.129.2, but is using OrientDB on two different ports: 2434 and 2435, where the current is called `this`. The remainder of the log is relative to the distribution of the database to the second server.

On the second server node output, OrientDB dumps messages like this:

```
WARN [node1384015873680]<-[node1383734730415] installing database
     GratefulDeadConcerts in databases/GratefulDeadConcerts... [OHazelcastPlugin]
WARN [node1384015873680] installed database GratefulDeadConcerts in
     databases/GratefulDeadConcerts, setting it online... [OHazelcastPlugin]
WARN [node1384015873680] database GratefulDeadConcerts is online [OHazelcastPlugin]
WARN [node1384015873680] updated node status to 'ONLINE' [OHazelcastPlugin]
INFO OrientDB Server v2.2.11-SNAPSHOT is active. [OServer]
```

What these messages mean is that the database `GratefulDeadConcerts` was correctly installed from the first node, that is `node1383734730415` through the network.

Migrating from standalone server to a cluster

If you have a standalone instance of OrientDB and you want to move to a cluster you should follow these steps:

- Install OrientDB on all the servers of the cluster and configure it (according to the sections above)
- Stop the standalone server
- Copy the specific database directories under `$ORIENTDB_HOME/database` to all the servers of the cluster
- Start all the servers in the cluster using the script `dserver.sh` (or `dserver.bat` if on Windows)

If the standalone server will be part of the cluster, you can use the existing installation of OrientDB; you don't need to copy the database directories since they're already in place and you just have to start it before all the other servers with `dserver.sh`.

Working with Distributed Graphs

When OrientDB joins a distributed cluster, all clients connecting to the server node are constantly notified about this state. This ensures that, in the event that server node fails, the clients can switch transparently to the next available server.

You can check this through the console. When OrientDB runs in a distributed configuration, the current cluster shape is visible through the `INFO` command.

```
$ $ORIENTDB_HOME/bin/console.sh

OrientDB console v.1.6 www.orientechnologies.com
Type 'help' to display all the commands supported.
Installing extensions for GREMLIN language v.2.5.0-SNAPSHOT


orientdb> CONNECT remote:localhost/GratefulDeadConcerts admin admin

Connecting to database [remote:localhost/GratefulDeadConcerts] with user 'admin'...OK


orientdb> INFO

Current database: GratefulDeadConcerts (url=remote:localhost/GratefulDeadConcerts)
```

For reference purposes, the server nodes in the example have the following configurations. As you can see, it is a two node cluster running a single server host. The first node listens on port `2481` while the second on port `2480`.

```
+---------+------+-----------------------------------------+-----+---------+--------------+--------------+--------------------
---+
|Name     |Status|Databases                                |Conns|StartedOn|Binary        |HTTP          |UsedMemory
   |
+---------+------+-----------------------------------------+-----+---------+--------------+--------------+--------------------
---+
|europe-0 |ONLINE|distributed-node-deadlock=ONLINE (MASTER)|5    |16:53:59 |127.0.0.1:2424|127.0.0.1:2480|269.32MB/3.56GB (7.40
%)|
|europe-1 |ONLINE|distributed-node-deadlock=ONLINE (MASTER)|4    |16:54:03 |127.0.0.1:2425|127.0.0.1:2481|268.89MB/3.56GB (7.38
%)|
+---------+------+-----------------------------------------+-----+---------+--------------+--------------+--------------------
---+
```

Testing Distributed Architecture

Once you have a distributed database up and running, you can begin to test its operations on a running environment. For example, begin by creating a vertex, setting the `node` property to `1`.

```
orientdb> CREATE VERTEX V SET node = 1

Created vertex 'V#9:815{node:1} v1' in 0,013000 sec(s).
```

From another console, connect to the second node and execute the following command:

```
orinetdb> SELECT FROM V WHERE node = 1

----+--------+-------+
 #  | @RID   | node  |
----+--------+-------+
 0  | #9:815 | 1     |
----+--------+-------+
1 item(s) found. Query executed in 0.19 sec(s).
```

This shows that the vertex created on the first node has successfully replicated to the second node.

Logs in Distributed Architecture

From time to time server nodes go down. This does not necessarily relate to problems in OrientDB, (for instance, it could originate from limitations in system resources).

To test this out, kill the first node. For example, assuming the first node has a process identifier, (that is, a PID), of `1254` on your system, run the following command:

```
$ kill -9 1254
```

This command kills the process on PID `1254`. Now, check the log messages for the second node:

```
$ less orientdb.log

INFO [127.0.0.1]:2435 [orientdb] Removing Member [127.0.0.1]:2434
     [ClusterService]
INFO [127.0.0.1]:2435 [orientdb]
Members [1] {
    Member [127.0.0.1]:2435 this
}
 [ClusterService]
WARN [europe-0] node removed id=Member [127.0.0.1]:2434
     name=europe-1 [OHazelcastPlugin]
INFO [127.0.0.1]:2435 [orientdb] Partition balance is ok, no need to
     re-partition cluster data...  [PartitionService]
```

What the logs show you is that the second node is now aware that it cannot reach the first node. You can further test this by running the console connected to the first node..

```
orientdb> SELECT FROM V LIMIT 2

WARN Caught I/O errors from /127.0.0.1:2425 (local
     socket=0.0.0.0/0.0.0.0:51512), trying to reconnect (error:
     java.io.IOException: Stream closed) [OStorageRemote]
WARN Connection re-acquired transparently after 30ms and 1 retries: no errors
     will be thrown at application level [OStorageRemote]
---+------+----------------+--------+--------------+------+-----------------+-----
 # | @RID | name           | song_type | performances | type | out_followed_by | ...
---+------+----------------+--------+--------------+------+-----------------+-----
 1 | #9:1 | HEY BO DIDDLEY | cover  | 5            | song | [5]             | ...
 2 | #9:2 | IM A MAN       | cover  | 1            | song | [2]             | ...
---+------+----------------+--------+--------------+------+-----------------+-----
```

This shows that the console auto-switched to the next available node. That is, it switched to the second node upon noticing that the first was no longer functional. The warnings reports show what happened in a transparent way, so that the application doesn't need to manage the issue.

From the console connected to the second node, create a new vertex.

```
orientdb> CREATE VERTEX V SET node=2

Created vertex 'V#9:816{node:2} v1' in 0,014000 sec(s).
```

Given that the first node remains nonfunctional, OrientDB journals the operation. Once the first node comes back online, the second node synchronizes the changes into it.

Restart the first node and check that it successfully auto-realigns. Reconnect the console to the first node and run the following command:

```
orientdb> SELECT FROM V WHERE node=2

---+--------+-------+
 # | @RID   | node  |
---+--------+-------+
 0 | #9:816 | 2     |
---+--------+-------+
1 item(s) found. Query executed in 0.209 sec(s).
```

This shows that the first node has realigned itself with the second node.

This process is repeatable with N server nodes, where every server is a master. There is no limit to the number of running servers. With many servers spread across a slow network, you can tune the network timeouts to be more permissive and let a large, distributed cluster of servers work properly.

For more information, Distributed Architecture.

Multi-Model

The OrientDB engine supports **Graph**, **Document**, **Key/Value**, and **Object** models, so you can use OrientDB as a replacement for a product in any of these categories. However, the main reason why users choose OrientDB is because of its true **Multi-Model** DBMS abilities, which combine all the features of the four models into the core. These abilities are not just interfaces to the database engine, but rather the engine itself was built to support all four models. This is also the main difference to other multi-model DBMSs, as they implement an additional layer with an API, which mimics additional models. However, under the hood, they're truly only one model, therefore they are limited in speed and scalability.

The Document Model

The data in this model is stored inside documents. A document is a set of key/value pairs (also referred to as fields or properties), where the key allows access to its value. Values can hold primitive data types, embedded documents, or arrays of other values. Documents are not typically forced to have a schema, which can be advantageous, because they remain flexible and easy to modify. Documents are stored in collections, enabling developers to group data as they decide. OrientDB uses the concepts of "classes" and "clusters" as its form of "collections" for grouping documents. This provides several benefits, which we will discuss in further sections of the documentation.

OrientDB's Document model also adds the concept of a "LINK" as a relationship between documents. With OrientDB, you can decide whether to embed documents or link to them directly. When you fetch a document, all the links are automatically resolved by OrientDB. This is a major difference to other Document Databases, like MongoDB or CouchDB, where the developer must handle any and all relationships between the documents herself.

The table below illustrates the comparison between the relational model, the document model, and the OrientDB document model:

Relational Model	Document Model	OrientDB Document Model
Table	Collection	Class or Cluster
Row	Document	Document
Column	Key/value pair	Document field
Relationship	not available	Link

The Graph Model

A graph represents a network-like structure consisting of Vertices (also known as Nodes) interconnected by Edges (also known as Arcs). OrientDB's graph model is represented by the concept of a property graph, which defines the following:

- **Vertex** - an entity that can be linked with other Vertices and has the following mandatory properties:
  - unique identifier
  - set of incoming Edges
  - set of outgoing Edges
- **Edge** - an entity that links two Vertices and has the following mandatory properties:
  - unique identifier
  - link to an incoming Vertex (also known as head)
  - link to an outgoing Vertex (also known as tail)
  - label that defines the type of connection/relationship between head and tail vertex

In addition to mandatory properties, each vertex or edge can also hold a set of custom properties. These properties can be defined by users, which can make vertices and edges appear similar to documents. In the table below, you can find a comparison between the graph model, the relational data model, and the OrientDB graph model:

Relational Model	Graph Model	OrientDB Graph Model
Table	Vertex and Edge Class	Class that extends "V" (for Vertex) and "E" (for Edges)
Row	Vertex	Vertex
Column	Vertex and Edge property	Vertex and Edge property
Relationship	Edge	Edge

The Key/Value Model

This is the simplest model of the three. Everything in the database can be reached by a key, where the values can be simple and complex types. OrientDB supports Documents and Graph Elements as values allowing for a richer model, than what you would normally find in the classic Key/Value model. The classic Key/Value model provides "buckets" to group key/value pairs in different containers. The most classic use cases of the Key/Value Model are:

- POST the value as payload of the HTTP call -> `/<bucket>/<key>`
- GET the value as payload from the HTTP call -> `/<bucket>/<key>`
- DELETE the value by Key, by calling the HTTP call -> `/<bucket>/<key>`

The table below illustrates the comparison between the relational model, the Key/Value model, and the OrientDB Key/Value model:

Relational Model	Key/Value Model	OrientDB Key/Value Model
Table	Bucket	Class or Cluster
Row	Key/Value pair	Document
Column	not available	Document field or Vertex/Edge property
Relationship	not available	Link

The Object Model

This model has been inherited by Object Oriented programming and supports **Inheritance** between types (sub-types extends the super-types), **Polymorphism** when you refer to a base class and **Direct binding** from/to Objects used in programming languages.

The table below illustrates the comparison between the relational model, the Object model, and the OrientDB Object model:

Relational Model	Object Model	OrientDB Object Model
Table	Class	Class or Cluster
Row	Object	Document or Vertex
Column	Object property	Document field or Vertex/Edge property
Relationship	Pointer	Link

Basic Concepts

Record

The smallest unit that you can load from and store in the database. Records come in four types:

- Document
- RecordBytes
- Vertex
- Edge

A **Record** is the smallest unit that can be loaded from and stored into the database. A record can be a Document, a RecordBytes record (BLOB) a Vertex or even an Edge.

Document

The Document is the most flexible record type available in OrientDB. Documents are softly typed and are defined by schema classes with defined constraints, but you can also use them in a schema-less mode too.

Documents handle fields in a flexible manner. You can easily import and export them in JSON format. For example,

```
{
     "name"      : "Jay",
     "surname"   : "Miner",
     "job"       : "Developer",
     "creations" : [
          {
               "name"    : "Amiga 1000",
               "company" : "Commodore Inc."
           }, {
               "name"    : "Amiga 500",
               "company" : "Commodore Inc."
           }
     ]
}
```

For Documents, OrientDB also supports complex relationships. From the perspective of developers, this can be understood as a persistent `Map<String,Object>` .

BLOB

In addition to the Document record type, OrientDB can also load and store binary data. The BLOB record type was called `RecordBytes` before OrientDB v2.2.

Vertex

In Graph databases, the most basic unit of data is the node, which in OrientDB is called a vertex. The Vertex stores information for the database. There is a separate record type called the Edge that connects one vertex to another.

Vertices are also documents. This means they can contain embedded records and arbitrary properties.

Edge

In Graph databases, an arc is the connection between two nodes, which in OrientDB is called an edge. Edges are bidirectional and can only connect two vertices.

Edges can be regular or lightweight. The Regular Edge saves as a Document, while the Lightweight Edge does not. For an understanding of the differences between these, see Lightweight Edges.

For more information on connecting vertices in general, see Relationships, below.

Record ID

When OrientDB generates a record, it auto-assigns a unique unit identifier, called a Record ID, or RID. The syntax for the Record ID is the pound sign with the cluster identifier and the position. The format is like this:

`#<cluster>:<position>` .

- **Cluster Identifier**: This number indicates the cluster to which the record belongs. Positive numbers in the cluster identifier indicate persistent records. Negative numbers indicate temporary records, such as those that appear in result-sets for queries that use projections.
- **Position**: This number defines the absolute position of the record in the cluster.

> **NOTE**: The prefix character `#` is mandatory to recognize a Record ID.

Records never lose their identifiers unless they are deleted. When deleted, OrientDB never recycles identifiers, except with `local` storage. Additionally, you can access records directly through their Record ID's. For this reason, you don't need to create a field to serve as the primary key, as you do in Relational databases.

Record Version

Records maintain their own version number, which increments on each update. In optimistic transactions, OrientDB checks the version in order to avoid conflicts at commit time.

Class

The concept of the Class is taken from the Object Oriented Programming paradigm. In OrientDB, classes define records. It is closest to the concept of a table in Relational databases.

Classes can be schema-less, schema-full or a mix. They can inherit from other classes, creating a tree of classes. Inheritance, in this context, means that a sub-class extends a parent class, inheriting all of its attributes.

Each class has its own cluster. A class must have at least one cluster defined, which functions as its default cluster. But, a class can support multiple clusters. When you execute a query against a class, it automatically propagates to all clusters that are part of the class. When you create a new record, OrientDB selects the cluster to store it in using a configurable strategy.

When you create a new class, by default, OrientDB creates a new persistent cluster with the same name as the class, in lowercase.

Abstract Class

The concept of an Abstract Class is one familiar to Object-Oriented programming. In OrientDB, this feature has been available since version 1.2.0. Abstract classes are classes used as the foundation for defining other classes. They are also classes that cannot have instances. For more information on how to create an abstract class, see CREATE CLASS.

This concept is essential to Object Orientation, without the typical spamming of the database with always empty, auto-created clusters.

> For more information on Abstract Class as a concept, see Abstract Type and Abstract Methods and Classes

Class vs. Cluster in Queries

The combination of classes and clusters is very powerful and has a number of use cases. Consider an example where you create a class `Invoice` , with two clusters `invoice2016` and `invoice2017` . You can query all invoices using the class as a target with `SELECT` .

```
orientdb> SELECT FROM Invoice
```

In addition to this, you can filter the result-set by year. The class `Invoice` includes a `year` field, you can filter it through the `WHERE` clause.

```
orientdb> SELECT FROM Invoice WHERE year = 2016
```

You can also query specific objects from a single cluster. By splitting the class `Invoice` across multiple clusters, (that is, one per year), you can optimize the query by narrowing the potential result-set.

```
orientdb> SELECT FROM CLUSTER:invoice2016
```

Due to the optimization, this query runs significantly faster, because OrientDB can narrow the search to the targeted cluster.

Cluster

Where classes in provide you with a logical framework for organizing data, clusters provide physical or in-memory space in which OrientDB actually stores the data. It is comparable to the collection in Document databases and the table in Relational databases.

When you create a new class, the `CREATE CLASS` process also creates a physical cluster that serves as the default location in which to store data for that class. OrientDB forms the cluster name using the class name, with all lower case letters. Beginning with version 2.2, OrientDB creates additional clusters for each class, (one for each CPU core on the server), to improve performance of parallelism.

> For more information, see the Clusters Tutorial.

Relationships

OrientDB supports two kinds of relationships: **referenced** and **embedded**. It can manage relationships in a schema-full or schema-less scenario.

Referenced Relationships

In Relational databases, tables are linked through `JOIN` commands, which can prove costly on computing resources. OrientDB manges relationships natively without computing `JOIN` 's. Instead, it stores direct links to the target objects of the relationship. This boosts the load speed for the entire graph of connected objects, such as in Graph and Object database systems.

For example

```
                 customer
  Record A     ------------->    Record B
CLASS=Invoice                 CLASS=Customer
  RID=5:23                       RID=10:2
```

Here, record `A` contains the reference to record `B` in the property `customer` . Note that both records are reachable by other records, given that they have a Record ID.

With the Graph API, Edges are represented with two links stored on both vertices to handle the bidirectional relationship.

1:1 and *n*:1 Referenced Relationships

OrientDB expresses relationships of these kinds using links of the `LINK` type.

1:*n* and *n*:*n* Referenced Relationships

OrientDB expresses relationships of these kinds using a collection of links, such as:

- `LINKLIST` An ordered list of links.
- `LINKSET` An unordered set of links, which does not accept duplicates.
- `LINKMAP` An ordered map of links, with `String` as the key type. Duplicates keys are not accepted.

With the Graph API, Edges connect only two vertices. This means that 1:*n* relationships are not allowed. To specify a 1:*n* relationship with graphs, create multiple edges.

Embedded Relationships

When using Embedded relationships, OrientDB stores the relationship within the record that embeds it. These relationships are stronger than Reference relationships. You can represent it as a UML Composition relationship.

Embedded records do not have their own Record ID, given that you can't directly reference it through other records. It is only accessible through the container record.

In the event that you delete the container record, the embedded record is also deleted. For example,

```
                 address
  Record A     <>---------->   Record B
CLASS=Account               CLASS=Address
  RID=5:23                     NO RID!
```

Here, record `A` contains the entirety of record `B` in the property `address`. You can reach record `B` only by traversing the container record. For example,

```
orientdb> SELECT FROM Account WHERE address.city = 'Rome'
```

1:1 and *n*:1 Embedded Relationships

OrientDB expresses relationships of these kinds using the `EMBEDDED` type.

1:*n* and *n*:*n* Embedded Relationships

OrientDB expresses relationships of these kinds using a collection of links, such as:

- `EMBEDDEDLIST` An ordered list of records.
- `EMBEDDEDSET` An unordered set of records, that doesn't accept duplicates.
- `EMBEDDEDMAP` An ordered map of records as the value and a string as the key, it doesn't accept duplicate keys.

Inverse Relationships

In OrientDB, all Edges in the Graph model are bidirectional. This differs from the Document model, where relationships are always unidirectional, requiring the developer to maintain data integrity. In addition, OrientDB automatically maintains the consistency of all bidirectional relationships.

Database

The database is an interface to access the real Storage. IT understands high-level concepts such as queries, schemas, metadata, indices and so on. OrientDB also provides multiple database types. For more information on these types, see Database Types.

Each server or Java VM can handle multiple database instances, but the database name must be unique. You can't manage two databases at the same time, even if they are in different directories. To handle this case, use the `$` dollar character as a separator instead of the `/` slash character. OrientDB binds the entire name, so it becomes unique, but at the file system level it converts `$` with `/`, allowing multiple databases with the same name in different paths. For example,

```
test$customers -> test/customers
production$customers = production/customers
```

To open the database, use the following code:

```
test = new ODatabaseDocumentTx("remote:localhost/test$customers");
production = new ODatabaseDocumentTx("remote:localhost/production$customers");
```

Database URL

OrientDB uses its own URL format, of engine and database name as `<engine>:<db-name>`.

Engine	Description	Example
plocal	This engine writes to the file system to store data. There is a LOG of changes to restore the storage in case of a crash.	`plocal:/temp/databases/petshop/petshop`
memory	Open a database completely in memory	`memory:petshop`
remote	The storage will be opened via a remote network connection. It requires an OrientDB Server up and running. In this mode, the database is shared among multiple clients. Syntax: `remote:<server>:[<port>]/db-name` . The port is optional and defaults to 2424.	`remote:localhost/petshop`

Database Usage

You must always close the database once you finish working on it.

> **NOTE**: OrientDB automatically closes all opened databases, when the process dies gracefully (not by killing it by force). This is assured if the Operating System allows a graceful shutdown.

Supported Types

OrientDB supports several types natively. Below is the complete table.

#id	Type	Description	Java type	Minimum Maximum	Auto-conversion from/to
0	Boolean	Handles only the values *True* or *False*	`java.lang.Boolean` or `boolean`	0 1	String
1	Integer	32-bit signed Integers	`java.lang.Integer` or `int`	-2,147,483,648 +2,147,483,647	Any Number, String
2	Short	Small 16-bit signed integers	`java.lang.Short` or `short`	-32,768 32,767	Any Number, String
3	Long	Big 64-bit signed integers	`java.lang.Long` or `long`	-2^{63} $+2^{63}-1$	Any Number, String
4	Float	Decimal numbers	`java.lang.Float` or `float`	2^{-149} $(2-2^{-23})*2^{127}$	Any Number, String
5	Double	Decimal numbers with high precision	`java.lang.Double` or `double`	2^{-1074} $(2-2^{-52})*2^{1023}$	Any Number, String
6	Datetime	Any date with the precision up to milliseconds. To know more about it, look at Managing Dates	`java.util.Date`	- 1002020303	Date, Long, String
7	String	Any string as alphanumeric sequence of chars	`java.lang.String`	- -	-
8	Binary	Can contain any value as byte array	`byte[]`	0 2,147,483,647	String
9	Embedded	The Record is contained inside the owner. The contained Record has no Record ID	`ORecord`	- -	ORecord
10	Embedded list	The Records are contained inside the owner. The contained records have no Record ID's and are reachable only by navigating the owner record	`List<Object>`	0 41,000,000 items	String
11	Embedded set	The Records are contained inside the owner. The contained Records have no Record ID and are reachable only by navigating the owner record	`Set<Object>`	0 41,000,000 items	String
12	Embedded map	The Records are contained inside the owner as values of the entries, while the keys can only be Strings. The contained ords e no Record IDs and are reachable only by navigating the owner Record	`Map<String, ORecord>`	0 41,000,000 items	`Collection<? extends ORecord<?>>` , `String`
13	Link	Link to another Record. It's a common one-to-one relationship	`ORID` , `<? extends ORecord>`	1:-1 32767:2^63-1	String
14	Link list	Links to other Records. It's a common one-to-many relationship where only the Record IDs are stored	`List<? extends ORecord`	0 41,000,000 items	String
		Links to other Records. It's a		0	`Collection<?`

15	Link set	Links to other Records. It's a common one-to-many relationship	`Set<? extends ORecord>`	41,000,000 items	`extends ORecord>` , `String`
16	Link map	Links to other Records as value of the entries, while keys can only be Strings. It's a common One-to-Many Relationship. Only the Record IDs are stored	`Map<String, ? extends Record>`	0 41,000,000 items	String
17	Byte	Single byte. Useful to store small 8-bit signed integers	`java.lang.Byte` or `byte`	-128 +127	Any Number, String
18	Transient	Any value not stored on database			
19	Date	Any date as year, month and day. To know more about it, look at Managing Dates	`java.util.Date`	--	Date, Long, String
20	Custom	used to store a custom type providing the marshall and unmarshall methods	`OSerializableStream`	0 X	-
21	Decimal	Decimal numbers without rounding	`java.math.BigDecimal`	? ?	Any Number, String
22	LinkBag	List of Record IDs as spec RidBag	`ORidBag`	? ?	-
23	Any	Not determinated type, used to specify Collections of mixed type, and null	-	-	-

Inheritance

Unlike many Object-relational mapping tools, OrientDB does not split documents between different classes. Each document resides in one or a number of clusters associated with its specific class. When you execute a query against a class that has subclasses, OrientDB searches the clusters of the target class and all subclasses.

Declaring Inheritance in Schema

In developing your application, bear in mind that OrientDB needs to know the class inheritance relationship. This is an abstract concept that applies to both POJO's and Documents.

For example,

```
OClass account = database.getMetadata().getSchema().createClass("Account");
OClass company = database.getMetadata().getSchema().createClass("Company").setSuperClass(account);
```

Using Polymorphic Queries

By default, OrientDB treats all queries as polymorphic. Using the example above, you can run the following query from the console:

```
orientdb> SELECT FROM Account WHERE name.toUpperCase() = 'GOOGLE'
```

This query returns all instances of the classes `Account` and `Company` that have a property name that matches `Google`.

How Inheritance Works

Consider an example, where you have three classes, listed here with the cluster identifier in the parentheses.

```
Account(10) <|--- Company (13) <|--- OrientTechnologiesGroup (27)
```

By default, OrientDB creates a separate cluster for each class. It indicates this cluster by the `defaultClusterId` property in the class `OClass` and indicates the cluster used by default when not specified. However, the class `OClass` has a property `clusterIds`, (as `int[]`), that contains all the clusters able to contain the records of that class. `clusterIds` and `defaultClusterId` are the same by default.

When you execute a query against a class, OrientDB limits the result-sets to only the records of the clusters contained in the `clusterIds` property. For example,

```
orientdb> SELECT FROM Account WHERE name.toUpperCase() = 'GOOGLE'
```

This query returns all the records with the name property set to `GOOGLE` from all three classes, given that the base class `Account` was specified. For the class `Account`, OrientDB searches inside the clusters `10`, `13` and `27`, following the inheritance specified in the schema.

Concurrency

OrientDB uses an optimistic approach to concurrency. Optimistic Concurrency Control, or OCC assumes that multiple transactions can compete frequently without interfering with each other. It's very important that you don't share instances of databases, graphs, records, documents, vertices and edges between threads because they are non thread-safe. For more information look at Multi-Threading.

How does it work?

Consider the following scenario, where 2 clients, A and B, want to update the amount of a bank account:

```
        Client A                              Client B

            |                                     |
           (t1)                                   |
     Read record #13:22                           |
       amount is 100                             (t2)
            |                             Read record #13:22
           (t3)                             amount is 100
    Update record #13:22                          |
  set amount = amount + 10                       (t4)
            |                            Update record #13:22
            |                          set amount = amount + 10
            |                                     |
```

Client A (t1) and B (t2) read the record #13:22 and both receive the last amount as USD 100. Client A updates the amount by adding USD 10 (t3), then the Client B is trying to do the same thing: updates the amount by adding USD 10. Here is the problem: Client B is doing an operation based on current information: the amount was USD 100. But at the moment of update, such information is changed (by Client A on t3), so the amount is USD 110 in the database. Should the update succeed by setting the new amount to USD 120?

In some cases this could be totally fine, in others not. It depends by the use case. For example, in your application there could be a logic where you are donating USD 10 to all the accounts where the amount is <=100. The owner of the account behind the record #13:22 is more lucky than the others, because it receives the donation even if it has USD 110 at that moment.

For this reason in OrientDB when this situation happens a `OConcurrentModificationException` exception is thrown, so the application can manage it properly. Usually the 3 most common strategies to handle this exceptions are:

1. **Retry** doing the same operation by reloading the record #13:22 first with the updated amount
2. **Ignore** the change, because the basic condition is changed
3. **Propagate the exception to the user**, so he can decide what to do in this case

Optimistic Concurrency in OrientDB

Optimistic concurrency control is used in environments with low data contention. That is, where conflicts are rare and transactions can complete without the expense of managing locks and without having transactions wait for locks to clear. This means an increased throughput over other concurrency control methods.

OrientDB uses OCC for both Atomic Operations and Transactions.

Atomic Operations

OrientDB supports Multi-Version Concurrency Control, or MVCC, with atomic operations. This allows it to avoid locking server side resources. At the same time, it checks the version in the database. If the version is equal to the record version contained in the operation, the operation is successful. If the version found is higher than the record version contained in the operation, then another thread or user has already updated the same record. In this case, OrientDB generates an `OConcurrentModificationException` exception.

Given that behavior of this kind is normal on systems that use optimistic concurrency control, developers need to write concurrency-proof code. Under this design, the application retries transactions *x* times before reporting the error. It does this by catching the exception, reloading the affected records and attempting to update them again. For example, consider the code for saving a document,

```
int maxRetries = 10;
List<ODocument> result = db.query("SELECT FROM Client WHERE id = '39w39D32d2d'");
ODocument address = result.get(0);

for (int retry = 0; retry < maxRetries; ++retry) {
     try {
          // LOOKUP FOR THE INVOICE VERTEX
          address.field( "street", street );
          address.field( "zip", zip );
          address.field( "city", cityName );
          address.field( "country", countryName );

          address.save();

          // EXIT FROM RETRY LOOP
          break;
     }
     catch( ONeedRetryException e ) {
          // IF SOMEONE UPDATES THE ADDRESS DOCUMENT
          // AT THE SAME TIME, RETRY IT.
     }
}
```

Transactions

OrientDB supports optimistic transactions. The database does not use locks when transactions are running, but when the transaction commits, each record (document or graph element) version is checked to see if there have been updates from another client. For this reason, you need to code your applications to be concurrency-proof.

Optimistic concurrency requires that you retry the transaction in the event of conflicts. For example, consider a case where you want to connect a new vertex to an existing vertex:

```
int maxRetries = 10;
for (int retry = 0; retry < maxRetries; ++retry) {
     try {
          // LOOKUP FOR THE INVOICE VERTEX
          Vertex invoice = graph.getVertices("invoiceId", 2323);

          // CREATE A NEW ITEM
          Vertex invoiceItem = graph.addVertex("class:InvoiceItem");
          invoiceItem.field("price", 1000);

          // ADD IT TO THE INVOICE
          invoice.addEdge(invoiceItem);

          graph.commit();

          // EXIT FROM RETRY LOOP
          break;
     }
     catch( OConcurrentModificationException e ) {
          // SOMEONE HAS UPDATED THE INVOICE VERTEX
          // AT THE SAME TIME, RETRY IT
     }
}
```

Concurrency Level

In order to guarantee atomicity and consistency, OrientDB uses an exclusive lock on the storage during transaction commits. This means that transactions are serialized.

Given this limitation, developers with OrientDB are working on improving parallelism to achieve better scalability on multi-core machines, by optimizing internal structure to avoid exclusive locking.

Concurrency when Adding Edges

Consider the case where multiple clients attempt to add edges on the same vertex. OrientDB could throw the `OConcurrentModificationException` exception. This occurs because collections of edges are kept on vertices, meaning that, every time OrientDB adds or removes an edge, both vertices update and their versions increment. You can avoid this issue by using RIDBAG Bonsai structure, which are never embedded, so the edge never updates the vertices.

To use this configuration at run-time, before launching OrientDB, use this code:

```
OGlobalConfiguration.RID_BAG_EMBEDDED_TO_SBTREEBONSAI_THRESHOLD.setValue(-1);
```

Alternatively, you can set a parameter for the Java virtual-machine on startup, or even at run-time, before OrientDB is used:

```
$ java -DridBag.embeddedToSbtreeBonsaiThreshold=-1
```

While running in distributed mode SBTrees are not supported. If using a distributed database then you must set

```
ridBag.embeddedToSbtreeBonsaiThreshold = Integer.MAX\_VALUE
```

to avoid replication errors.

Troubleshooting

Reduce Transaction Size

On occasion, OrientDB throws the `OConcurrentModificationException` exception even when you concurrently update the first element. In particularly large transactions, where you have thousands of records involved in a transaction, one changed record is enough to roll the entire process back with an `OConcurrentModificationException` exception.

To avoid issues of this kind, if you plan to update many elements in the same transaction with high-concurrency on the same vertices, a best practice is to reduce the transaction size.

Schema

While OrientDb can work in a schema-less mode, you may find it necessary at times to enforce a schema on your data model. OrientDB supports both schema-full and schema-hybrid solutions.

In the case of schema-hybrid mode, you only set constraints for certain fields and leave the user to add custom fields to the record. This mode occurs at a class level, meaning that you can have an `Employee` class as schema-full and an `EmployeeInformation` class as schema-less.

- **Schema-full** Enables strict-mode at a class-level and sets all fields as mandatory.
- **Schema-less** Enables classes with no properties. Default is non-strict-mode, meaning that records can have arbitrary fields.
- **Schema-hybrid** Enables classes with some fields, but allows records to define custom fields. This is also sometimes called schema-mixed.

> **NOTE** Changes to the schema are not transactional. You must execute these commands outside of a transaction.

You can access the schema through SQL or through the Java API. Examples here use the latter. To access the schema API in Java, you need the Schema instance of the database you want to use. For example,

```
OSchema schema = database.getMetadata().getSchema();
```

Class

OrientDB draws from the Object Oriented programming paradigm in the concept of the Class. A class is a type of record. In comparison to Relational database systems, it is most similar in conception to the table.

Classes can be schema-less, schema-full or schema-hybrid. They can inherit from other classes, shaping a tree of classes. In other words, a sub-class extends the parent class, inheriting all attributes.

Each class has its own clusters. By default, these clusters are logical, but they can also be physical. A given class must have at least one cluster defined as its default, but it can support multiple clusters. OrientDB writes new records into the default cluster, but always reads from all defined clusters.

When you create a new class, OrientDB creates a default physical cluster that uses the same name as the class, but in lowercase.

Creating Persistent Classes

Classes contain one or more properties. This mode is similar to the classical model of the Relational database, where you must define tables before you can begin to store records.

To create a persistent class in Java, use the `createClass()` method:

```
OClass account = database.getMetadata().getSchema().createClass("Account");
```

This method creates the class `Account` on the database. It simultaneously creates the physical cluster `account`, to provide storage for records in the class `Account`.

Getting Persistent Classes

With the new persistent class created, you may also need to get its contents.

To retrieve a persistent class in Java, use the `getClass()` method:

```
OClass account = database.getMetadata().getSchema().getClass("Account");
```

This method retrieves from the database the persistent class `Account`. If the query finds that the `Account` class does not exist, it returns `NULL`.

Dropping Persistent Classes

In the event that you no longer want the class, you can drop, or delete, it from the database.

To drop a persistent class in Java, use the `OSchema.dropClass()` method:

```
database.getMetadata().getSchema().dropClass("Account");
```

This method drops the class `Account` from your database. It does not delete records that belong to this class unless you explicitly ask it to do so:

```
database.command(new OCommandSQL("DELETE FROM Account")).execute();
database.getMetadata().getSchema().dropClass("Account");
```

Constraints

Working in schema-full mode requires that you set the strict mode at the class-level, by defining the `setStrictMode()` method to `TRUE`. In this case, records of that class cannot have undefined properties.

Properties

In OrientDB, a property is a field assigned to a class. For the purposes of this tutorial, consider Property and Field as synonymous.

Creating Class Properties

After you create a class, you can define fields for that class. To define a field, use the `createProperty()` method.

```
OClass account = database.getMetadata().getSchema().createClass("Account");
account.createProperty("id", OType.Integer);
account.createProperty("birthDate", OType.Date);
```

These lines create a class `Account`, then defines two properties `id` and `birthDate`. Bear in mind that each field must belong to one of the supported types. Here these are the integer and date types.

Dropping Class Properties

In the event that you would like to remove properties from a class you can do so using the `dropProperty()` method under `OClass`.

```
database.getMetadata().getSchema().getClass("Account").dropProperty("name");
```

When you drop a property from a class, it does not remove records from that class unless you explicitly ask for it, using the `UPDATE... REMOVE` statements. For instance,

```
database.getMetadata().getSchema().getClass("Account").dropProperty("name");
database.command(new OCommandSQL("UPDATE Account REMOVE name")).execute();
```

The first method drops the property from the class. The second updates the database to remove the property.

Relationships

OrientDB supports two types of relationships: referenced and embedded.

Referenced Relationships

In the case of referenced relationships, OrientDB uses a direct link to the referenced record or records. This allows the database to avoid the costly `JOIN` operations used by Relational databases.

```
                  customer
  Record A     ------------->    Record B
CLASS=Invoice                 CLASS=Customer
  RID=5:23                       RID=10:2
```

In the example, Record A contains the reference to Record B in the property `customer`. Both records are accessible by any other records since each has a Record ID.

1:1 and *n*:1 Reference Relationships

In one to one and many to one relationships, the reference relationship is expressed using the `LINK` type. For instance.

```
OClass customer= database.getMetadata().getSchema().createClass("Customer");
customer.createProperty("name", OType.STRING);

OClass invoice = database.getMetadata().getSchema().createClass("Invoice");
invoice.createProperty("id", OType.INTEGER);
invoice.createProperty("date", OType.DATE);
invoice.createProperty("customer", OType.LINK, customer);
```

Here, records of the class `Invoice` link to a record of the class `Customer`, through the field `customer`.

1:*n* and *n*:*n* Reference Relationships.

In one to many and many to many relationships, OrientDB expresses the referenced relationship using collections of links.

- `LINKLIST` An ordered list of links.
- `LINKSET` An unordered set of links, that does not accept duplicates.
- `LINKMAP` An ordered map of links, with a string key. It does not accept duplicate keys.

For example,

```
OClass orderItem = db.getMetadata().getSchema().createClass("OrderItem");
orderItem.createProperty("id", OType.INTEGER);
orderItem.createProperty("animal", OType.LINK, animal);

OClass order = db.getMetadata().getSchema().createClass("Order");
order.createProperty("id", OType.INTEGER);
order.createProperty("date", OType.DATE);
order.createProperty("items", OType.LINKLIST, orderItem);
```

Here, you have two classes: `Order` and `OrderItem` and a 1:*n* referenced relationship is created between them.

Embedded Relationships

In the case of embedded relationships, OrientDB contains the relationship within the record. Embedded relationships are stronger than referenced relationships, but the embedded record does not have its own Record ID. Because of this, you cannot reference them directly through other records. The relationship is only accessible through the container record. If the container record is deleted, then the embedded record is also deleted.

```
                  address
  Record A     <>---------->   Record B
CLASS=Account               CLASS=Address
  RID=5:23                     NO RID!
```

Here, Record A contains the entirety of Record B in the property `address`. You can only reach Record B by traversing the container, Record A.

```
orientdb> SELECT FROM Account WHERE Address.city = 'Rome'
```

1:1 and *n*:1 Embedded Relationships

For one to one and many to one embedded relationships, OrientDB uses links of the `EMBEDDED` type. For example,

```
OClass address = database.getMetadata().getSchema().createClass("Address");

OClass account = database.getMetadata().getSchema().createClass("Account");
account.createProperty("id", OType.INTEGER);
account.createProperty("birthDate", OType.DATE);
account.createProperty("address", OType.EMBEDDED, address);
```

Here, records of the class `Account` embed records for the class `Address` .

1:*n* and *n*:*n* Embedded Relationships

In the case of one to many and many to many relationships, OrientDB sues a collection embedded link types:

- `EMBEDDEDLIST` An ordered list of records.
- `EMBEDDEDSET` An unordered set of records. It doesn't accept duplicates.
- `EMBEDDEDMAP` An ordered map of records as key-value pairs. It doesn't accept duplicate keys.

For example,

```
OClass orderItem = db.getMetadata().getSchema().createClass("OrderItem");
orderItem.createProperty("id", OType.INTEGER);
orderItem.createProperty("animal", OType.LINK, animal);

OClass order = db.getMetadata().getSchema().createClass("Order");
order.createProperty("id", OType.INTEGER);
order.createProperty("date", OType.DATE);
order.createProperty("items", OType.EMBEDDEDLIST, orderItem);
```

This establishes a one to many relationship between the classes `Order` and `OrderItem` .

Constraints

OrientDB supports a number of constraints for each field. For more information on setting constraints, see the `ALTER PROPERTY` command.

- **Minimum Value**: `setMin()` The field accepts a string, because it works also for date ranges.
- **Maximum Value**: `setMax()` The field accepts a string, because it works also for date rangers.
- **Mandatory**: `setMandatory()` This field is required.
- **Read Only**: `setReadonly()` This field cannot update after being created.
- **Not Null**: `setNotNull()` This field cannot be null.
- **Unique**: This field doesn't allow duplicates or speedup searches.
- **Regex**: This field must satisfy Regular Expressions

For example,

```
profile.createProperty("nick", OType.STRING).setMin("3").setMax("30").setMandatory(true).setNotNull(true);
profile.createIndex("nickIdx", OClass.INDEX_TYPE.UNIQUE, "nick"); // Creates unique constraint

profile.createProperty("name", OType.STRING).setMin("3").setMax("30");
profile.createProperty("surname", OType.STRING).setMin("3").setMax("30");
profile.createProperty("registeredOn", OType.DATE).setMin("2010-01-01 00:00:00");
profile.createProperty("lastAccessOn", OType.DATE).setMin("2010-01-01 00:00:00");
```

Indices as Constraints

To define a property value as unique, use the `UNIQUE` index constraint. For example,

```
profile.createIndex("EmployeeId", OClass.INDEX_TYPE.UNIQUE, "id");
```

You can also constrain a group of properties as unique by creating a composite index made from multiple fields. For instance,

```
profile.createIndex("compositeIdx", OClass.INDEX_TYPE.NOTUNIQUE, "name", "surname");
```

For more information about indexes look at Index guide.

Graph or Document API?

In OrientDB, we created 2 different APIs: the Document API and the Graph API. The Graph API works on top of the Document API. The Document API contains the Document, Key/Value and Object Oriented models. The Graph API handles the Vertex and Edge relationships.

```
         YOU, THE USER

    ||                 ||
   _||_                ||
   \  /                ||
    \/                _||_
+-------------+       \  /
|  Graph API  |        \/
+-------------+-----------------+
|         Document API          |
+-------------------------------+
| Key/Value and Object Oriented |
+-------------------------------+
```

Graph API

With OrientDB 2.0, we improved our Graph API to support all models in just one Multi-Model API. This API will probably cover 80% of your database use cases, so it should be your "go to" API, if you're starting with OrientDB.

Using the Graph API:

- Your Data ('records' in the RDBMS world) will be modeled as Vertices and Edges. You can store properties in both.
- You can still work in Schema-Less, Schema-Full or Hybrid modes.
- Relationships are modeled as Bidirectional Edges. If the Lightweight edge setting is active, OrientDB uses Lightweight Edges in cases where edges have no properties, so it has the same impact on speed and space as with Document LINKs, but with the additional bonus of having bidirectional connections. This means you can use the `MOVE VERTEX` command to refactor your graph with no broken LINKs. For more information how Edges are managed, please refer to Lightweight Edges.

Document API

What about the remaining 20% of your database use cases? Should you need a Document Database (while retaining the additional OrientDB features, like LINKs) or you come from the Document Database world, using the Document API could be the right choice.

These are the Pros and Cons of using the Document API:

- The Document API is simpler than the Graph API in general.
- Relationships are only mono-directional. If you need bidirectional relationships, it is your responsibility to maintain both LINKs.
- A Document is an atomic unit, while with Graphs, the relationships are modeled through In and Out properties. For this reason, Graph operations must be done within transactions. In contrast, when you create a relationship between documents with a LINK, the targeted linked document is not involved in this operation. This results in better Multi-Threaded support, especially with insert, delete and update operations.

Cluster Selection

When you create a new record and specify the class to which it belongs, OrientDB automatically selects a cluster, where it stores the physical data of the record. There are a number of configuration strategies available for you to use in determining how OrientDB selects the appropriate cluster for the new record.

- `default` It selects the cluster using the `defaultClusterId` property from the class. Prior to version 1.7, this was the default method.
- `round-robin` It arranges the configured clusters for the class into sequence and assigns each new record to the next cluster in order.
- `balanced` It checks the number of records in the configured clusters for the class and assigns the new record to whichever is the smallest at the time. To avoid latency issues on data insertions, OrientDB calculates cluster size every five seconds or longer.
- `local` When the database is run in distributed mode, it selects the master cluster on the current node. This helps to avoid conflicts and reduce network latency with remote calls between nodes.

In distributed mode the local cluster strategy is always selected automatically and can't be changed. The local strategy acts as a wrapper for the underlying strategy (round-robin by default) by filtering the allowed clusters by selecting only those the local server is a master.

But in Studio is never displayed properly, because the underlying name is taken.

Whichever cluster selection strategy works best for your application, you can assign it through the `ALTER CLASS...CLUSTERSELECTION` command. For example,

```
orientdb> ALTER CLASS Account CLUSTERSELECTION round-robin
```

When you run this command, it updates the `Account` class to use the `round-robin` selection strategy. It cycles through available clusters, adding new records to each in sequence.

Custom Cluster Selection Strategies

In addition to the cluster selection strategies listed above, you can also develop your own select strategies through the Java API. This ensures that the strategies that are available by default do not meet your particular needs, you can develop one that does.

1. Using your preferred text editor, create the implementation in Java. In order to use a custom strategy, the class must implement the `OClusterSelectionStrategy` interface.

   ```
   package mypackage;
   public class RandomSelectionStrategy implements OClusterSelectionStrategy {
      public int getCluster(final OClass iClass, final ODocument doc) {
         final int[] clusters = iClass.getClusterIds();

         // RETURN A RANDOM CLUSTER ID IN THE LIST
         int r = new Random().nextInt(clusters.length);
           return clusters[r];
      }

      public String getName(){ return "random"; }
   }
   ```

   Bear in mind that the method `getCluster()` also receives the `ODocument` cluster to insert. You may find this useful, if you want to assign the `clusterId` variable, based on the Document content.

2. Register the implementation as a service. You can do this by creating a new file under `META-INF/services`. Use the filename `com.orientechnologies.orient.core.metadata.schema.clusterselection.OClusterSelectionStrategy`. For its contents, code your class with the full package. For instance,

   ```
   mypackage.RandomSelectionStrategy
   ```

This adds to the default content in the OrientDB core:

```
com.orientechnologies.orient.core.metadata.schema.clusterselection.ORoundRobinClusterSelectionStrategy
com.orientechnologies.orient.core.metadata.schema.clusterselection.ODefaultClusterSelectionStrategy
com.orientechnologies.orient.core.metadata.schema.clusterselection.OBalancedClusterSelectionStrategy
```

3. From the database console, assign the new selection strategy to your class with the `ALTER CLASS...CLUSTERSELECTION` command.

```
orientdb> ALTER CLASS Employee CLUSTERSELECTION random
```

The class `Employee` now selects clusters using `random`, your custom strategy.

Managing Dates

OrientDB treats dates as first class citizens. Internally, it saves dates in the Unix time format. Meaning, it stores dates as a `long` variable, which contains the count in milliseconds since the Unix Epoch, (that is, 1 January 1970).

Date and Datetime Formats

In order to make the internal count from the Unix Epoch into something human readable, OrientDB formats the count into date and datetime formats. By default, these formats are:

- Date Format: `yyyy-MM-dd`
- Datetime Format: `yyyy-MM-dd HH:mm:ss`

In the event that these default formats are not sufficient for the needs of your application, you can customize them through `ALTER DATABASE...DATEFORMAT` and `DATETIMEFORMAT` commands. For instance,

```
orientdb> ALTER DATABASE DATEFORMAT "dd MMMM yyyy"
```

This command updates the current database to use the English format for dates. That is, 14 Febr 2015.

SQL Functions and Methods

To simplify the management of dates, OrientDB SQL automatically parses dates to and from strings and longs. These functions and methods provide you with more control to manage dates:

SQL	Description
`DATE()`	Function converts dates to and from strings and dates, also uses custom formats.
`SYSDATE()`	Function returns the current date.
`.format()`	Method returns the date in different formats.
`.asDate()`	Method converts any type into a date.
`.asDatetime()`	Method converts any type into datetime.
`.asLong()`	Method converts any date into long format, (that is, Unix time).

For example, consider a case where you need to extract only the years for date entries and to arrange them in order. You can use the `.format()` method to extract dates into different formats.

```
orientdb> SELECT @RID, id, date.format('yyyy') AS year FROM Order

--------+----+------+
 @RID   | id | year |
--------+----+------+
 #31:10 | 92 | 2015 |
 #31:10 | 44 | 2014 |
 #31:10 | 32 | 2014 |
 #31:10 | 21 | 2013 |
--------+----+------+
```

In addition to this, you can also group the results. For instance, extracting the number of orders grouped by year.

```
orientdb> SELECT date.format('yyyy') AS Year, COUNT(*) AS Total
          FROM Order ORDER BY Year

------+--------+
 Year |  Total |
------+--------+
 2015 |      1 |
 2014 |      2 |
 2013 |      1 |
------+--------+
```

Dates before 1970

While you may find the default system for managing dates in OrientDB sufficient for your needs, there are some cases where it may not prove so. For instance, consider a database of archaeological finds, a number of which date to periods not only before 1970 but possibly even before the Common Era. You can manage this by defining an era or epoch variable in your dates.

For example, consider an instance where you want to add a record noting the date for the foundation of Rome, which is traditionally referred to as April 21, 753 BC. To enter dates before the Common Era, first run the [`ALTER DATABASE DATETIMEFORMAT`] command to add the `GG` variable to use in referencing the epoch.

```
orientdb> ALTER DATABASE DATETIMEFORMAT "yyyy-MM-dd HH:mm:ss GG"
```

Once you've run this command, you can create a record that references date and datetime by epoch.

```
orientdb> CREATE VERTEX V SET city = "Rome", date = DATE("0753-04-21 00:00:00 BC")
orientdb> SELECT @RID, city, date FROM V

-------+------+------------------------+
 @RID  | city | date                   |
-------+------+------------------------+
 #9:10 | Rome | 0753-04-21 00:00:00 BC |
-------+------+------------------------+
```

Using `.format()` on Insertion

In addition to the above method, instead of changing the date and datetime formats for the database, you can format the results as you insert the date.

```
orientdb> CREATE VERTEX V SET city = "Rome", date = DATE("yyyy-MM-dd HH:mm:ss GG")
orientdb> SELECT @RID, city, date FROM V

------+------+------------------------+
 @RID | city | date                   |
------+------+------------------------+
 #9:4 | Rome | 0753-04-21 00:00:00 BC |
------+------+------------------------+
```

Here, you again create a vertex for the traditional date of the foundation of Rome. However, instead of altering the database, you format the date field in `CREATE VERTEX` command.

Viewing Unix Time

In addition to the formatted date and datetime, you can also view the underlying count from the Unix Epoch, using the `asLong()` method for records. For example,

```
orientdb> SELECT @RID, city, date.asLong() FROM #9:4

------+------+------------------------+
 @RID | city | date                   |
------+------+------------------------+
 #9:4 | Rome | -85889120400000        |
------+------+------------------------+
```

Meaning that, OrientDB represents the date of April 21, 753 BC, as -85889120400000 in Unix time. You can also work with dates directly as longs.

```
orientdb> CREATE VERTEX V SET city = "Rome", date = DATE(-85889120400000)
orientdb> SELECT @RID, city, date FROM V

-------+------+------------------------+
 @RID  | city | date                   |
-------+------+------------------------+
 #9:11 | Rome | 0753-04-21 00:00:00 BC |
-------+------+------------------------+
```

Use ISO 8601 Dates

According to ISO 8601, Combined date and time in UTC: 2014-12-20T00:00:00. To use this standard change the datetimeformat in the database:

```
ALTER DATABASE DATETIMEFORMAT "yyyy-MM-dd'T'HH:mm:ss.SSS'Z'"
```

Graph Consistency

Before OrientDB v2.1.7, the graph consistency could be assured only by using transactions. The problems with using transactions for simple operations like creation of edges are:

- **speed**, the transaction has a cost in comparison with non-transactional operations
- management of **optimistic retry** at application level. Furthermore, with 'remote' connections this means high latency
- **low scalability on high concurrency** (this will be resolved in OrientDB v3.0, where commits will not lock the database anymore)

As of v2.1.7, OrientDB provides a new mode to manage graphs without using transactions. It uses the Java class `OrientGraphNoTx` or via SQL by changing the global setting `sql.graphConsistencyMode` to one of the following values:

- `tx` , the default, uses transactions to maintain consistency. This was the only available setting before v2.1.7
- `notx_sync_repair` , avoids the use of transactions. Consistency, in case of a JVM crash, is guaranteed through a database repair operation, which runs at startup in synchronous mode. The database cannot be used until the repair is finished.
- `notx_async_repair` , also avoids the use of transactions. Consistency, in case of JVM crash, is guaranteed through a database repair operation, which runs at startup in asynchronous mode. The database can be used immediately, as the repair procedure will run in the background.

Both the new modes `notx_sync_repair` and `notx_async_repair` will manage conflicts automatically, with a configurable RETRY (default=50). In case changes to the graph occur concurrently, any conflicts are caught transparently by OrientDB and the operations are repeated. The operations that support the auto-retry are:

- `CREATE EDGE`
- `DELETE EDGE`
- `DELETE VERTEX`

Usage

To use consistency modes that don't use transactions, set the `sql.graphConsistencyMode` global setting to `notx_sync_repair` or `notx_async_repair` in OrientDB `bin/server.sh` script or in the `config/orientdb-server-config.xml` file under properties section. Example:

```
...
<properties>
  ...
  <entry name="sql.graphConsistencyMode" value="notx_sync_repair"/>
  ...
</properties>
```

The same could be set by code, before you open any Graph. Example:

```
OGlobalConfiguration.SQL_GRAPH_CONSISTENCY_MODE.setValue("notx_sync_repair");
```

To make this setting persistent, set the `txRequiredForSQLGraphOperations` property in the storage configuration, so during the following opening of the Graph, you don't need to set the global setting again:

```
g.getRawGraph().getStorage().getConfiguration().setProperty("txRequiredForSQLGraphOperations", "false");
```

Usage via Java API

In order to use non-transactional graphs, after having configured the consistency mode (as above), you can now work with the `OrientGraphNoTx` class. Example:

```
OrientGraphNoTx g = new OrientGraphNoTx("plocal:/temp/mydb");
...
v1.addEdge( "Friend", v2 );
```

Concurrent threads that change the graph will retry the graph change in case of concurrent modification (MVCC). The default value for maximum retries is 50. To change this value, call the `setMaxRetries()` API:

```
OrientGraphNoTx g = new OrientGraphNoTx("plocal:/temp/mydb");
g.setMaxRetries(100);
```

This setting will be used on the active graph instance. You can have multiple threads, which work on the same graph by using multiple graph instances, one per thread. Each thread can then have different settings. It's also allowed to wirk with threads, which use transactions (`OrientGraph` class) and to work with concurrent threads, which don't use transactions.

Fetching Strategies

Fetchplans are used in two different scopes:

1. A Connection that uses the Binary Protocol can *early load* records to the client. On traversing of connected records, the client doesn't have to execute further remote calls to the server, because the requested records are already in the client's cache.
2. A Connection that uses the HTTP/JSON Protocol can *expand the resulting JSON* to include connected records as embedded in the same JSON. This is useful with the HTTP protocol to fetch all the connected records in just one call.

Format for Fetch Plans

In boths scopes, the fetchplan syntax is the same. In terms of their use, Fetch Plans are strings that you can use at run-time on queries and record loads. The syntax for these strings is:

```
[[levels]]fieldPath:depthLevel
```

- **Levels** Is an optional value that indicates which levels to use with the Fetch Plans. Levels start from `0`. As of version 2.1, levels use the following syntax:
  - *Level* The specific level on which to use the Fetch Plan. For example, using the level `[0]` would apply only to the first level.
  - *Range* The range of levels on which to use the Fetch Plan. For example, `[0-2]` means to use it on the first through third levels. You can also use the partial range syntax: `[-3]` which means from the first to fourth levels, while `[4-]` means from the fifth level to infinity.
  - *Any* The wildcard variable indicates that you want to use the Fetch Plan on all levels. For example, `[*]`.
- **Field Path** Is the field name path, which OrientDB expects in dot notation. The path begins from either the root record or the wildcard variable `*` to indicate any field. You can also use the wildcard at the end of the path to specify all paths that start for a name.
- **Depth Level** Is the depth of the level requested. The depth level variable uses the following syntax:
  - `0` Indicates to load the current record.
  - `1-N` Indicates to load the current record to the *n*th record.
  - `-1` Indicates an unlimited level.
  - `-2` Indicates an excluded level.

In the event that you want to express multiple rules for your Fetch Plans, separate them by spaces.

Consider the following Fetch Plans for use with the example above:

Fetch Plan	Description
`*:-1`	Fetches recursively the entire tree.
`*:-1 orders:0`	Fetches recursively all records, but uses the field `orders` in the root class. Note that the field `orders` only loads its direct content, (that is, the records `8:12`, `8:19`, and `8:23`). No other records inside of them load.
`*:0 address.city.country:0`	Fetches only non-document fields in the root class and the field `address.city.country`, (that is, records `10:1`, `11:2` and `12:3`).
`[*]in_*:-2 out_*:-2`	Fetches all properties, except for edges at any level.

Early loading of records

By default, OrientDB loads linked records in a lazy manner. That is to say, it does not load linked fields until it traverses these fields. In situations where you need the entire tree of a record, this can prove costly to performance. For instance,

```
Invoice
 3:100
   |
   | customer
   +---------> Customer
   |            5:233
   | address            city            country
   +---------> Address---------> City ---------> Country
   |            10:1             11:2             12:3
   |
   | orders
   +--------->* [OrderItem OrderItem OrderItem]
                [  8:12      8:19      8:23   ]
```

Here, you have a class, `Invoice`, with linked fields `customer`, `city`, and `orders`. If you were to run a `SELECT` query on `Invoice`, it would not load the linked class, and it would require seven different loads to build the returned value. In the event that you have a remote connection, that means seven network calls as well.

In order to avoid performance issues that may arise from this behavior, OrientDB supports fetching strategies, called Fetch Plans, that allow you to customize how it loads linked records. The aim of a Fetch Plan is to pre-load connected records in a single call, rather than several. The best use of Fetch Plans is on records loaded through remote connections and when using JSON serializers to produce JSON with nested records.

> **NOTE** OrientDB handles circular dependencies to avoid any loops while it fetches linking records.

Remote Connections

Under the default configuration, when a client executes a query or loads a single record directly from a remote database, it continues to send network calls for each linked record involved in the query, (that is, through `OLazyRecordList`). You can mitigate this with a Fetch Plan.

When the client executes a query, set a Fetch Plan with a level different from `0`. This causes the server to traverse all the records of the return result-set, sending them in response to a single call. OrientDB loads all connected records into the local client, meaning that the collections remain lazy, but when accessing content, the record is loaded from the local cache to mitigate the need for additional connections.

Examples using SQL

Acquire Profile and it's first level friendships

```
SELECT OUT("out_Friend") as friends FROM Profile fetchplan friends:1
```

This will provide a result set of Profile records with a field called `friends` that contains an array of verticies connected via a `out_Friend` edge. Only the `friends` field is apart of the fetchplan in this instance, any further will be treated as normal.

Examples using the Java APIs

Execute a query with a custom fetch plan

```
List<ODocument> resultset = database.query(new OSQLSynchQuery<ODocument>("select * from Profile").setFetchPlan("*:-1"));
```

Export a document and its nested documents in JSON

Export an invoice and its customer:

```
invoice.toJSON("fetchPlan:customer:1");
```

Export an invoice, its customer, and orders:

```
invoice.toJSON("fetchPlan:customer:1 orders:2");
```

Export an invoice and all the connected records up to 3rd level of depth:

```
invoice.toJSON("fetchPlan:*:3");
```

From SQL:

```
SELECT @this.toJSON('fetchPlan:out_Friend:4') FROM #10:20
```

Export path in outgoing direction by removing all the incoming edges by using wildcards (Since 2.0):

```
SELECT @this.toJSON('fetchPlan:in_*:-2') FROM #10:20
```

> **NOTES:**:
>
> - To avoid looping, the record already traversed by fetching are exported only by their RIDs (RecordID) form
> - "fetchPlan" setting is case sensitive

Browse objects using a custom fetch plan

```
for (Account a : database.browseClass(Account.class).setFetchPlan("*:0 addresses:-1")) {
  System.out.println( a.getName() );
}
```

> **NOTE:** Fetching Object will mean their presence inside your domain entities. So if you load an object using fetchplan `*:0` all LINK type references won't be loaded.

Use Cases

This page contains the solution to the most common use cases. Please don't consider them as the definitive solution, but as suggestions where to get the idea to solve your needs.

Use cases

- Time Series
- Chat
- Use OrientDB as a Key/Value DBMS
- Persistent, Distributed and Transactional Queues

Time Series Use Case

Managing records related to historical information is pretty common. When you have millions of records, indexes start show their limitations, because the cost to find the records is O(logN). This is also the main reason why Relational DBMS are so slow with huge databases.

So when you have millions of record the best way to scale up linearly is avoid using indexes at all or as much as you can. But how can you retrieve records in a short time without indexes? Should OrientDB scan the entire database at every query? No. You should use the Graph properties of OrientDB. Let's look at a simple example, where the domain are logs.

A typical log record has some information about the event and a date. Below is the Log record to use in our example. We're going to use the JSON format to simplify reading:

```
{
  "date" : 12293289328932,
  "priority" : "critical",
  "note" : "System reboot"
}
```

Now let's create a tree (that is a directed, non cyclic graph) to group the Log records based on the granularity we need. Example:

```
Year -> month (map) -> Month -> day (map) -> Day -> hour  (map) -> Hour
```

Where Year, Month, Day and Hour are vertex classes. Each Vertex links the other Vertices of smaller type. The links should be handled using a Map to make easier the writing of queries.

Create the classes:

```
CREATE CLASS Year
CREATE CLASS Month
CREATE CLASS Day
CREATE CLASS Hour

CREATE PROPERTY Year.month LINKMAP Month
CREATE PROPERTY Month.day LINKMAP Day
CREATE PROPERTY Day.hour LINKMAP Hour
```

Example to retrieve the vertex relative to the date March 2012, 20th at 10am (2012/03/20 10:00:00):

```
SELECT month[3].day[20].hour[10].logs FROM Year WHERE year = "2012"
```

If you need more granularity than the Hour you can go ahead until the Time unit you need:

```
Hour -> minute (map) -> Minute -> second (map) -> Second
```

Now connect the record to the right Calendar vertex. If the usual way to retrieve Log records is by hour you could link the Log records in the Hour. Example:

```
Year -> month (map) -> Month -> day (map) -> Day -> hour  (map) -> Hour -> log (set) -> Log
```

The "log" property connects the Time Unit to the Log records. So to retrieve all the log of March 2012, 20th at 10am:

```
SELECT expand( month[3].day[20].hour[10].logs ) FROM Year WHERE year = "2012"
```

That could be used as starting point to retrieve only a sub-set of logs that satisfy certain rules. Example:

```
SELECT FROM (
  SELECT expand( month[3].day[20].hour[10].logs ) FROM Year WHERE year = "2012"
) WHERE priority = 'critical'
```

That retrieves all the CRITICAL logs of March 2012, 20th at 10am.

Join multiple hours

If you need multiple hours/days/months as result set you can use the `unionAll()` function to create a unique result set:

```
SELECT expand( records ) from (
  SELECT unionAll( month[3].day[20].hour[10].logs, month[3].day[20].hour[11].logs ) AS records
  FROM Year WHERE year = "2012"
)
```

In this example we create a union between the 10th and 11th hours. But what about extracting all the hours of a day without writing a huge query? The shortest way is using the Traverse. Below the Traverse to get all the hours of one day:

```
TRAVERSE hour FROM (
  SELECT expand( month[3].day[20] ) FROM Year WHERE year = "2012"
)
```

So putting all together this query will extract all the logs of all the hours in a day:

```
SELECT expand( logs ) FROM (
  SELECT unionAll( logs ) AS logs FROM (
    TRAVERSE hour FROM (
     SELECT expand( month[3].day[20] ) FROM Year WHERE year = "2012"
    )
  )
)
```

Aggregate

Once you built up a Calendar in form of a Graph you can use it to store aggregated values and link them to the right Time Unit. Example: store all the winning ticket of Online Games. The record structure in our example is:

```
{
  "date" : 12293289328932,
  "win" : 10.34,
  "machine" : "AKDJKD7673JJSH",
}
```

You can link this record to the closest Time Unit like in the example above, but you could sum all the records in the same Day and link it to the Day vertex. Example:

Create a new class to store the aggregated daily records:

```
CREATE CLASS DailyLog
```

Create the new record from an aggregation of the hour:

```
INSERT INTO DailyLog
SET win = (
  SELECT SUM(win) AS win FROM Hour WHERE date BETWEEN '2012-03-20 10:00:00' AND '2012-03-20 11:00:00'
)
```

Link it in the Calendar graph assuming the previous command returned #23:45 as the RecordId of the brand new DailyLog record:

```
UPDATE (
  SELECT expand( month[3].day[20] ) FROM Year WHERE year = "2012"
) ADD logs = #23:45
```

Chat Use Case

OrientDB allows modeling of rich and complex domains. If you want to develop a chat based application, you can use whatever you want to create the relationships between User and Room.

We suggest avoiding using Edges or Vertices connected with edges for messages. The best way is using the document API by creating one class per chat room, with no index, to have super fast access to last X messages. In facts, OrientDB stores new records in append only, and the @rid is auto generated as incrementing.

The 2 most common use cases in a chat are:

- writing a message in a chat room
- load last page of messages in a chat room

Create the initial schema

In order to work with the chat rooms, the rule of the thumb is creating a base abstract class ("ChatRoom") and then let to the concrete classes to represent individual ChatRooms.

Create the base ChatRoom class

```
create class ChatRoom
alter class ChatRoom abstract true
create property ChatRoom.date datetime
create property ChatRoom.text string
create property ChatRoom.user LINK OUser
```

Create a new ChatRoom

```
create class ItalianRestaurant extends ChatRoom
```

Class "ItalianRestaurant" will extend all the properties from ChatRoom.

Why creating a base class? Because you could always execute polymorphic queries that are cross-chatrooms, like get all the message from user "Luca":

```
select from ChatRoom where user.name = 'Luca'
```

Create a new message in the Chat Room

To create a new message in the chat room you can use this code:

```
public ODocument addMessage(String chatRoom, String message, OUser user) {
  ODocument msg = new ODocument(chatRoom);
  msg.field( "date", new Date() );
  msg.field( "text", message );
  msg.field( "user", user );
  msg.save();
  return msg;
}
```

Example:

```
addMessage("ItalianRestaurant", "Have you ever been at Ponza island?", database.getUser());
```

Retrieve last messages

You can easily fetch pages of messages ordered by date in descending order, by using the OrientDB's `@rid`. Example:

```
select from ItalianRestaurant order by @rid desc skip 0 limit 50
```

You could write a generic method to access to a page of messages, like this:

```
public Iterable<ODocument> loadMessages(String chatRoom, fromLast, pageSize) {
  return graph.getRawGraph().command("select from " + chatRoom + " order by @rid desc skip " + fromLast + " limit " + pageSize
).execute();
}
```

Loading the 2nd (last) page from chat "ItalianRestaurant", would become this query (with pageSize = 50):

```
select from ItalianRestaurant order by @rid desc skip 50 limit 50
```

This is super fast and O(1) even with million of messages.

Limitations

Since OrientDB can handle only 32k clusters, you could have maximum 32k chat rooms. Unless you want to rewrite the entire FreeNode, 32k chat rooms will be more than enough for most of the cases.

However, if you need more than 32k chat rooms, the suggested solution is still using this approach, but with multiple databases (even on the same server, because one OrientDB Server instance can handle thousands of databases concurrently).

In this case you could use one database to handle all the metadata, like the following classes:

- **ChatRoom**, containing all the chatrooms, and the database where are stored. Example: `{ "@class": "ChatRoom", "description": "OrientDB public channel", "databaseName", "db1", "clusterName": "orientdb" }`
- **User**, containing all the information about accounts with the edges to the ChatRoom vertices where they are subscribed

OrientDB cannot handle cross-database links, so when you want to know the message's author, you have to look up into the "Metadata" database by @RID (that is O(1)).

Key Value Use Case

OrientDB can also be used as a Key Value DBMS by using the super fast Indexes. You can have as many Indexes as you need.

HTTP

OrientDB RESTful HTTP protocol allows to talk with a OrientDB Server instance using the HTTP protocol and JSON. OrientDB supports also a highly optimized Binary protocol for superior performances.

Operations

To interact against OrientDB indexes use the four methods of the HTTP protocol in REST fashion:

- **PUT**, to create or modify an entry in the database
- **GET**, to retrieve an entry from the database. It's idempotent that means no changes to the database happen. Remember that in IE6 the URL can be maximum of 2,083 characters. Other browsers supports longer URLs, but if you want to stay compatible with all limit to 2,083 characters
- **DELETE**, to delete an entry from the database

Create an entry

To create a new entry in the database use the Index-PUT API.

Syntax: `http://<server>:[<port>]/index/<index-name>/<key>`

Example:

HTTP PUT: `http://localhost:2480/index/customers/jay`

```
{
  "name" : "Jay",
  "surname" : "Miner"
}
```

HTTP Response 204 is returned.

Retrieve an entry

To retrieve an entry from the database use the Index-GET API.

Syntax: `http://<server>:[<port>]/index/<index-name>/<key>`

Example:

HTTP GET: `http://localhost:2480/index/customers/jay`

HTTP Response 200 is returned with this JSON as payload:

```
{
  "name" : "Jay",
  "surname" : "Miner"
}
```

Remove an entry

To remove an entry from the database use the Index-DELETE API.

Syntax: `http://<server>:[<port>]/index/<index-name>/<key>`

Example:

HTTP DELETE: `http://localhost:2480/index/customers/jay`

HTTP Response 200 is returned

Step-by-Step tutorial

Before to start assure you've a OrientDB server up and running. In this example we'll use curl considering the connection to localhost to the default HTTP post 2480. The default "admin" user is used.

Create a new index

To use OrientDB as a Key/Value store we need a brand new manual index, let's call it "mainbucket". We're going to create it as UNIQUE because keys cannot be duplicated. If you can have multiple keys consider:

- creating the index as NOTUNIQUE
- leave it as UNIQUE but as value handle array of documents

Create the new manual unique index "mainbucket":

```
> curl --basic -u admin:admin localhost:2480/command/demo/sql -d "create index mainbucket UNIQUE"
```

Response:

```
{ "result" : [
    { "@type" : "d" , "@version" : 0, "value" : 0, "@fieldTypes" : "value=l" }
  ]
}
```

Store the first entry

Below we're going to insert the first entry by using the HTTP PUT method passing "jay" as key in the URL and as value the entire document in form of JSON:

```
> curl --basic -u admin:admin -X PUT localhost:2480/index/demo/mainbucket/jay -d "{'name':'Jay','surname':'Miner'}"
```

Response:

```
Key 'jay' correctly inserted into the index mainbucket.
```

Retrieve the entry just inserted

Below we're going to retrieve the entry we just entered by using the HTTP GET method passing "jay" as key in the URL:

```
> curl --basic -u admin:admin localhost:2480/index/demo/mainbucket/jay
```

Response:

```
[{
  "@type" : "d" , "@rid" : "#3:477" , "@version" : 0,
  "name" : "Jay",
  "surname" : "Miner"
}]
```

Note that an array is always returned in case multiple records are associated to the same key (if NOTUNIQUE index is used). Look also at the document has been created with RID #3:477. You can load it directly if you know the RID. Remember to remove the # character. Example:

```
> curl --basic -u admin:admin localhost:2480/document/demo/3:477
```

Response:

```
{
  "@type" : "d" , "@rid" : "#3:477" , "@version" : 0,
  "name" : "Jay",
  "surname" : "Miner"
}
```

Drop an index

Once finished drop the index "mainbucket" created for the example:

```
> curl --basic -u admin:admin localhost:2480/command/demo/sql -d "drop index mainbucket"
```

Response:

```
{ "result" : [
    { "@type" : "d" , "@version" : 0, "value" : 0, "@fieldTypes" : "value=l" }
  ]
}
```

Distributed queues use case

Implementing a persistent, distributed and transactional queue system using OrientDB is possible and easy. Besides the fact you don't need a specific API accomplish a queue, there are multiple approaches you can follow depending by your needs. The easiest way is using OrientDB SQL, so this works with any driver.

Create the queue class first:

```
create class queue
```

You could have one class per queue. Example of push operation:

```
insert into queue set text = "this is the first message", date = date()
```

Since OrientDB by default keeps the order of creation of records, a simple delete from the queue class with limit = 1 gives to you the perfect pop:

```
delete from queue return before limit 1
```

The "return before" allows you to have the deleted record content. If you need to peek the queue, you can just use the select:

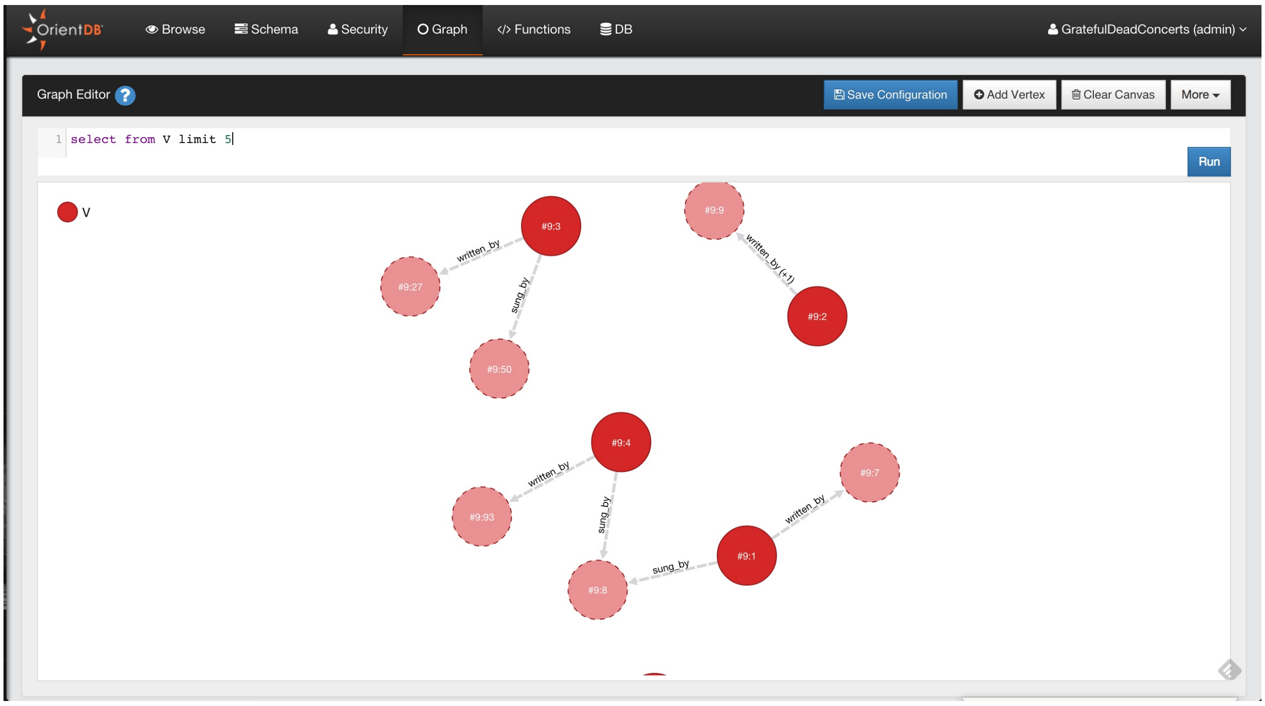
```
select from queue limit 1
```

That's it. Your queue will be persistent, if you want transactional and running in cluster distributed.

Administration

OrientDB has a number of tools to make administration of the database easier. There is the console, which allows you to run a large number of commands.

There is also the OrientDB Studio, which allows you to run queries and visually look at the graph.



```
select from V limit 5
```

OrientDB also offers several tools for the import and export of data, logging and trouble shooting, along with ETL tools.

All of OrientDB's administration facilities are aimed to make your usage of OrientDB as simple and as easy as possible.

> For more information see:
>
> - Studio
> - Console
> - Backup and Restore
> - Export and Import
> - Logging
> - Trouble shooting
> - Performance Tuning
> - ETL Tools
> - Stress Test Tool

Console

OrientDB provides a Console Tool, which is a Java application that connects to and operates on OrientDB databases and Server instances.

Console Modes

There are two modes available to you, while executing commands through the OrientDB Console: interactive mode and batch mode.

Interactive Mode

By default, the Console starts in interactive mode. In this mode, the Console loads to an `orientdb>` prompt. From there you can execute commands and SQL statements as you might expect in any other database console.

You can launch the console in interactive mode by executing the `console.sh` for Linux OS systems or `console.bat` for Windows systems in the `bin` directory of your OrientDB installation. Note that running this file requires execution permissions.

```
$ cd $ORIENTDB_HOME/bin
$ ./console.sh

OrientDB console v.X.X.X (build 0) www.orientdb.com
Type 'HELP' to display all the commands supported.
Installing extensions for GREMLIN language v.X.X.X

orientdb>
```

From here, you can begin running SQL statements or commands. For a list of these commands, see commands.

Batch mode

When the Console runs in batch mode, it takes commands as arguments on the command-line or as a text file and executes the commands in that file in order. Use the same `console.sh` or `console.bat` file found in `bin` at the OrientDB installation directory.

- **Command-line**: To execute commands in batch mode from the command line, pass the commands you want to run in a string, separated by a semicolon.

  ```
  $ $ORIENTDB_HOME/bin/console.sh "CONNECT REMOTE:localhost/demo;SELECT FROM Profile"
  ```

- **Script Commands**: In addition to entering the commands as a string on the command-line, you can also save the commands to a text file as a semicolon-separated list.

  ```
  $ vim commands.txt

    CONNECT REMOTE:localhost/demo;SELECT FROM Profile

  $ $ORIENTDB_HOME/bin/console.sh commands.txt
  ```

Ignoring Errors

When running commands in batch mode, you can tell the console to ignore errors, allowing the script to continue the execution, with the `ignoreErrors` setting.

```
$ vim commands.txt

  SET ignoreErrors TRUE
```

Enabling Echo

Regardless of whether you call the commands as an argument or through a file, when you run console commands in batch mode, you may also need to display them as they execute. You can enable this feature using the `echo` setting, near the start of your commands list.

```
$ vim commands.txt

  SET echo TRUE
```

Enabling Date in prompt

Starting from v2.2.9, to enable the date in the prompt, set the variable `promptDateFormat` with the date format following the SimpleDateFormat specs.

```
orientdb {db=test1}> set promptDateFormat "yyy-MM-dd hh:mm:ss.sss"

orientdb {db=test1 (2016-08-26 09:34:12.012)}>
```

Console commands

OrientDB implements a number of SQL statements and commands that are available through the Console. In the event that you need information while working in the console, you can access it using either the `HELP` or `?` command.

Command	Description
`ALTER CLASS`	Changes the class schema
`ALTER CLUSTER`	Changes the cluster attributes
`ALTER DATABASE`	Changes the database attributes
`ALTER PROPERTY`	Changes the class's property schema
`BACKUP DATABASE`	Backup a database
`BEGIN`	Begins a new transaction
`BROWSE CLASS`	Browses all the records of a class
`BROWSE CLUSTER`	Browses all the records of a cluster
`CLASSES`	Displays all the configured classes
`CLUSTER STATUS`	Displays the status of distributed cluster of servers
`CLUSTERS`	Displays all the configured clusters
`COMMIT`	Commits an active transaction
`CONFIG`	Displays the configuration where the opened database is located (local or remote)
`CONFIG GET`	Returns a configuration value
`CONFIG SET`	Set a configuration value
`CONNECT`	Connects to a database

`CREATE CLASS`	Creates a new class
`CREATE CLUSTER`	Creates a new cluster inside a database
`CREATE CLUSTER`	Creates a new record cluster
`CREATE DATABASE`	Creates a new database
`CREATE EDGE`	Create a new edge connecting two vertices
`CREATE INDEX`	Create a new index
`CREATE LINK`	Create a link reading a RDBMS JOIN
`CREATE VERTEX`	Create a new vertex
`DECLARE INTENT`	Declares an intent
`DELETE`	Deletes a record from the database using the SQL syntax. To know more about the SQL syntax go here
`DICTIONARY KEYS`	Displays all the keys in the database dictionary
`DICTIONARY GET`	Loookups for a record using the dictionary. If found set it as the current record
`DICTIONARY PUT`	Inserts or modify an entry in the database dictionary. The entry is composed by key=String, value=record-id
`DICTIONARY REMOVE`	Removes the association in the dictionary
`DISCONNECT`	Disconnects from the current database
`DISPLAY RECORD`	Displays current record's attributes
`DISPLAY RAW RECORD`	Displays current record's raw format
`DROP CLASS`	Drop a class
`DROP CLUSTER`	Drop a cluster
`DROP DATABASE`	Drop a database
`DROP INDEX`	Drop an index
`DROP PROPERTY`	Drop a property from a schema class
`EXPLAIN`	Explain a command by displaying the profiling values while executing it
`EXPORT DATABASE`	Exports a database
`EXPORT RECORD`	Exports a record in any of the supported format (i.e. json)
`FIND REFERENCES`	Find the references to a record
`FREEZE DATABASE`	Freezes the database locking all the changes. Use this to raw backup. Once frozen it uses the `RELEASE DATABASE` to release it
`GET`	Returns the value of a property
`GRANT`	Grants a permission to a user
`GREMLIN`	Executes a Gremlin script
`IMPORT DATABASE`	Imports a database previously exported
`INDEXES`	Displays information about indexes
`INFO`	Displays information about current status
`INFO CLASS`	Displays information about a class
`INSERT`	Inserts a new record in the current database using the SQL syntax. To know more about the SQL syntax go here
`JS`	Executes a Javascript in the console
`JSS`	Executes a Javascript in the server

`LIST DATABASES`	List the available databases
`LIST CONNECTIONS`	List the available connections
`LOAD RECORD`	Loads a record in memory and set it as the current one
`LOAD SCRIPT`	Loads a script and execute it
`PROFILER`	Controls the Profiler
`PROPERTIES`	Returns all the configured properties
pwd	Display current path
`REBUILD INDEX`	Rebuild an index
`RELEASE DATABASE`	Releases a Console Freeze Database database
`RELOAD RECORD`	Reloads a record in memory and set it as the current one
RELOAD SCHEMA	Reloads the schema
`ROLLBACK`	Rollbacks the active transaction started with begin
`RESTORE DATABASE`	Restore a database
`SELECT`	Executes a SQL query against the database and display the results. To know more about the SQL syntax go here
`REVOKE`	Revokes a permission to a user
`SET`	Changes the value of a property
`SLEEP`	Sleep for the time specified. Useful on scripts
`TRAVERSE`	Traverse a graph of records
`TRUNCATE CLASS`	Remove all the records of a class (by truncating all the underlying configured clusters)
`TRUNCATE CLUSTER`	Remove all the records of a cluster
`TRUNCATE RECORD`	Truncate a record you can't delete because it's corrupted
`UPDATE`	Updates a record in the current database using the SQL syntax. To know more about the SQL syntax go here
HELP	Prints this help
`EXIT`	Closes the console

Custom Commands

In addition to the commands implemented by OrientDB, you can also develop custom commands to extend features in your particular implementation. To do this, edit the OConsoleDatabaseApp class and add to it a new method. There's an auto-discovery system in place that adds the new method to the available commands. To provide a description of the command, use annotations. The command name must follow the Java code convention of separating words using camel-case.

For instance, consider a case in which you might want to add a `MOVE CLUSTER` command to the console:

```
@ConsoleCommand(description = "Move the physical location of cluster files")
public void moveCluster(
   @ConsoleParameter(name = "cluster-name", description = "The name or the id of the cluster to remove") String iClusterName,
   @ConsoleParameter(name = "target-path", description = "path of the new position where to move the cluster files") String iN
ewPath ) {

   checkCurrentDatabase(); // THE DB MUST BE OPENED

   System.out.println("Moving cluster '" + iClusterName + "' to path " + iNewPath + "...");
   }
```

Once you have this code in place, `MOVE CLUSTER` now appears in the listing of available commands shown by `HELP`.

```
orientdb> HELP

AVAILABLE COMMANDS:

 * alter class    Alter a class in the database schema
 * alter cluster  Alter class in the database schema
 ...                            ...
 * move cluster                 Move the physical location of cluster files
 ...                            ...
 * help                         Print this help
 * exit                         Close the console

orientdb>  MOVE CLUSTER foo /temp

Moving cluster 'foo' to path /tmp...
```

In the event that you develop a custom command and find it especially useful in your deployment, you can contribute your code to the OrientDB Community!

Console - `BACKUP`

Executes a complete backup on the currently opened database. It then compresses the backup file using the ZIP algorithm. You can then restore a database from backups, using the `RESTORE DATABASE` command. You can automate backups using the Automatic-Backup server plugin.

Backups and restores are similar to the `EXPORT DATABASE` and `IMPORT DATABASE`, but they offer better performance than these options.

> **NOTE**: OrientDB Community Edition does not support backing up remote databases. OrientDB Enterprise Edition does support this feature. For more information on how to implement this with Enterprise Edition, see Remote Backups.

Syntax:

```
BACKUP DATABASE <output-file> [-incremental] [-compressionLevel=<compressionLevel>] [-bufferSize=<bufferSize>]
```

- **`<output-file>`** Defines the path to the backup file.
- **`-incremental`** Option to execute an incremental backup. When enabled, it computes the data to backup as all new changes since the last backup. Available only in the OrientDB Enterprise Edition version 2.2 or later.
- **`-compressionLevel`** Defines the level of compression for the backup file. Valid levels are `0` to `9`. The default is `9`. Available in 1.7 or later.
- **`-bufferSize`** Defines the compression buffer size. By default, this is set to 1MB. Available in 1.7 or later.

Permissions:

In order to enable a user to execute this command, you must add the permission of `create` for the resource `database.backup` to the database user.

Example:

- Backing up a database:

```
orientdb> CONNECT plocal:../databases/mydatabase admin admin
orientdb> BACKUP DATABASE /backups/mydb.zip

Backing current database to: database mydb.zip
Backup executed in 0.52 seconds
```

Backup API

In addition to backups called through the Console, you can also manage backups through the Java API. Using this, you can perform either a full or incremental backup on your database.

Full Backup

In Java or any other language that runs on top of the JVM, you can initiate a full backup by using the `backup()` method on a database instance.

```
db.backup(out, options, callable, listener, compressionLevel, bufferSize);
```

- **`out`** Refers to the `OutputStream` that it uses to write the backup content. Use a `FileOutputStream` to make the backup persistent on disk.
- **`options`** Defines backup options as a `Map<String, Object>` object.
- **`callable`** Defines the callback to execute when the database is locked.
- **`listener`** Defines the listened called for backup messages.
- **`compressionLevel`** Defines the level of compression for the backup. It supports levels between `0` and `9`, where `0` equals no compression and `9` the maximum. Higher compression levels do mean smaller files, but they also mean the backup requires more

from the CPU at execution time.

- `bufferSize` Defines the buffer size in bytes. The larger the buffer, the more efficient the comrpession.

Example:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/temp/mydb");
db.open("admin", "admin");
try{
  OCommandOutputListener listener = new OCommandOutputListener() {
    @Override
    public void onMessage(String iText) {
      System.out.print(iText);
    }
  };

  OutputStream out = new FileOutputStream("/temp/mydb.zip");
  db.backup(out,null,null,listener,9,2048);
} finally {
   db.close();
}
```

Incremental Backup

As of version 2.2, OrientDB Enterprise Edition supports incremental backups executed through Java or any language that runs on top of the JVM, using the `incrementalBackup()` method against a database instance.

```
db.incrementalBackup(backupDirectory);
```

- `backupDirectory` Defines the directory where it generates the incremental backup files.

It is important that previous incremental backup files are present in the same directory, in order to compute the database portion to back up, based on the last incremental backup.

Example:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/temp/mydb");
db.open("admin", "admin");
try{
  db.backup("/var/backup/orientdb/mydb");
} finally {
   db.close();
}
```

> For more information, see:
>
> - Restore Database
> - Export Database
> - Import Database
> - Console-Commands
> - ODatabaseExport Java class

Console - `BEGIN`

Initiates a transaction. When a transaction is open, any commands you execute on the database remain temporary. In the event that you are satisfied with the changes, you can call the `COMMIT` command to commit them to the database. Otherwise, you can call the `ROLLBACK` command, to roll the changes back to the point where you called `BEGIN` .

Syntax:

```
BEGIN
```

Examples

- Begin a transaction:

```
orientdb> BEGIN

Transaction 1 is running
```

- Attempting to begin a transaction when one is already open:

```
orinetdb> BEGIN

Error: an active transaction is currently open (id=1).  Commit or rollback
before starting a new one.
```

- Making changes when a transaction is open:

```
orientdb> INSERT INTO Account (name) VALUES ('tx test') SELECT FROM Account WHERE name LIKE 'tx%'

 ---+-------+----------
  # | RID   | name
 ---+-------+----------
  0 | #9:-2 | tx test
 ---+-------+----------
```

When a transaction is open, new records all have temporary Record ID's, which are given negative values, (for instance, like the `#9:-2` shown above). These remain in effect until you run `COMMIT`

> For more information on Transactions, see
>
> - Transactions
> - Console Command COMMIT
> - Console Command ROLLBACK
> - Console Commands

Console - `BROWSE CLASS`

Displays all records associated with the given class.

Syntax:

```
BROWSE CLASS <class-name>
```

- **`<class-name>`** Defines the class for the records you want to display.

Permissions:

In order to enable a user to execute this command, you must add the permission of `read` for the resource `database.class.<class>` to the database user.

Example:

- Browse records associated with the class `City` :

  ```
  orientdb> BROWSE CLASS City

  ----+------+-------------------
    # | RID  | NAME
  ----+------+-------------------
    0 | -6:0 | Rome
    1 | -6:1 | London
    2 | -6:2 | Honolulu
  ----+------+-------------------
  ```

> For more information on other commands, see Console Commands.

Console - `BROWSE CLUSTER`

Displays all records associated with the given cluster.

Syntax:

```
BROWSE CLUSTER <cluster-name>
```

- **`<cluster-name>`** Defines the cluster for the records you want to display.

Permissions:

In order to enable a user to execute this command, you must add the permission of `read` for the resource `database.cluster.<class>` to the database user.

Example:

- Browse records associated with the cluster `City`:

  ```
  orientdb> BROWSE CLUSTER City

  ----+------+-------------------
    # | RID  | NAME
  ----+------+-------------------
    0 | -6:0 | Rome
    1 | -6:1 | London
    2 | -6:2 | Honolulu
  ----+------+-------------------
  ```

> For more information on other commands, see Console Commands.

Console - `LIST CLASSES`

Displays all configured classes in the current database.

Syntax:

- Long Syntax:

```
LIST CLASSES
```

- Short Syntax:

```
CLASSES
```

Example

- List current classes in the database:

```
orientdb> LIST CLASSES

CLASSES
-------------+------+-------------+-----------
 NAME        |  ID  | CLUSTERS    | ELEMENTS
-------------+------+-------------+-----------
 Person      |    0 | person      |         7
 Animal      |    1 | animal      |         5
 AnimalRace  |    2 | AnimalRace  |         0
 AnimalType  |    3 | AnimalType  |         1
 OrderItem   |    4 | OrderItem   |         0
 Order       |    5 | Order       |         0
 City        |    6 | City        |         3
-------------+------+-------------+-----------
 TOTAL                                     16
----------------------------------------------
```

> For more information on other commands, see Console Commands.

Console - `CLUSTER STATUS`

Displays the status of the cluster in distributed configuration.

Syntax:

```
CLUSTER STATUS
```

Example:

- Display the status of the cluster:

  ```
  orientdb> CLUSTER STATUS

  {
      "localName": "_hzInstance_1_orientdb",
      "localId": "3735e690-9a7b-44d2-b4bc-27089da065e2",
      "members": [
          {
              "id": "3735e690-9a7b-44d2-b4bc-27089da065e2",
              "name": "node1",
              "startedOn": "2015-05-14 17:06:40:418",
              "listeners": [
                  {
                      "protocol": "ONetworkProtocolBinary",
                      "listen": "10.3.15.55:2424"
                  },
                  {
                      "protocol": "ONetworkProtocolHttpDb",
                      "listen": "10.3.15.55:2480"
                  }
              ],
              "databases": []
          }
      ]
  }
  ```

> For more information on other commands, see Console Commands.

Console - `LIST CLUSTERS`

Displays all configured clusters in the current database.

Syntax:

- Long Syntax:

```
LIST CLUSTERS
```

- Short Syntax:

```
CLUSTERS
```

Example:

- List current clusters on database:

```
orientdb> LIST CLUSTERS

CLUSTERS
-------------+------+-----------+-----------
 NAME        |  ID  | TYPE      | ELEMENTS
-------------+------+-----------+-----------
 metadata    |    0 | Physical  |        11
 index       |    1 | Physical  |         0
 default     |    2 | Physical  |       779
 csv         |    3 | Physical  |      1000
 binary      |    4 | Physical  |      1001
 person      |    5 | Physical  |         7
 animal      |    6 | Physical  |         5
 animalrace  |   -2 | Logical   |         0
 animaltype  |   -3 | Logical   |         1
 orderitem   |   -4 | Logical   |         0
 order       |   -5 | Logical   |         0
 city        |   -6 | Logical   |         3
-------------+------+-----------+-----------
 TOTAL                                  2807
--------------------------------------------
```

> For information on creating new clusters in the current database, see the `CREATE CLUSTER` command. For more information on other commands, see Console Commands.

Console - `LIST SERVERS`

Displays all active servers connected within a cluster.

> This command was introduced in OrientDB version 2.2.

Syntax:

```
LIST SERVERS
```

Example:

- List the servers currently connected to the cluster:

```
orientdb> LIST SERVERS

CONFIGURED SERVERS
-+----+------+-----------+-------------+-----------+-----------+-----------+----------
+---------
#|Name|Status|Connections|StartedOn     |Binary     |HTTP       |UsedMemory
|FreeMemory|MaxMemory
-+----+------+-----------+-------------+-----------+-----------+-----------+----------
+---------
0|no2 |ONLINE|0          |2015-10-
30...|192.168.0.6|192.168.0.6|80MB(8.80%)|215MB(23%)|910MB
1|no1 |ONLINE|0          |2015-10-30...|192.168.0.6|192.168.0.6|90MB(2.49%)|195MB(5%)
|3.5GB
-+----+------+-----------+-------------+-----------+-----------+-----------+----------
+---------
```

- Use the `DISPLAY` command to show information on a specific server:

```
orientdb> DISPLAY 0
-------------+------------------------------
        Name | Value
-------------+------------------------------
        Name | node2
      Status | ONLINE
 Connections | 0
   StartedOn | Fri Oct 30 21:41:07 CDT 2015
      Binary | 192.168.0.6:2425
        HTTP | 192.168.0.6:2481
  UsedMemory | 80,16MB (8,80%)
  FreeMemory | 215,34MB (23,65%)
   MaxMemory | 910,50MB
-------------+------------------------------
```

> For more information on other commands, see Console Commands.

Console - LIST SERVER USERS

> This feature was introduced in OrientDB version 2.2.

Displays all configured users on the server. In order to display the users, the current system user that is running the console must have permissions to read the `$ORINETDB_HOME/config/orientdb-server-config.xml` configuration file. For more information, see OrientDB Server Security.

Syntax:

```
LIST SERVER USERS
```

Example:

- List configured users on a server:

```
orientdb> LIST SERVER USERS

SERVER USERS
- 'root', permissions: *
- 'guest', permissions: connect,server.listDatabases,server.dblist
```

> For more information, see
>
> - `SET SERVER USER`
> - `DROP SERVER USER`
>
> For more information on other console commands, see Console Commands.

Console - `CHECK DATABASE`

Checks the integrity of a database. In the case the database contains graphs, their consistency is checked. To repair a database, use Repair Database Command.

Syntax

```
CHECK DATABASE [--skip-graph] [-v]
```

- `[--skip-graph]` Skips the check of the graph
- `[-v]` Verbose mode

Examples

- Check a graph database:

```
orientdb> CHECK DATABASE

Check of graph 'plocal:/temp/testdb' is started ...
Scanning 1 edges (skipEdges=0)...
+ found corrupted edge E#17:0{out:#9:0,in:#11:0,test:true} v2 because incoming vertex (#11:0) does not contain the edge
Scanning edges completed
Scanning 710 vertices...
+ found corrupted vertex V#10:0{in_:[#17:0],name:Marko} v2 the edge should be removed from property in_ (ridbag)
Scanning vertices completed
Check of graph 'plocal:/temp/testdb' completed in 0 secs
 scannedEdges.....: 1
 edgesToRemove....: 1
 scannedVertices..: 710
 scannedLinks.....: 2
 linksToRemove....: 1
 verticesToRepair.: 0
Check of storage completed in 296ms.  without errors.
```

> For more information on other commands, see Console Commands.

Console - COMMIT

Closes a transaction, committing the changes you have made to the database. Use the `BEGIN` command to open a transaction. If you don't want to save the changes you've made, use the `ROLLBACK` command to revert the database state back to the point where you opened the transaction.

> For more information, see Transactions.

Syntax

```
COMMIT
```

Example

- Initiate a transaction, using the `BEGIN` command:

```
orientdb> BEGIN

Transaction 2 is running
```

- For the sake of example, attempt to open another transaction:

```
orientdb> BEGIN

Error: an active transaction is currently open (id=2). Commit or rollback
before starting a new one.
```

- Insert data into the class `Account` , using an `INSERT` statement:

```
orientdb> INSERT INTO Account (name) VALUES ('tx test')

Inserted record 'Account#9:-2{name:tx test} v0' in 0,000000 sec(s).
```

- Commit the transaction to the database:

```
orientdb> COMMIT

Transaction 2 has been committed in 4ms
```

- Display the new content, using a `SELECT` query:

```
orientdb> SELECT FROM Account WHERE name LIKE 'tx%'

---+---------+----------
 # | RID     | name
---+---------+----------
 0 | #9:1107 | tx test
---+---------+----------

1 item(s) found. Query executed in 0.041 sec(s).
```

When a transaction is open, all new records use a temporary Record ID that features negative numbers. After the commit, they have a permanent Record ID that uses with positive numbers.

> For more information, see
>
> - Transactions
> - `BEGIN`
> - `ROLLBACK`
> - Console Commands

Console - `CONFIG`

Displays the configuration information on the current database, as well as whether it is local or remote.

Syntax

```
CONFIG
```

Examples

- Display the configuration of the current database:

```
orientdb> CONFIG

REMOTE SERVER CONFIGURATION:
+------------------------------------+--------------------------------+
| NAME                               | VALUE                          |
+------------------------------------+--------------------------------+
| treemap.lazyUpdates                | 300                            |
| db.cache.enabled                   | false                          |
| file.mmap.forceRetry               | 5                              |
| treemap.optimizeEntryPointsFactor  | 1.0                            |
| storage.keepOpen                   | true                           |
| treemap.loadFactor                 | 0.7                            |
| file.mmap.maxMemory                | 110000000                      |
| network.http.maxLength             | 10000                          |
| storage.cache.size                 | 5000                           |
| treemap.nodePageSize               | 1024                           |
| ...                                | ...                            |
| treemap.entryPoints                | 30                             |
+------------------------------------+--------------------------------+
```

> You can change configuration variables displayed here using the `CONFIG SET` command. To display the value set to one configuration variable, use the `CONFIG GET` command.
>
> For more information on other commands, see Console Commands.

Console - `CONFIG GET`

Displays the value of the requested configuration variable.

Syntax

```
CONFIG GET <config-variable>
```

- **`<config-variable>`** Defines the configuration variable you want to query.

Examples

- Display the value to the `tx.log.fileType` configuration variable:

  ```
  orientdb> CONFIG GET tx.log.fileType

  Remote configuration: tx.log.fileType = classic
  ```

> You can display all configuration variables using the `CONFIG` command. To change the values, use the `CONFIG SET` command.
>
> For more information on other commands, see Config Commands.

Console - `CONFIG SET`

Updates a configuration variable to the given value.

Syntax

```
CONFIG SET <config-variable> <config-value>
```

- **`<config-variable>`** Defines the configuration variable you want to change.
- **`<config-value>`** Defines the value you want to set.

Example

- Display the current value for `tx.autoRetry` :

  ```
  orientdb> CONFIG GET tx.autoRetry

  Remote configuration: tx.autoRetry = 1
  ```

  Change the `tx.autoRetry` value to `5` :

  ```
  orientdb> CONFIG SET tx.autoRetry 5

  Remote configuration value changed correctly.
  ```

  Display new value:

  ```
  orientdb> CONFIG GET tx.autoRetry

  Remote configuration: tx.autoRetry = 5
  ```

> You can display all configuration variables with the `CONFIG` command. You can view the current value on a configuration variable using the `CONFIG GET` command.
>
> For more information on other commands, see Console Commands

Console - `CONNECT`

Opens a database.

Syntax

```
CONNECT <database-url> <user> <password>
```

- **`<database-url>`** Defines the URL of the database you want to connect to. It uses the format `<mode>:<path>`
  - *`<mode>`* Defines the mode you want to use in connecting to the database. It can be `plocal` or `remote` .
  - *`<path>`* Defines the path to the database.
- **`<user>`** Defines the user you want to connect to the database with.
- **`<password>`** Defines the password needed to connect to the database, with the defined user.

Examples:

- Connect to a local database as the user `admin` , loading it directly into the console:

```
orientdb> CONNECT plocal:../databases/GratefulDeadConcerts admin my_admin_password

Connecting to database [plocal:../databases/GratefulDeadConcerts]...OK
```

- Connect to a remote database:

```
orientdb> CONNECT remote:192.168.1.1/GratefulDeadConcerts admin my_admin_password

Connecting to database [remote:192.168.1.1/GratefulDeadConcerts]...OK
```

> For more information on other commands, see Console Commands.

Console - `CREATE CLUSTER`

Creates a new cluster in the current database. The cluster you create can either be physical or in memory. OrientDB saves physical clusters to disk. Memory clusters are volatile, so any records you save to them are lost, should the server be stopped.

Syntax

```
CREATE CLUSTER <cluster-name> <cluster-type> <data-segment> <location> [<position>]
```

- **`<cluster-name>`** Defines the name of the cluster.
- **`<cluster-type>`** Defines whether the cluster is `PHYSICAL` or `LOGICAL`.
- **`<data-segment>`** Defines the data segment you want to use.
  - *`DEFAULT`* Sets the cluster to the default data segment.
- **`<location>`** Defines the location for new cluster files, if applicable. Use `DEFAULT` to save these to the database directory.
- **`<position>`** Defines where to add new cluster. Use `APPEND` to create it as the last cluster. Leave empty to replace.

Example

- Create a new cluster `documents`:

  ```
  orientdb> CREATE CLUSTER documents PHYSICAL DEFAULT DEFAULT APPEND

  Creating cluster [documents] of type 'PHYSICAL' in database demo as last one...
  PHYSICAL cluster created correctly with id #68
  ```

> You can display all configured clusters in the current database using the `CLUSTERS` command. To delete an existing cluster, use the `DROP CLUSTER` command.
>
> For more information on other commands, see Console Commands

Console - `CREATE DATABASE`

Creates and connects to a new database.

Syntax

```
CREATE DATABASE <database-url> [<user> <password> <storage-type> [<db-type>]] [-restore=<backup-path>]
```

- **`<database-url>`** Defines the URL of the database you want to connect to. It uses the format `<mode>:<path>`
  - *`<mode>`* Defines the mode you want to use in connecting to the database. It can be `PLOCAL` or `REMOTE` .
  - *`<path>`* Defines the path to the database.
- **`<user>`** Defines the user you want to connect to the database with.
- **`<password>`** Defines the password needed to connect to the database, with the defined user.
- **`<storage-type>`** Defines the storage type that you want to use. You can choose between `PLOCAL` and `MEMORY` .
- **`<db-type>`** Defines the database type. You can choose between `GRAPH` and `DOCUMENT` . The default is `GRAPH` .

Examples

- Create a local database `demo` :

```
orientdb> CREATE DATABASE PLOCAL:/usr/local/orientdb/databases/demo

Creating database [plocal:/usr/local/orientdb/databases/demo]...
Connecting to database [plocal:/usr/local/orientdb/databases/demo]...OK
Database created successfully.

Current database is: plocal:/usr/local/orientdb/databases/demo

orientdb {db=demo}>
```

- Create a remote database `trick` :

```
orientdb> CREATE DATABASE REMOTE:192.168.1.1/trick root
          E30DD873203AAA245952278B4306D94E423CF91D569881B7CAD7D0B6D1A20CE9 PLOCAL

Creating database [remote:192.168.1.1/trick ]...
Connecting to database [remote:192.168.1.1/trick ]...OK
Database created successfully.

Current database is: remote:192.168.1.1/trick

orientdb {db=trick}>
```

> To create a static database to use from the server, see `Server pre-configured storage types` .
>
> To remove a database, see `DROP DATABASE` . To change database configurations after creation, see `ALTER DATABASE` .
>
> For more information on other commands, see Console Commands.

Incremental restore option

You can execute an incremental restore at creation time through the option `-restore` specifying as value the path where your backup is placed. Let's suppose we want create a new fresh database "mydb" and restore data from a backup, located in `/tmp/backup` , performed from another database in one shot. In this case we can type:

```
orientdb> create database remote:localhost/mydb root root plocal graph -restore=/tmp/backup

Creating database [remote:localhost/mydb] using the storage type [plocal]...
Connecting to database [remote:localhost/mydb] with user 'admin'...OK

Database created successfully.

Current database is: remote:localhost/mydb
```

For further details on incremental backup and restore you can refer to the page Incremental Backup and Restore.

Console - `CREATE INDEX`

Create an index on a given property. OrientDB supports three index algorithms and several index types that use these algorithms.

- **SB-Tree Algorithm**
  - `UNIQUE` Does not allow duplicate keys, fails when it encounters duplicates.
  - `NOTUNIQUE` Does allow duplicate keys.
  - `FULLTEXT` Indexes to any single word of text.
  - `DICTIONARY` Does not allow duplicate keys, overwrites when it encounters duplicates.
- **Hash Index Algorithm**
  - `UNIQUE_HASH_INDEX` Does not allow duplicate keys, it fails when it encounters duplicates.
  - `NOTUNIQUE_HASH_INDEX` Does allow duplicate keys.
  - `FULLTEXT_HASH_INDEX` Indexes to any single word.
  - `DICTIONARY` Does not allow duplicate keys, it overwrites when it encounters duplicates.
- **Lucene Engine**
  - `LUCENE` Full text index type using the Lucene Engine.
  - `SPATIAL` Spatial index using the Lucene Engine.

> For more information on indexing, see Indexes.

Syntax

```
CREATE INDEX <index-name> [ON <class-name> (<property-names>)] <index-type> [<key-type>]
```

- **`<index-name>`** Defines a logical name for the index. Optionally, you can use the format `<class-name>.<property-name>`, to create an automatic index bound to the schema property.

  > **NOTE** Because of this feature, index names cannot contain periods.

- **`<class-name>`** Defines the class to index. The class must already exist in the database schema.

- **`<property-names>`** Defines a comma-separated list of properties that you want to index. These properties must already exist in the database schema.
- **`<index-type>`** Defines the index type that you want to use.
- **`<key-type>`** Defines the key that you want to use. On automatic indexes, this is auto-determined by reading the target schema property where you create the index. When not specified for manual indexes, OrientDB determines the type at run-time during the first insertion by reading the type of the class.

Examples

- Create an index that uses unique values and the SB-Tree index algorithm:

  ```
  orientdb> CREATE INDEX jobs.job_id UNIQUE
  ```

> The SQL `CREATE INDEX` page provides more information on creating indexes. More information on indexing can be found under Indexes. Further SQL information can be found under `SQL Commands`.
>
> For more information on other commands, see Console Commands

Console - `CREATE LINK`

Creates a link between two or more records of the Document type.

Syntax

```
CREATE LINK <link-name> FROM <source-class>.<source-property> TO <target-class>.<target-property>
```

- `<link-name>` Defines the logical name of the property for the link. When not expressed, it overwrites the `<target-property>` field.
- `<source-class>` Defines the source class for the link.
- `<source-property>` Defines the source property for the link.
- `<target-class>` Defines the target class for the link.
- `<target-property>` Defines the target property for the link.

Examples

- Create a 1-*n* link connecting comments to posts:

  ```
  orientdb> CREATE LINK comments FROM Comments.!PostId TO Posts.Id INVERSE
  ```

Understanding Links

Links are useful when importing data from a Relational database. In the Relational world, the database resolves relationships as foreign keys. For instance, consider the above example where you need to show instances in the class `Post` as having a 1-*n* relationship to instances in class `Comment`. That is, `Post 1 ---> * Comment`.

In a Relational database, where classes are tables, you might have something like this:

```
reldb> SELECT * FROM Post;

+----+----------------+
| Id | Title          |
+----+----------------+
| 10 | NoSQL movement |
| 20 | New OrientDB   |
+----+----------------+
2 rows in (0.01 sec)

reldb> SELECT * FROM Comment;

+----+--------+--------------+
| Id | PostId | Text         |
+----+--------+--------------+
|  0 |   10   | First        |
|  1 |   10   | Second       |
| 21 |   10   | Another      |
| 41 |   20   | First again  |
| 82 |   20   | Second Again |
+----+--------+--------------+
5 rows in sec (0.03 sec)
```

In OrientDB, you have a direct relationship in your object model. Navigation runs from `Post` to `Comment` and not vice versa, (as in the Relational database model). For this reason, you need to create a link as `INVERSE`.

> For more information on SQL commands, see SQL Commands.
> For more information on other commands, see Console Commands.

Console - `CREATE PROPERTY`

Creates a new property on the given class. The class must already exist.

Syntax

```
CREATE PROPERTY <class-name>.<property-name> <property-type> [<linked-type>][ <linked-class>]
```

- `<class-name>` Defines the class you want to create the property in.
- `<property-name>` Defines the logical name of the property.
- `<property-type>` Defines the type of property you want to create. Several options are available:
  - `<linked-type>` Defines the container type, used in container property types.
  - `<linked-class>` Defines the container class, used in container property types.

> **NOTE**: There are several property and link types available.

Examples

- Create the property `name` on the class `User`, of the string type:

  ```
  orientdb> CREATE PROPERTY User.name STRING
  ```

- Create a list of strings as the property `tags` in the class `Profile`, using an embedded list of the string type.

  ```
  orientdb> CREATE PROPERTY Profile.tags EMBEDDEDLIST STRING
  ```

- Create the embedded map property `friends` in the class `Profile`, link it to the class `Profile`.

  ```
  orientdb> CREATE PROPERTY Profile.friends EMBEDDEDMAP Profile
  ```

  This forms a circular reference.

> To remove a property, use the `DROP PROPERTY` command.

Property Types

When creating properties, you need to define the property type, so that OrientDB knows the kind of data to expect in the field. There are several standard property types available:

`BOOLEAN`	`INTEGER`	`SHORT`	`LONG`
`FLOAT`	`DATE`	`STRING`	`EMBEDDED`
`LINK`	`BYTE`	`BINARY`	`DOUBLE`

In addition to these, there are several more property types that function as containers. These form lists, sets and maps. Using container property types requires that you also define a link type or class.

`EMBEDDEDLIST`	`EMBEDDEDSET`	`EMBEDDEDMAP`
`LINKLIST`	`LINKSET`	`LINKMAP`

Link Types

The link types available are the same as those available as the standard property types:

`BOOLEAN`	`INTEGER`	`SHORT`	`LONG`
`FLOAT`	`DOUBLE`	`DATE`	`STRING`
`BINARY`	`EMBEDDED`	`LINK`	`BYTE`

> For more information, see SQL Commands and Console Commands.

Console - `DECLARE INTENT`

Declares an intent for the current database. Intents allow you to tell the database what you want to do.

Syntax

```
DECLARE INTENT <intent-name>
```

- **`<intent-name>`** Defines the name of the intent. OrientDB supports three intents:
  - *`NULL`* Removes the current intent.
  - *`MASSIVEINSERT`*
  - *`MASSIVEREAD`*

Examples

- Declare an intent for a massive insert:

  ```
  orientdb> DECLARE INTENT MASSIVEINSERT
  ```

- After the insert, clear the intent:

  ```
  orientdb> DECLARE INTENT NULL
  ```

> For more information on other commands, see Console Commands.

Console - DELETE

Remove one or more records from the database. You can determine which records get deleted using the `WHERE` clause.

Syntax

```
DELETE FROM <target-name> [LOCK <lock-type>] [RETURN <return-type>]
  [WHERE <condition>*] [LIMIT <MaxRecords>] [TIMEOUT <timeout-value>]
```

- **`<target-name>`** Defines the target from which you want to delete records. Use one of the following target names:
  - `<class-name>` Determines what class you want to delete from.
  - `CLUSTER:<cluster-name>` Determines what cluster you want to delete from.
  - `INDEX:<index-name>` Determines what index you want to delete from.
- **`LOCK <lock-type>`** Defines how the record locks between the load and deletion. It takes one of two types:
  - `DEFAULT` Operation uses no locks. In the event of concurrent deletions, the MVCC throws an exception.
  - `RECORD` Locks the record during the deletion.
- **`RETURN <return-type>`** Defines what the Console returns. There are two supported return types:
  - `COUNT` Returns the number of deleted records. This is the default return type.
  - `BEFORE` Returns the records before the deletion.
- **`WHERE <condition>`** Defines the condition used in selecting records for deletion.
- **`LIMIT`** Defines the maximum number of records to delete.
- **`TIMEOUT`** Defines the time-limit to allow the operation to run before it times out.

> **NOTE**: When dealing with vertices and edges, do not use the standard SQL `DELETE` command. Doing so can disrupt graph integrity. Instead, use the `DELETE VERTEX` or the `DELETE EDGE` commands.

Examples

- Remove all records from the class `Profile`, where the surname is unknown, ignoring case:

  ```
  orientdb> DELETE FROM Profile WHERE surname.toLowerCase() = 'unknown'
  ```

> For more information on other commands, see SQL Commands and Console Commands.

Console - `DICTIONARY GET`

Displays the value of the requested key, loaded from the database dictionary.

Syntax

```
DICTIONARY GET <key>
```

- `<key>` Defines the key you want to access.

Example

- In a dictionary of U.S. presidents, display the entry for Barack Obama:

  ```
  orientdb> DICTIONARY GET obama

  -------------------------------------------------------------------------
  Class: Person id: 5:4 v.1
  -------------------------------------------------------------------------
      parent: null
   children : [Person@5:5{parent:Person@5:4,children:null,name:Malia Ann,
              surname:Obama,city:null}, Person@5:6{parent:Person@5:4,
              children:null,name:Natasha,surname:Obama,city:null}]
       name : Barack
    surname : Obama
       city : City@-6:2{name:Honolulu}
  -------------------------------------------------------------------------
  ```

> You can display all keys stored in a database using the `DICTIONARY KEYS` command. For more information on indexes, see Indexes.
>
> For more information on other commands, see Console Commands.

Console - `DICTIONARY KEYS`

Displays all the keys stored in the database dictionary.

Syntax

```
DICTIONARY KEYS
```

Example

- Display all the keys stored in the database dictionary:

  ```
  orientdb> DICTIONARY KEYS

  Found 4 keys:
  #0: key-148
  #1: key-147
  #2: key-146
  #3: key-145
  ```

> To load the records associated with these keys, use the `DICTIONARY GET` command. For more information on indexes, see Indexes.
>
> For more information on other commands, see Console Commands.

Console - `DICTIONARY PUT`

Binds a record to a key in the dictionary database, making it accessible to the `DICTIONARY GET` command.

Syntax

```
DICTIONARY PUT <key> <record-id>
```

- **`<key>`** Defines the key you want to bind.
- **`<record-id>`** Defines the ID for the record you want to bind to the key.

Example

- In the database dictionary of U.S. presidents, bind the record for Barack Obama to the key `obama`:

```
orientdb> DICTIONARY PUT obama 5:4

------------------------------------------------------------------------
 Class: Person   id: 5:4   v.1
------------------------------------------------------------------------
    parent : null
  children : [Person@5:5{parent:Person@5:4,children:null,name:Malia Ann,
             surname:Obama,city:null}, Person@5:6{parent:Person@5:4,
             children:null,name:Natasha,surname:Obama,city:null}]
      name : Barack
   surname : Obama
      city : City@-6:2{name:Honolulu}
------------------------------------------------------------------------
The entry obama=5:4 has been inserted in the database dictionary
```

> To see all the keys stored in the database dictionary, use the `DICTIONARY KEYS` command. For more information on dictionaries and indexes, see Indexes.
>
> For more information on other commands, see Console Commands.

Console - `DICTIONARY REMOVE`

Removes the association from the database dictionary.

Syntax

```
DICTIONARY REMOVE <key>
```

- `<key>` Defines the key that you want to remove.

Example

- In a database dictionary of U.S. presidents, remove the key for Barack Obama:

```
orientdb> DICTIONARY REMOVE obama

Entry removed from the dictionary. Last value of entry was:
------------------------------------------------------------------------
Class: Person   id: 5:4   v.1
------------------------------------------------------------------------
   parent : null
 children : [Person@5:5{parent:Person@5:4,children:null,name:Malia Ann,
            surname:Obama,city:null}, Person@5:6{parent:Person@5:4,
            children:null,name:Natasha,surname:Obama,city:null}]
     name : Barack
  surname : Obama
     city : City@-6:2{name:Honolulu}
------------------------------------------------------------------------
```

> You can display information for all keys stored in the database dictionary using the `DICTIONARY KEY` command. For more information on dictionaries and indexes, see Indexes.
>
> For more information on other commands, see Console Commands.

Console - DISCONNECT

Closes the currently opened database.

Syntax

```
DISCONNECT
```

Example

- Disconnect from the current database:

  ```
  orientdb> DISCONNECT

  Disconnecting from the database [../databases/petshop/petshop]...OK
  ```

> To connect to a database, see `CONNECT`. For more information on other commands, see Console Commands.

Console - DISPLAYS RECORD

Displays details on the given record from the last returned result-set.

Syntax

```
DISPLAY RECORD <record-number>
```

- `<record-number>` Defines the relative position of the record in the last result-set.

Example

- Query the database on the class `Person` to generate a result-set:

```
orientdb> SELECT FROM Person

---+-----+--------+----------+-----------+-----------+------
 # | RID | PARENT | CHILDREN | NAME      | SURNAME   | City
---+-----+--------+----------+-----------+-----------+------
 0 | 5:0 | null   | null     | Giuseppe  | Garibaldi | -6:0
 1 | 5:1 | 5:0    | null     | Napoleon  | Bonaparte | -6:0
 2 | 5:2 | 5:3    | null     | Nicholas  | Churchill | -6:1
 3 | 5:3 | 5:2    | null     | Winston   | Churchill | -6:1
 4 | 5:4 | null   | [2]      | Barack    | Obama     | -6:2
 5 | 5:5 | 5:4    | null     | Malia Ann | Obama     | null
 6 | 5:6 | 5:4    | null     | Natasha   | Obama     | null
---+-----+--------+----------+-----------+-----------+------
7 item(s) found. Query executed in 0.038 sec(s).
```

- With the result-set ready, display record number four in the result-set, (for Malia Ann Obama):

```
orientdb> DISPLAY RECORD 5

------------------------------------------------------------------------
Class: Person   id: 5:5   v.0
------------------------------------------------------------------------
  parent : Person@5:4{parent:null,children:[Person@5:5, Person@5:6],
           name:Barack,surname:Obama,city:City@-6:2}
 children : null
     name : Malia Ann
  surname : Obama
      city : null
------------------------------------------------------------------------
```

> For more information on other commands, see Console Commands.

Console - `DISPLAYS RAW RECORD`

Displays details on the given record from the last returned result-set in a binary format.

Syntax

```
DISPLAY RAW RECORD <record-number>
```

- **`<record-number>`** Defines the relative position of the record in the last result-set.

Example

- Query the database on the class `V` to generate a result-set:

```
orientdb {db=GratefulDeadConcerts}> SELECT song_type, name, performances FROM V LIMIT 6

-----+-------+--------+----------+-------------------------+--------------
 #   | @RID  | @CLASS | song_type | name                    | performances
-----+-------+--------+----------+-------------------------+--------------
 0   | #9:1  | V      | cover     | HEY BO DIDDLEY          | 5
 1   | #9:2  | V      | cover     | IM A MAN                | 1
 2   | #9:3  | V      | cover     | NOT FADE AWAY           | 531
 3   | #9:4  | V      | original  | BERTHA                  | 394
 4   | #9:5  | V      | cover     | GOING DOWN THE ROAD...  | 293
 5   | #9:6  | V      | cover     | MONA                    | 1
 6   | #9:7  | V      | null      | Bo_Diddley              | null
-----+-------+--------+-----------+-----------------------+------------
LIMIT EXCEEDED: resultset contains more items not displayed (limit=6)
6 item(s) found. Query executed in 0.136 sec(s).
```

- Display raw record on the song "Hey Bo Diddley" from the result-set:

```
orientdb {db=GratefulDeadConcerts}> DISPLAY RAW RECORD 0

Raw record content. The size is 292 bytes, while settings force to print first 150
bytes:
Vsong_typenametypeperformancesout_followed_byout_written_byout_sung_byin_followed_byco
verHEY BO D
```

> For more information on other commands available, see Console Commands.

Console - `DROP CLUSTER`

Removes a cluster from the database completely, deleting it with all records and caches in the cluster.

Syntax

```
DROP CLUSTER <cluster-name>
```

- **`<cluster-name>`** Defines the name of the cluster you want to drop.

> **NOTE**: When you drop a cluster, the cluster and all records and caches in the cluster are gone. Unless you have made backups, there is no way to restore the cluster after you drop it.

Examples

- Drop a cluster `person` from the current, local database:

  ```
  orientdb> DROP CLUSTER person
  ```

  This removes both the cluster `Person` and all records of the `Person` class in that cluster.

> You can create a new cluster using the `CREATE CLUSTER` command.
>
> For information on other commands, see SQL and Console commands.

Console - `DROP DATABASE`

Removes a database completely. If the database is open and a database name not given, it removes the current database.

Syntax

```
DROP DATABASE [<database-name> <server-username> <server-user-password>]
```

- `<database-name` Defines the database you want to drop. By default it uses the current database, if it's open.
- `<server-username>` Defines the server user. This user must have the privileges to drop the database.
- `<server-user-password>` Defines the password for the server user.

> **NOTE**: When you drop a database, it deletes the database and all records, caches and schema information it contains. Unless you have made backups, there is no way to restore the database after you drop it.

Examples

- Remove the current local database:

  ```
  orientdb> DROP DATABASE
  ```

- Remove the database `demo` at localhost:

  ```
  orientdb> DROP DATABASE remote:localhost/demo root root_password
  ```

> You can create a new database using the `CREATE DATABASE` command. To make changes to an existing database, use the `ALTER DATABASE` command.
>
> For more information on other commands, see SQL and Console commands.

Console - `DROP SERVER USER`

Removes a user from the server. In order to do so, the current system user running the Console, must have permissions to write to the `$ORIENTDB_HOME/config/orientdb-server-config.xmL` configuration file.

Syntax

```
DROP SERVER USER <user-name>
```

- **`<user-name>`** Defines the user you want to drop.

> **NOTE**: For more information on server users, see OrientDB Server Security.
>
> This feature was introduced in version 2.2.

Example

- Remove the user `editor` from the Server:

  ```
  orientdb> DROP SERVER USER editor

  Server user 'editor' dropped correctly
  ```

> To view the current server users, see the `LIST SERVER USERS` command. To create or update a server user, see the `SET SERVER USER` command.
>
> For more information on other commands, see Console Commands.

Console - `EXPORT`

Exports the current database to a file. OrientDB uses a JSON-based Export Format. By default, it compresses the file using the GZIP algorithm.

With the `IMPORT` command, this allows you to migrate the database between different versions of OrientDB without losing data.

> If you receive an error about the database version, export the database using the same version of OrientDB that has generated the database.

Bear in mind, exporting a database browses it, rather than locking it. While this does mean that concurrent operations can execute during the export, it also means that you cannot create an exact replica of the database at the point when the command is issued. In the event that you need to create a snapshot, use the `BACKUP` command.

You can restore a database from an export using the `IMPORT`.

> **NOTE**: While the export format is JSON, there are some constraints in the field order. Editing this file or adjusting its indentation may cause imports to fail.

Syntax

By default, this command exports the full database. Use its options to disable the parts you don't need to export.

```
EXPORT DATABASE <output-file>
      [-excludeAll]
      [-includeClass=<class-name>*]
      [-excludeClass=<class-name>*]
      [-includeCluster=<cluster-name>*]
      [-excludeCluster=<cluster-name>*]
      [-includeInfo=<true|false>]
      [-includeClusterDefinitions=<true|false>]
      [-includeSchema=<true|false>]
      [-includeSecurity=<true|false>]
      [-includeRecords=<true|false>]
      [-includeIndexDefinitions=<true|false>]
      [-includeManualIndexes=<true|false>]
      [-compressionLevel=<0-9>]
      [-compressionBuffer=<bufferSize>]
```

- **`<output-file>`** Defines the path to the output file.
- **`-excludeAll`** Sets the export to exclude everything not otherwise included through command options
- **`-includeClass`** Export includes certain classes, specifically those defined by a space-separated list. In case you specify multiple class names, you have to wrap the list between quotes, eg. `-includeClass="Foo Bar Baz"`
- **`-excludeClass`** Export excludes certain classes, specifically those defined by a space-separated list.
- **`-includeCluster`** Export includes certain clusters, specifically those defined by a space-separated list.
- **`-excludeCluster`** Export excludes certain clusters, specifically those defined by a space-separated list.
- **`-includeInfo`** Defines whether the export includes database information.
- **`-includeClusterDefinitions`** Defines whether the export includes cluster definitions.
- **`-includeSchema`** Defines whether the export includes the database schema.
- **`-includeSecurity`** Defines whether the export includes database security parameters.
- **`-includeRecords`** Defines whether the export includes record contents.
- **`-includeIndexDefinitions`** Defines whether the export includes the database index definitions.
- **`-includeManualIndexes`** Defines whether the export includes manual index contents.
- **`-compressionLevel`** Defines the compression level to use on the export, in a range between `0` (no compression) and `9` (maximum compression). The default is `1`. (Feature introduced in version 1.7.6.)
- **`-compressionBuffer`** Defines the compression buffer size in bytes to use in compression. The default is 16kb. (Feature introduced in version 1.7.6.)

Examples

- Export the current database, including everything:

```
orientdb> EXPORT DATABASE C:\temp\petshop.export

Exporting current database to: C:\temp\petshop.export...

Exporting database info...OK
Exporting dictionary...OK
Exporting schema...OK
Exporting clusters...
- Exporting cluster 'metadata' (records=11) -> ..........OK
- Exporting cluster 'index' (records=0) -> OK
- Exporting cluster 'default' (records=779) -> OK
- Exporting cluster 'csv' (records=1000) -> OK
- Exporting cluster 'binary' (records=1001) -> OK
- Exporting cluster 'person' (records=7) -> OK
- Exporting cluster 'animal' (records=5) -> OK
- Exporting cluster 'animalrace' (records=0) -> OK
- Exporting cluster 'animaltype' (records=1) -> OK
- Exporting cluster 'orderitem' (records=0) -> OK
- Exporting cluster 'order' (records=0) -> OK
- Exporting cluster 'city' (records=3) -> OK
Export of database completed.
```

- Export the current database, including only its functions:

```
orientdb> EXPORT DATABASE functions.gz -includeClass=OFunction -includeInfo=FALSE
          -includeClusterDefinitions=FALSE -includeSchema=FALSE
          -includeIndexDefinitions=FALSE -includeManualIndexes=FALSE
```

- Alternatively, you can simplify the above by excluding all, then including only those features that you need. For instance, export the current database, including only the schema:

```
orientdb> EXPORT DATABASE schema.gz -excludeALL -includeSchema=TRUE
```

Export API

In addition to the Console, you can also trigger exports through Java and any other language that runs on the JVM, by using the ODatabaseExport class.

For example:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/temp/mydb");
db.open("admin", "admin");
try{
  OCommandOutputListener listener = new OCommandOutputListener() {
    @Override
    public void onMessage(String iText) {
      System.out.print(iText);
    }
  };

  ODatabaseExport export = new ODatabaseExport(db, "/temp/export", listener);
  export.exportDatabase();
  export.close();
} finally {
  db.close();
}
```

For more information on backups and restores, imports and exports, see the following commands:

- IMPORT DATABASE
- BACKUP DATABASE
- RESTORE DATABASE

as well as the following pages:

- Export File Format
- `ODatabaseExport` Java Class

For more information on other commands, see Console Commands.

Console - `EXPORT RECORD`

Exports the current record, using the requested format. In the event that you give a format that OrientDB does not support, it provides a list of supported formats.

Syntax

```
EXPORT RECORD <format>
```

- `<format>` Defines the export format you want to use.

Examples

- Use `SELECT` to create a record for export:

```
orientdb> SELECT name, surname, parent, children, city FROM Person WHERE
        name='Barack' AND surname='Obama'

---+-----+--------+---------+--------+------------+------
 # | RID | name   | surname | parent | children   | city
---+-----+--------+---------+--------+------------+------
 0 | 5:4 | Barack | Obama   | null   | [5:5, 5:6] | -6:2
---+-----+--------+---------+--------+------------+------
```

- Export JSON data from this record:

```
orientdb> EXPORT RECORD JSON

{
  'name': 'Barack',
  'surname': 'Obama',
  'parent': null,
  'children': [5:5, 5:6],
  'city': -6:2
}
```

- Use a bad format value to determine what export formats are available on your database:

```

orientdb> EXPORT RECORD GIBBERISH

ERROR: Format 'GIBBERISH' was not found.
Supported formats are:
- json
- ORecordDocument2csv
```

> For more information on other commands, see Console Commands.

Console - `FREEZE DATABASE`

Flushes all cached content to disk and restricts permitted operations to read commands. With the exception of reads, none of the commands made on a frozen database execute. It remains in this state until you run the `RELEASE` command.

Executing this command requires server administration rights. You can only execute it on remote databases. If you would like to freeze or release a local database, use the `ODatabase.freeze()` and `ODatabase.release()` methods directly through the OrientDB API.

> You may find this command useful in the event that you would like to perform backups on a live database. To do so, freeze the database, perform a file system snapshot, then release the database. You can now copy the snapshot anywhere you want.
>
> This works best when the backup doesn't take very long to run.

Syntax

```
FREEZE DATABASE
```

Example

- Freezes the current database:

  ```
  orientdb> FREEZE DATABASE
  ```

> To unfreeze a database, use the `RELEASE DATABASE` command.
>
> For more information on other commands, see SQL and Console commands.

Console - `GET`

Returns the value of the requested property.

Syntax

```
GET <property-name>
```

- `<property-name>` Defines the name of the property.

Example

- Find the default limit on your database:

  ```
  orientdb> GET LIMIT

  limit = 20
  ```

> To display all available properties configured on your database, use the `PROPERTIES` command.
>
> For more information on other commands, see Console Commands.

Console - `GREMLIN`

Executes commands in the Gremlin language from the Console.

Gremlin is a graph traversal language. OrientDB supports it from the Console, API and through a Gremlin shell launched from `$ORIENTDB_HOME/bin/gremlin.sh`.

Syntax

```
GREMLIN <command>
```

- **`<command>`** Defines the commands you want to know.

> **NOTE**: OrientDB parses Gremlin commands as multi-line input. It does not execute the command until you type `end`. Bear in mind, the `end` here is case-sensitive.

Examples

- Create a vertex using Gremlin:

```
orientdb> gremlin
[Started multi-line command.  Type just 'end' to finish and execute.]
orientdb> v1 = g.addVertex();
orientdb> end

v[#9:0]

Script executed in 0,100000 sec(s).
```

> For more information on the Gremlin language, see Gremlin. For more information on other commands, see Console Commands.

Console - `IMPORT`

Imports an exported database into the current one open. Import process doesn't lock the database, so any concurrent operations are allowed, but they could interfer in the import process causing errors.

The input file must use the JSON Export Format, as generated by the `EXPORT` command. By default, this file is compressed using the GZIP algorithm.

With `EXPORT`, this command allows you to migrate between releases without losing data, by exporting data from the old version and importing it into the new version.

Syntax

```
IMPORT DATABASE <input-file> [-format = <format>]
                             [-preserveClusterIDs = <true|false>]
                             [-deleteRIDMapping = <true|false>]
                             [-merge = <true|false>]
                             [-migrateLinks = <true|false>]
                             [-rebuildIndexes = <true|false>]
```

- **`<format>`** Is the input file format. If not specified, OrientDB tries to recognize it. The available formats are (since v2.2.8):
  - **orientdb**, the OrientDB export file format
  - **graphml**, for Graph XML
  - **graphson**, for Graph JSON
- **`<inputy-file>`** Defines the path to the file you want to import.
- **`-preserveClusterIDs`** Defines whether you want to preserve cluster ID's during the import. When turned off, the import creates temporary cluster ID's, which can sometimes fail. This option is only valid with PLocal storage.
- **`-deleteRIDMapping`** Defines whether you want to preserve the dictionary index used by the import to map old RIDs to new RIDs. The index name is `___exportImportRIDMap` and you could use in your application. By default the index is removed after the import.
- **`-merge`** Defines whether you want to merge the import with the data already in the current database. When turned off, the default, the import overwrites current data, with the exception of security classes, (`ORole` , `OUser` , `OIdentity`), which it always preserves. This feature was introduced in version 1.6.1.
- **`-migrateLinks`** Defines whether you want to migrate links after the import. When enabled, this updates all references from the old links to the new Record ID's. By default, it is enabled. Advisable that you only turn it off when merging and you're certain no other existent records link to those you're importing. This feature was introduced in version 1.6.1.
- **`-rebuildIndexes`** Defines whether you want to rebuild indexes after the import. By default, it does. You can set it to false to speed up the import, but do so only when you're certain the import doesn't affect indexes. This feature was introduced in version 1.6.1.

Example

- Import the database `petshop.export` :

```
orientdb> IMPORT DATABASE C:/temp/petshop.export -preserveClusterIDs=true

Importing records...
- Imported records into the cluster 'internal': 5 records
- Imported records into the cluster 'index': 4 records
- Imported records into the cluster 'default': 1022 records
- Imported records into the cluster 'orole': 3 records
- Imported records into the cluster 'ouser': 3 records
- Imported records into the cluster 'csv': 100 records
- Imported records into the cluster 'binary': 101 records
- Imported records into the cluster 'account': 1005 records
- Imported records into the cluster 'company': 9 records
- Imported records into the cluster 'profile': 9 records
- Imported records into the cluster 'whiz': 1000 records
- Imported records into the cluster 'address': 164 records
- Imported records into the cluster 'city': 55 records
- Imported records into the cluster 'country': 55 records
- Imported records into the cluster 'animalrace': 3 records
- Imported records into the cluster 'ographvertex': 102 records
- Imported records into the cluster 'ographedge': 101 records
- Imported records into the cluster 'graphcar': 1 records
```

> For more information on backups, restores, and exports, see: `BACKUP`, `RESTORE` and `EXPORT` commands, and the `ODatabaseImport` Java class. For the JSON format, see Export File Format.
>
> For more information on other commands, see Console Commands.

Import API

In addition to the Console, you can also manage imports through the Java API, and with any language that runs on top of the JVM, using the `ODatabaseImport` class.

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/temp/mydb");
db.open("admin", "admin");
try{
  OCommandOutputListener listener = new OCommandOutputListener() {
    @Override
    public void onMessage(String iText) {
      System.out.print(iText);
    }
  };

  ODatabaseImport import = new ODatabaseImport(db, "/temp/export/export.json.gz", listener);
  import.importDatabase();
  import.close();
} finally {
  db.close();
}
```

Troubleshooting

Validation Errors

Occasionally, you may encounter validation errors during imports, usually shown as an `OValidationException` exception. Beginning with version 2.2, you can disable validation at the database-level using the `ALTER DATABASE` command, to allow the import to go through.

1. Disable validation for the current database:

```
orientdb> ALTER DATABASE validation false
```

2. Import the exported database:

```
orientdb> IMPORT DATABASE /path/to/my_data.export -preserveClusterIDs=TRUE
```

3. Re-enable validation:

```
orientdb> ALTER DATABASE validation true
```

Cluster ID's

During imports you may occasionally encounter an error that reads: `Imported cluster 'XXX' has id=6 different from the original: 5`. Typically occurs in databases that were created in much older versions of OrientDB. You can correct it using the `DROP CLASS` on the class `ORIDs`, then attempting the import again.

1. Import the database:

```
orientdb> IMPORT DATABASE /path/to/old_data.export

Importing records...
- Creating cluster 'company'...Error on database import happened just before line
16, column 52 com.orientechnologies.orient.core.exception.OConfigurationException:
Imported cluster 'company has id=6 different from the original: 5 at
com.orientechnologies.orient.core.db.tool.ODatabaseImport.importClusters(
ODatabaseImport.java:500) at
com.orientechnologies.orient.core.db.tool.ODatabaseIMport.importDatabase(
ODatabaseImport.java:121)
```

2. Drop the `ORIDs` class:

```
orientdb> DROP CLASS ORIDs
```

3. Import the database:

```
 orientdb> IMPORT DATABASE /path/to/old_data.export
```

The database now imports without error.

Console - `INDEXES`

Displays all indexes in the current database.

Syntax

```
INDEXES
```

Example

- Display indexes in the current database:

  ```
  orientdb {db=GratefulDeadConcerts}> INDEXES

  INDEXES
  --------------+------------+-------+--------+---------
   NAME         | TYPE       | CLASS | FIELDS | RECORDS
  --------------+------------+-------+--------+---------
   dictionary   | DICTIONARY |       |        |       0
   Group.Grp_Id | UNIQUE     | Group | Grp_Id |       1
   ORole.name   | UNIQUE     | ORole | name   |       3
   OUser.name   | UNIQUE     | OUser | name   |       4
  --------------+------------+----------------+---------
   TOTAL = 4                                          8
  -----------------------------------------------------
  ```

> For more information on other commands, see Console Commands.

Console - `INFO`

Displays all information on the current database.

Syntax

```
INFO
```

Example

- Display information on database `petshop` :

```
orientdb {db=petshop}> INFO

Current database: ../databases/petshop/petshop
CLUSTERS:
------------+------+----------+----------
 NAME       |  ID  | TYPE     | ELEMENTS
------------+------+----------+----------
 metadata   |    0 | Physical |       11
 index      |    1 | Physical |        0
 default    |    2 | Physical |      779
 csv        |    3 | Physical |     1000
 binary     |    4 | Physical |     1001
 person     |    5 | Physical |        7
 animal     |    6 | Physical |        5
 animalrace |   -2 | Logical  |        0
 animaltype |   -3 | Logical  |        1
 orderitem  |   -4 | Logical  |        0
 order      |   -5 | Logical  |        0
 city       |   -6 | Logical  |        3
------------+------+----------+----------
 TOTAL                              2807
-----------------------------------------

CLASSES:
------------+----+------------+----------
 NAME       | ID | CLUSTERS   | ELEMENTS
------------+----+------------+----------
 Person     |  0 | person     |        7
 Animal     |  1 | animal     |        5
 AnimalRace |  2 | AnimalRace |        0
 AnimalType |  3 | AnimalType |        1
 OrderItem  |  4 | OrderItem  |        0
 Order      |  5 | Order      |        0
 City       |  6 | City       |        3
------------+----+------------+----------
 TOTAL                                16
-----------------------------------------
```

> For more information on other commands, see Console Commands.

Console - `INFO CLASS`

Displays all information on given class.

Syntax

```
INFO CLASS <class-name>
```

- **`<class-name>`** Defines what class you want information on.

Example

- Display information on class `Profile`

  ```
  orientdb> INFO CLASS Profile

  Default cluster......: profile (id=10)
  Supported cluster ids: [10]
  Properties:
  --------+----+----------+-----------+---------+-----------+----------+-----+----
   NAME   | ID | TYPE     | LINK TYPE | INDEX   | MANDATORY | NOT NULL | MIN | MAX
  --------+----+----------+-----------+---------+-----------+----------+-----+----
   nick   |  3 | STRING   | null      |         | false     | false    | 3   | 30
   name   |  2 | STRING   | null      |NOTUNIQUE| false     | false    | 3   | 30
   surname|  1 | STRING   | null      |         | false     | false    | 3   | 30
   ...    |    | ...      | ...       | ...     | ...       | ...      |...  | ...
   photo  |  0 | TRANSIENT| null      |         | false     | false    |     |
  --------+----+----------+-----------+---------+-----------+----------+-----+----
  ```

> For more information on other commands, see Console Commands.

Console - `INFO PROPERTY`

Displays all information on the given property.

Syntax

```
INFO PROPERTY <class-name>.<property-name>
```

- **`<class-name>`** Defines the class to which the property belongs.
- **`<property-name>`** Defines the property you want information on.

Example

- Display information on the property `name` in the class `OUser`:

```
orientdb> INFO PROPERTY OUser.name

PROPERTY 'OUser.name'

Type.................: STRING
Mandatory............: true
Not null.............: true
Read only............: false
Default value........: null
Minimum value........: null
Maximum value........: null
REGEXP...............: null
Collate..............: {OCaseInsensitiveCollate : name = ci}
Linked class.........: null
Linked type..........: null

INDEXES (1 altogether)
--------------------+------------
 NAME               | PROPERTIES
--------------------+------------
 OUser.name         | name
--------------------+------------
```

> For more information on other commands, see Console Commands.

Console - `INSERT`

Inserts a new record into the current database. Remember, OrientDB can work in schema-less mode, meaning that you can create any field on the fly.

Syntax

```
INSERT INTO <<class-name>|CLUSTER:<cluster-name>> (<field-names>) VALUES ( <field-values> )
```

- `<class-name>` Defines the class you want to create the record in.
- `CLUSTER:<cluster-name>` Defines the cluster you want to create the record in.
- `<field-names>` Defines the fields you want to add the records to, in a comma-separated list.
- `<field-values>` Defines the values you want to insert, in a comma-separated list.

Examples

- Insert a new record into the class `Profile`, using the name `Jay` and surname `Miner`:

  ```
  orientdb> INSERT INTO Profile (name, surname) VALUES ('Jay', Miner')

  Inserted record in 0,060000 sec(s).
  ```

- Insert a new record into the class `Employee`, while defining a relationship:

  ```
  orientdb> INSERT INTO Employee (name, boss) VALUES ('Jack', 11:99)
  ```

- Insert a new record, adding a collection of relationships:

  ```
  orientdb> INSERT INTO Profile (name, friends) VALUES ('Luca', [10:3, 10:4])
  ```

> For more information on other commands, see SQL and Console commands.

Console - `JS`

Executes commands in the Javascript language from the Console. Look also Javascript Command.

Syntax

```
JS <commands>
```

- **`<commands>`** Defines the commands you want to execute.

Interactive Mode You can execute a command in just one line (`JS print('Hello World!')`) or enable the interactive input by just executing `JS` and then typing the Javascript expression as multi-line inputs. It does not execute the command until you type `end`. Bear in mind, the `end` here is case-sensitive.

Examples

- Execute a query and display the result:

```
orientdb> js
[Started multi-line command.  Type just 'end' to finish and execute.]
orientdb> var r = db.query('select from ouser');
orientdb> for(var i=0;i
orientdb> print( r[i] );
orientdb> }
orientdb> end

OUser#5:0{roles:[1],status:ACTIVE,password:{PBKDF2WithHmacSHA256}C08CE0F5160EA4050B8F10EDBB86F06EB0A2EE82DF73A340:BC1B604
0727C1E11E3A961A1B2A49615C96938710AF17ADD:65536,name:admin} v1
OUser#5:1{name:reader,password:{PBKDF2WithHmacSHA256}41EF9B675430D215E0970AFDEB735899B6665DF44A29FE98:5BC48B2D20752B12B5E
32BE1F22C6C85FF7CCBEFB318B826:65536,status:ACTIVE,roles:[1]} v1
OUser#5:2{name:writer,password:{PBKDF2WithHmacSHA256}FA0AD7301EA2DB371355EB2855D63F4802F13858116AB82E:18B8077E1E63A45DB0A
3347F91E03E4D2218EA16E5100105:65536,status:ACTIVE,roles:[1]} v1

Client side script executed in 0.142000 sec(s). Value returned is: null
```

> For more information on the Javascript execution, see Javascript Command. For more information on other commands, see Console Commands.

Console - `JSS`

Executes commands on OrientDB Server in the Javascript language from the Console. Look also Javascript Command.

Syntax

```
JSS <commands>
```

- `<commands>` Defines the commands you want to execute.

Interactive Mode You can execute a command in just one line (`JSS print('Hello World!')`) or enable the interactive input by just executing `JSS` and then typing the Javascript expression as multi-line inputs. It does not execute the command until you type `end` . Bear in mind, the `end` here is case-sensitive.

Examples

- Execute a query and display the result:

```
orientdb> jss
[Started multi-line command.  Type just 'end' to finish and execute.]
orientdb> var r = db.query('select from ouser');
orientdb> for(var i=0;i
orientdb> print( r[i] );
orientdb> }
orientdb> end

Server side script executed in 0.146000 sec(s). Value returned is: null
```

In this case the output will be displayed on the server console.

> For more information on the Javascript execution, see Javascript Command. For more information on other commands, see Console Commands.

Console - `LIST DATABASES`

Displays all databases hosted on the current server. Note that this command requires you connect to the OrientDB Server.

Syntax

```
LIST DATABASES
```

Example

- Connect to the server:

  ```
  orientdb> CONNECT REMOTE:localhost admin admin_password
  ```

- List the databases hosted on the server:

  ```
  orientdb {server=remote:localhost/}> LIST DATABASES

  Found 4 databases:

  * ESA (plocal)
  * Napster (plocal)
  * Homeland (plocal)
  * GratefulDeadConcerts (plocal)
  ```

> For more information on other commands, see Console Commands.

Console - `LIST CONNECTIONS`

Displays all active connections to the OrientDB Server. Command introduced in version 2.2. The connections as per server, so you should connect to the server, not to the database.

Syntax

```
LIST CONNECTIONS
```

Permissions

In order to enable a user to execute this command, you must add `"server.info"` as resource to the server user.

Example

- List the current connections to the OrientDB Server:

```
orientdb {server=remote:localhost/}> LIST CONNECTIONS

---+----+--------------+------+-------------------+--------+-----+--------+--------
 # | ID |REMOTE_ADDRESS|PROTOC|LAST_OPERATION_ON  |DATABASE|USER |COMMAND |TOT_REQS
---+----+--------------+------+-------------------+--------+-----+--------+--------
 0 | 17 |/127.0.0.1    |binary|2015-10-12 19:22:34|-       |-    |info    | 1
 1 | 16 |/127.0.0.1    |binary|1970-01-01 01:00:00|-       |-    |-       | 0
 5 | 1  |/127.0.0.1    |http  |1970-01-01 00:59:59|pokec   |admin|Listen  | 32
---+----+--------------+------+-------------------+--------+-----+--------+--------
```

> For more information on other commands, see Console Commands.

Console - `LOAD RECORD`

Loads a record the given Record ID from the current database.

Syntax

```
LOAD RECORD <record-id>
```

- `<record-id` Defines the Record ID of the record you want to load.

In the event that you don't have a Record ID, execute a query to find the one that you want.

Example

- Load the record for `#5:5` :

  ```
  orientdb> LOAD RECORD #5:5

  --------------------------------------------------------------------------------
   Class: Person   id: #5:5   v.0
  --------------------------------------------------------------------------------
     parent : Person@5:4{parent:null,children:[Person@5:5, Person@5:6],name:Barack,
              surname:Obama,city:City@-6:2}
   children : null
       name : Malia Ann
    surname : Obama
       city : null
  --------------------------------------------------------------------------------
  ```

> For more information on other commands, see Console Commands.

Console - `LOAD SCRIPT` (from 2.2.18)

Loads a sql script from the given path and executes it.

Syntax

```
LOAD SCRIPT <script path>
```

Example

- Load a script from an absolute path:

```
orientdb> LOAD SCRIPT /path/to/scripts/data.osql
```

- Launch the console in batch mode and load script to a remote database:

```
$ $ORIENTDB_HOME/bin/console.sh "CONNECT REMOTE:localhost/demo;LOAD SCRIPT /path/to/scripts/data.osql"
```

> For more information on other commands, see Console Commands.

Console - `PROFILER`

Controls the Profiler.

Syntax

```
PROFILER ON|OFF|DUMP|RESET
```

- `ON` Turn on the Profiler and begin recording.
- `OFF` Turn off the Profiler and stop recording.
- `DUMP` Dump the Profiler data.
- `RESET` Reset the Profiler data.

Example

- Turn the Profiler on:

  ```
  orientdb> PROFILER ON

  Profiler is ON now, use 'profiler off' to turn off.
  ```

- Dump Profiler data:

  ```
  orientdb> PROFILER DUMP
  ```

> For more information on other commands, see Console Commands.

Console - `PROPERTIES`

Displays all configured properties.

Syntax

```
PROPERTIES
```

Example

- List configured properties:

```
orientdb> PROPERTIES

PROPERTIES:
------------------------+-----------
 NAME                   | VALUE
------------------------+-----------
 limit                  | 20
 backupBufferSize       | 1048576
 backupCompressionLevel | 9
 collectionMaxItems     | 10
 verbose                | 2
 width                  | 150
 maxBinaryDisplay       | 150
 debug                  | false
 ignoreErrors           | false
------------------------+-----------
```

> To change a property value, use the `SET` command.
>
> For more information on other commands, see Console Commands.

Console - `RELEASE DATABASE`

Releases database from a frozen state, from where it only allows read operations back to normal mode. Execution requires server administration rights.

You may find this command useful in the event that you want to perform live database backups. Run the `FREEZE DATABASE` command to take a snapshot, you can then copy the snapshot anywhere you want. Use such approach when you want to take short-term backups.

Syntax

```
RELEASE DATABASE
```

Example

- Release the current database from a freeze:

  ```
  orientdb> RELEASE DATABASE
  ```

> To freeze a database, see the `FREEZE DATABASE` command.
>
> For more information on other commands, see Console and SQL commands.

Console - `RELOAD RECORD`

Reloads a record from the current database by its Record ID, ignoring the cache.

You may find this command useful in cases where external applications change the record and you need to see the latest update.

Syntax

```
RELOAD RECORD <record-id>
```

- **`<record-id>`** Defines the unique Record ID for the record you want to reload. If you don't have the Record ID, execute a query first.

Examples

- Reload record with the ID of `5:5`:

```
orientdb> RELOAD RECORD 5:5

------------------------------------------------------------------------
Class: Person   id: 5:5   v.0
------------------------------------------------------------------------
   parent : Person@5:4{parent:null,children:[Person@5:5, Person@5:6],
            name:Barack,surname:Obama,city:City@-6:2}
 children : null
     name : Malia Ann
  surname : Obama
     city : null
------------------------------------------------------------------------
```

> For more information on other commands, see Console Commands.

Console - `REPAIR DATABASE`

Repairs a database. To check if a database needs to be repaired, you can use the Check Database Command.

Syntax

```
REPAIR DATABASE [--fix-graph [-skipVertices=<vertices>] [-skipEdges=<edges>]]
                [--fix-links]
                [--fix-ridbags]
                [--fix-bonsai]
                [-v]
```

- **`[--fix-graph]`** Fixes the database as graph. All broken edges are removed and all corrupted vertices repaired. This mode takes the following optional parameters:
  - **`-skipVertices=<vertices>`**, where `<vertices>` are the number of vertices to skip on repair.
  - **`-skipEdges=<edges>`**, where `<edges>` are the number of edges to skip on repair.
- **`[--fix-links]`** Fixes links. It removes any reference to not existent records. The optional `[-v]` tells to print more information.
- **`[--fix-ridbags]`** Fixes the ridbag structures only (collection of references).
- **`[--fix-bonsai]`** Fixes the bonsai structures only (internal representation of trees)
- **`[-v]`** Verbose mode

Examples

- Repair a graph database:

```
  orientdb> REPAIR DATABASE --fix-graph

  Repair of graph 'plocal:/temp/demo' is started ...
  Scanning 26632523 edges (skipEdges=0)...
  ...
  + edges: scanned 100000, removed 0 (estimated remaining time 10 secs)
  + edges: scanned 200000, removed 0 (estimated remaining time 9 secs)
  + deleting corrupted edge friend#40:22044{out:#25:1429,in:#66:1,enabled:true} v7 because missing incoming vertex (#66:1)
  ...
  Scanning edges completed
  Scanning 32151775 vertices...
  + vertices: scanned 100000, repaired 0 (estimated remaining time 892 secs)
  + vertices: scanned 200000, repaired 0 (estimated remaining time 874 secs)
  + vertices: scanned 300000, repaired 0 (estimated remaining time 835 secs)
  + repaired corrupted vertex Account#25:961{out_friend:[],dateUpdated:Wed Aug 12 19:00:00 CDT 2015,createdOn:Wed Aug 12 19:00
:00 CDT 2015} v4
  ...
  + vertices: scanned 32100000, repaired 47 (estimated remaining time 2 secs)
  ...
  Scanning vertices completed
  Repair of graph 'plocal:/temp/demo' completed in 2106 secs
   scannedEdges.....: 1632523
   removedEdges.....: 129
   scannedVertices..: 32151775
   scannedLinks.....: 53264852
   removedLinks.....: 64
   repairedVertices.: 47
```

> For more information on other commands, see Console Commands.

Console - `RESTORE DATABASE`

Restores a database from a backup. It must be done against a new database. It does not support restores that merge with an existing database. If you need to backup and restore to an existing database, use the `EXPORT DATABASE` and `IMPORT DATABASE` commands.

OrientDB Enterprise Edition version 2.2 and major, support incremental backup.

To create a backup file to restore from, use the `BACKUP DATABASE` command.

Syntax

```
RESTORE DATABASE <backup-file>|<incremental-backup-directory>
```

- **`<backup-file>`** Defines the database file you want to restore.
- **`<incremental-backup-directory>`** Defines the database directory you want to restore from an incremental backup. Available only in OrientDB Enterprise Edition version 2.2 and major.

Example of full restore

- Create a new database to receive the restore:

  ```
  orientdb> CREATE DATABASE PLOCAL:/tmp/mydb
  ```

- Restore the database from the `mydb.zip` backup file:

  ```
  orientdb {db=/tmp/mydb}> RESTORE DATABASE /backups/mydb.zip
  ```

Example of incremental restore

This is available only in OrientDB Enterprise Edition version 2.2 and major.

- Open a database to receive the restore:

  ```
  orientdb> CONNECT PLOCAL:/tmp/mydb
  ```

- Restore the database from the `/backup` backup directory:

  ```
  orientdb {db=/tmp/mydb}> RESTORE DATABASE /backup
  ```

> For more information, see the `BACKUP DATABASE`, `EXPORT DATABASE`, `IMPORT DATABASE` commands. For more information on other commands, see Console Commands.

Restore API

In addition to the console commands, you can also execute restores through the Java API or with any language that can run on top of the JVM using the `restore()` method against the database instance.

```
db.restore(in, options, callable, listener);
```

- **`in`** Defines the `InputStream` used to read the backup content. Uses a `FileInputStream` to read the backup content from disk.
- **`options`** Defines backup options, such as `Map<String, Object>` object.
- **`callable`** Defines the callback to execute when the database is locked.
- **`listener`** Listener called for backup messages.
- **`compressionLevel`** Defines the Zip Compression level, between `0` for no compression and `9` for maximum compression. The greater the compression level, the smaller the final backup content and the greater the CPU and time it takes to execute.

- `bufferSize` Buffer size in bytes, the greater the buffer the more efficient the compression.

Example

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/temp/mydb");
db.open("admin", "admin");
try{
  OCommandOutputListener listener = new OCommandOutputListener() {
    @Override
    public void onMessage(String iText) {
      System.out.print(iText);
    }
  };

  InputStream out = new FileInputStream("/temp/mydb.zip");
  db.restore(in,null,null,listener);
} finally {
   db.close();
}
```

Console - `ROLLBACK`

Aborts a transaction, rolling the database back to its save point.

Syntax

```
BEGIN
```

> For more information on transactions, see Transactions. To initiate a transaction, use the `BEGIN` command. To save changes, see `COMMIT` command.

Example

- Initiate a new transaction:

```
orientdb> BEGIN

Transaction 1 is running
```

- Attempt to start a new transaction, while another is open:

```
orientdb> BEGIN

Error: an active transaction is currently open (id=1). Commit or rollback before
starting a new one.
```

- Make changes to the database:

```
orientdb> INSERT INTO Account (name) VALUES ('tx test')

Inserted record 'Account#9:-2{name:tx test} v0' in 0,004000 sec(s).
```

- View changes in database:

```
orientdb> SELECT FROM Account WHERE name LIKE 'tx%'

---+-------+--------------------
 # | RID   | name
---+-------+--------------------
 0 | #9:-2 | tx test
---+-------+--------------------
1 item(s) found. Query executed in 0.076 sec(s).
```

- Abort the transaction:

```
orientdb> ROLLBACK

Transaction 1 has been rollbacked in 4ms
```

- View rolled back database:

```
orientdb> SELECT FROM Account WHERE name LIKE 'tx%'

0 item(s) found. Query executed in 0.037 sec(s).
```

> For more information on other commands, see Console Commands.

Console - `SET`

Changes the value of a property.

Syntax

```
SET <property-name> <property-value>
```

- **`<property-name>`** Defines the name of the property
- **`<property-value>`** Defines the value you want to change the property to.

Example

- Change the `LIMIT` property to one hundred:

  ```
  orientdb> SET LIMIT 100

  Previous value was: 20
  limit = 100
  ```

> To display all properties use the `PROPERTIES` command. To display the value of a particular property, use the `GET` command.
>
> For more information on other commands, see Console Commands.

Console - `SET SERVER USER`

Creates a server user. If the server user already exists, it updates the password and permissions.

In order to create or modify the user, the current system user must have write permissions on the `$ORIENTDB_HOME/config/orientdb-server-config.xml` configuration file.

Syntax

```
SET SERVER USER <user-name> <user-password> <user-permissions>
```

- **`<user-name>`** Defines the server username.
- **`<user-password>`** Defines the password for the server user.
- **`<user-permissions>`** Defines the permissions for the server user.

> For more information on security, see OrientDB Server Security. Feature introduced in version 2.2.

Example

- Create the server user `editor`, give it all permissions:

  ```
  orientdb> SET SERVER USER editor my_password *

  Server user 'editor' set correctly
  ```

> To display all server users, see the `LIST SERVER USERS` command. To remove a server user, see `DROP SERVER USER` command.
>
> For more information on other commands, see Console Commands.

Console - `SLEEP`

Pauses the console for the given amount a time. You may find this command useful in working with batches or to simulate latency.

Syntax

```
SLEEP <time>
```

- `<time>` Defines the time the Console should pause in milliseconds.

Example

- Pause the console for three seconds:

  ```
  orientdb {server=remote:localhost/}> SLEEP 3000
  ```

> For more information on other commands, see Console Commands.

Upgrading

OrientDB uses the Semantic Versioning System (http://semver.org), where the version numbers follow this format MAJOR.MINOR.PATCH, Here are the meanings of the increments:

- MAJOR version entails incompatible API changes,
- MINOR version entails functionality in a backward-compatible manner
- PATCH version entails backward-compatible bug fixes.

So between PATCH versions, the compatibility is assured (example 2.1.0 -> 2.1.8). Between MINOR and MAJOR versions, you may need to export and re-import the database. To find out if your upgrade must be done over exporting and importing the database, see below in the column "Database":

Compatibility Matrix

FROM	TO	Guide	Blueprints	Database	Binary Protocol	HTTP Protocol
2.1.x	2.2.x	Release 2.2.x	Final v2.6.0	Automatic	34	10
2.0.x	2.1.x	Release 2.1.x	Final v2.6.0	Automatic	30	10
1.7.x	2.0.x	Migration-from-1.7.x-to-2.0.x	Final v2.6.0	Automatic	25	10
1.6.x	1.7.x	Migration-from-1.6.x-to-1.7.x	Final v2.5.0	Automatic	20, 21	10
1.5.x	1.6.x	Migration-from-1.5.x-to-1.6.x	Changed v2.5.x	Automatic	18, 19	10
1.4.x	1.5.x	Migration-from-1.4.x-to-1.5.x	Changed v2.4.x	Automatic	16, 17	10
1.3.x	1.4.x	Migration-from-1.3.x-to-1.4.x	Changed v2.3.x	Automatic	14, 15	n.a.

Instructions

The easiest way to upgrade a database from one version of OrientDB to the next is to plan ahead for future upgrades from the beginning.

The recommended strategy is to store databases separately from the OrientDB installation, often on a separate data volume.

As an example, you might have a data volume, called `/data`, with a `databases` directory under that where all of your database directories will be stored.

```
/data/databases:
MyDB
WebAppDB
```

New Databases

If you're just starting with OrientDB or just creating new databases, from the OrientDB installation directory, you can remove the `databases` directory and then create a symbolic link to the `/data/databases` directory previously created.

Make sure OrientDB is not running.

On a Linux system, you could call `rm -rf databases` to remove the *databases* directory and recursively remove all sub-directories.

DO NOT issue that command if you have any live databases that have not been moved!

Once the `databases` directory is removed, create a symbolic link to the `/data/databases` directory.

On a Linux system, you could call `ln -s /data/databases ./databases`.

On a Windows system, you can use `mklink /d .\databases d:\data\databases`, assuming your data volume resides as the `d:` drive in Windows. The `/d` switch creates a directory symbolic link. You can use `/j` instead to create a directory junction.

You should now be able to start OrientDB and create new databases that will reside under `/data/databases`.

Upgrading Symbolic-Linked Databases

If you used a similar symbolic link scheme as suggested in the prior section, upgrading OrientDB is very easy. Simply follow the same instructions to remove the `databases` directory from the **NEW** OrientDB installation directory and then create a symbolic link to `/data/databases`.

Upgrading Existing Databases

If you've been running OrientDB in the standard way, with the `databases` directory stored directly under the OrientDB installation directory, then you have two options when upgrading to a newer version of OrientDB.

1. You can move your database directories to the `databases` directory under the new installation.
2. You can move your database directories to an external location and then set-up a symbolic link from the new installation.

Moving Databases to a New Installation

Make sure OrientDB is not running. From the old OrientDB installation location, move each database directory under `databases` to the `databases` directory in the new OrientDB installation.

Moving Databases to an External Location

Make sure OrientDB is not running. Using the earlier example of `/data/databases`, from the old OrientDB installation location, move each database directory under `databases` to the `/data/databases` location.

Now, follow the instructions under New Databases to create a symbolic link from within the new OrientDB installation to the `/data/databases` directory.

External Storage Advantages

If you store your databases separately from the OrientDB installation, not only will you be able to upgrade more easily in the future, but you may even be able to improve performance by using a data volume that is mounted on a disk that's faster than the volume where the main installation resides.

Also, many cloud-based providers support creating snapshots of data volumes, which can be a useful way of backing up all of your databases.

Backward Compatibility

OrientDB supports binary compatibility between previous releases and latest release. Binary compatibility is supported at least between last 2 minor versions.

For example, lets suppose that we have following releases 1.5, 1.5.1, 1.6.1, 1.6.2, 1.7, 1.7.1 then binary compatibility at least between 1.6.1, 1.6.2, 1.7, 1.7.1 releases will be supported.

If we have releases 1.5, 1.5.1, 1.6.1, 1.6.2, 1.7, 1.7.1, 2.0 then binary compatibility will be supported at least between releases 1.7, 1.7.1, 2.0.

Binary compatibility feature is implemented using following algorithm:

1. When storage is opened, version of binary format which is used when storage is created is read from storage configuration.
2. Factory of objects are used to present disk based data structures for current binary format is created.

Only features and database components which were exist at the moment when current binary format was latest one will be used. It means that you can not use all database features available in latest release if you use storage which was created using old binary format version. It also means that bugs which are fixed in new versions may be (but may be not) reproducible on storage created using old binary format.

To update binary format storage to latest one you should export database in JSON format and import it back. Using either console commands export database and import database or Java API look at `com.orientechnologies.orient.core.db.tool.ODatabaseImport` , `com.orientechnologies.orient.core.db.tool.ODatabaseExport` classes and `com.orientechnologies.orient.test.database.auto.DbImportExportTest` test.

- Current binary format version can be read from `com.orientechnologies.orient.core.db.record.OCurrentStorageComponentsFactory#binaryFormatVersion` proporty.
- Instance of `OCurrentStorageComponentsFactory` class can be retrieved by call of `com.orientechnologies.orient.core.storage.OStorage#getComponentsFactory` method.
- Latest binary format version can be read from here `com.orientechnologies.orient.core.config.OStorageConfiguration#CURRENT_BINARY_FORMAT_VERSION` .

Please note that binary compatibility is supported since **1.7-rc2** version for plocal storage (as exception you can read database created in 1.5.1 version by 1.7-rc2 version).

Return to Upgrade.

Release 2.2.x

What's new?

NOTE: Release 2.2.6 introduced a change in the internal distributed protocol that does not allow for a cluster of OrientDB Servers, with version between 2.2.0 and 2.2.5, to be hot migrated to a release 2.2.6 or major. The solution is to stop the entire cluster and then execute the upgrade.

Spatial Module

OrientDB v2.2 offers a brand new module to handle geospatial information provided as external plugin. Look at Spatial Index.

Pattern Matching

Starting from v2.2, OrientDB provides an alternative way to query the database by using the Pattern Matching approach. For more information look at SQL Match.

Non-Stop Incremental Backup and Restore

OrientDB Enterprise Edition allows Non-Stop Incremental Backup and Restore.

Distributed

Release v2.2 contains many improvement on distributed part. First of all there is a huge improvement on performance. With 2 nodes we measured 5x better and with 3 nodes is about 10x faster than 2.1!

Management of the Quorum

Before 2.2, the `writeQuorum` was scaled down to `1` because the setting `failureAvailableNodesLessQuorum` (that now is no longer supported).

This wasn't correct, because if a node is unreachable, it could be because network temporary error, split brain, etc. So downgrading the `writeQuorum` means no guarantee for consistency when 2 nodes see each other again, because both nodes thought to be the only one and they continue working with quorum=1 with evident merge conflict risks.

In v2.2.0-rc1 the nodes is never removed automatically from the configuration for this reason, unless you manually remove a node from the configuration claiming that node is not part of the cluster anymore. The new SQL command to remove a server from the configuration is:

```
HA REMOVE SERVER <server-name>
```

Other changes

- Multi-Threads message management
- Static Ownership of clusters
- 'majority' and 'all' quorum to assure you have the majority (N/2+1) or the total of the consensus
- Removed `failureAvailableNodesLessQuorum` setting: with majority you don't need this setting anymore
- Removed `hotAlignment` setting: servers, once they join the cluster, remain always in the configuration until they are manually removed
- Server Roles, where you can specify a node is a read only "REPLICA"
- Load balancing on the client side
- OrientDB doesn't use Hazelcast Queues to exchange messages between nodes, but rather the OrientDB binary protocol
- New SQL commands to manage the distributed configuration:
  - `HA REMOVE SERVER <server-name>` , to remove a server from the configuration
  - `HA SYNC DATABASE` , to ask for a resync of the database
  - `HA SYNC CLUSTER <cluster-name>` , to ask for a resync of a single cluster

Command Cache

OrientDB 2.2 has a new component called Command Cache, disabled by default, but that can make a huge difference in performance on some use cases. Look at Command Cache to know more.

Sequences

In v2.2 we introduced Sequences. Thanks to the sequences it's easy to maintain counters and incremental ids in your application. You can use Sequences from both Java API and SQL.

Parallel queries

OrientDB v2.2 can run query in parallel, using multiple threads. To use parallel queries, append the `PARALLEL` keyword at the end of SQL SELECT. Example: `SELECT FROM V WHERE amount < 100 PARALLEL` .

Starting from v2.2, the OrientDB SQL executor can decide if execute or not a query in parallel, only if `query.parallelAuto` setting is enabled. To tune parallel query execution these are the new settings:

- `query.parallelAuto` enable automatic parallel query, if requirements are met. By default is false.
- `query.parallelMinimumRecords` is the minimum number of records to activate parallel query automatically. Default is 300,000.
- `query.parallelResultQueueSize` is the size of the queue that holds results on parallel execution. The queue is blocking, so in case the queue is full, the query threads will be in a wait state. Default is 20,000 results.

Automatic usage of Multiple clusters

Starting from v2.2, when a class is created, the number of underlying clusters will be the number of cores. Issue 4518.

Encryption at rest

OrientDB v2.2 can encrypt database at file system level by using DES and AES encryption.

New ODocument.eval()

To execute quick expression starting from a ODocument and Vertex/Edge objects, use the new `.eval()` method. The old syntax `ODocument.field("city[0].country.name")` has been deprecated, but still supported. Issue 4505.

Security

OrientDB v2.2 comes with a plethora of new security features, including a new centralized security module, external authenticators (including Kerberos), LDAP import of users, password validation, enhanced auditing features, support for syslog events, using a salt with password hashes, and a new *system user*.

security.json

The new security module uses a JSON configuration file, located at `config\security.json` .

External Authenticators

OrientDB v2.2 supports external authentication, meaning that authentication of database and server users can occur outside the database and server configuration. Kerberos/SPNEGO authentication is now fully supported.

LDAP Import

As part of the new security module, LDAP users can be imported automatically into OrientDB databases (including the new system database) using LDAP filters.

Password Validator

Password validation is now fully supported, including the ability to specify minimum length and the number of uppercase, special, and numeric characters.

Auditing

Auditing is no longer an Enterprise-only feature and supports many new auditing events, including the creation and dropping of classes, reloading of configuration files, and distributed node events. Additionally, if the new syslog plugin is installed, auditing events will also be recorded to syslog.

Salt

OrientDB v2.2 increases security by using SALT. This means that hashing of password is much slower than OrientDB v2.1. You can configure the number of cycles for the SALT: more is harder to decode but is slower. Change the setting `security.userPasswordSaltIterations` to the number of cycles. Default is 65k cycles. The default password hashing algorithm is now `PBKDF2WithHmacSHA256` this is not present in any environment so you can change it setting `security.userPasswordDefaultAlgorithm` possible alternatives values are `PBKDF2WithHmacSHA1` or `SHA-256`

System User

As part of the new "system database" implementation, OrientDB v2.2 offers a new kind of user, called the System User. A *system user* is like a hybrid between a server user and a database user, meaning that a system user can have permissions and roles assigned like a database user but it can be applied to the entire system not just a single database.

System Database

OrientDB now uses a "system database" to provide additional capabilities.

The system database, currently named *OSystem*, is created when the OrientDB server starts, if the database does not exist.

Here's a list of some of the features that the system database may support:

- A new class of user called the *system user*
- A centralized location for configuration files
- Logging of global auditing events
- Recording performance metrics about the server and its databases

Direct Memory

Starting from v2.2, OrientDB uses direct memory. The new server.sh (and .bat) already set the maximum size value to 512GB of memory by setting the JVM configuration

```
-XX:MaxDirectMemorySize=512g
```

If you run OrientDB embedded or with a different script, please set `MaxDirectMemorySize` to a high value, like `512g`.

NULL Values in Indexes

Starting from v2.2, by default any new index created will not ignore NULL values; null values will be indexed as any other values. This means that if you have a UNIQUE index, you cannot have multiple NULL keys. This applies only to the new indexes, opening an old database with indexes previously created, will all ignore NULL by default.

To create an index that explicitly ignore nulls (like the default with v2.1 and earlier), look at the following examples by usinng SQL or Java API.

SQL:

```
CREATE INDEX addresses ON Employee (address) NOTUNIQUE METADATA {ignoreNullValues: true}
```

And Java API:

```
schema.getClass(Employee.class).getProperty("address").createIndex(OClass.INDEX_TYPE.NOTUNIQUE, new ODocument().field("ignoreN
ullValues",true));
```

API changes

ODocument.field()

To execute quick expression starting from a ODocument and Vertex/Edge objects, use the new `.eval()` method. The old syntax `ODocument.field("city[0].country.name")` is still supported. Issue 4505.

Schema.dropClass()

On drop class are dropped all the cluster owned by the class, and not just the default cluster.

Configuration Changes

Since 2.2 you can force to not ask for a root password setting `<isAfterFirstTime>true</isAfterFirstTime>` inside the `<orient-server>` element in the orientdb-server-config.xml file.

SQL and Console commands Changes

Strict SQL parsing is now applied also to statements for **Schema Manipulation** (CREATE CLASS, ALTER CLASS, CREATE PROPERTY, ALTER PROPERTY etc.)

ALTER DATABASE: A statement like

```
ALTER DATABASE dateformat yyyy-MM-dd
```

is correctly executed, but is interpreted in the WRONG way: the `yyyy-MM-dd` is interpreted as an expression (two subtractions) and not as a single date format. Please re-write it as (see quotes)

```
ALTER DATABASE dateformat 'yyyy-MM-dd'
```

CREATE FUNCTION

In some cases a variant the syntax with curly braces was accepted (not documented), eg.

```
CREATE FUNCTION testCreateFunction {return 'hello '+name;} PARAMETERS [name] IDEMPOTENT true LANGUAGE Javascript
```

Now it's not supported anymore, the right syntax is

```
CREATE FUNCTION testCreateFunction "return 'hello '+name;" PARAMETERS [name] IDEMPOTENT true LANGUAGE Javascript
```

ALTER PROPERTY

The ALTER PROPERTY command, in previous versions, accepted any unformatted value as last argument, eg.

```
ALTER PROPERTY Foo.name min 2015-01-01 00:00:00
```

In v.2.2 the value must be a valid expression (eg. a string):

```
ALTER PROPERTY Foo.name min "2015-01-01 00:00:00"
```

CREATE USER and **DROP USER**

In v2.2 we introduced new specific commands to work with users.

Migration from 2.1.x to 2.2.x

Databases created with release 2.1.x are compatible with 2.2.x, so you don't have to export/import the database.

Release 2.1.x

What's new?

Live Query

OrientDB 2.1 includes the first experimental version of LiveQuery. See details here.

Migration from 2.0.x to 2.1.x

Databases created with release 2.0.x are compatible with 2.1, so you don't have to export/import the database.

Difference function

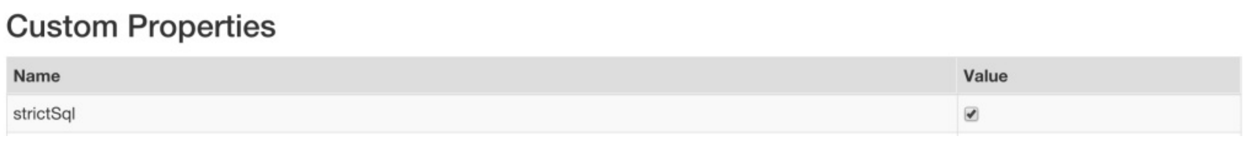
In 2.0.x difference() function had inconsistent behavior: it actually worked as a symmetric difference (see 4366, 3969) In 2.1 it was refactored to perform normal difference (https://proofwiki.org/wiki/Definition:Set_Difference) and another function was created for symmetric difference (called "symmetricDifference()").

If for some reason you application relied on the (wrong) behavior of difference() function, please change your queries to invoke symmetricDifference() instead.

Strict SQL parser

V 2.1 introduces a new implementation of the new SQL parser. This implementation is more strict, so some queries that were allowed in 2.0.x could not work now.

For backward compatibility, you can disable the new parser from Studio -> DB -> Configuration -> remove the flag from strictSql (bottom right of the page).



Custom Properties

Name	Value
strictSql	✓

Or via console by executing this command, just once:

```
ALTER DATABASE custom strictSql=false
```

Important improvements of the new parser are:

- full support for named (:param) and unnamed (?) input parameters: now you can use input parameters almost everywhere in a query: in subqueries, function parameters, between square brackets, as a query target
- better management of blank spaces and newline characters: the old parser was very sensitive to presence or absence of blank spaces (especially in particular points, eg. before and after square brackets), now the problem is completely fixed
- strict validation: the old parser in some cases failed to detect invalid queries (eg. a macroscopic example was a query with two WHERE conditions, like SELECT FORM Foo WHERE a = 2 WHERE a = 3), now all these problems are completely fixed

Writing the new parser was a good opportunity to validate our query language. We discovered some ambiguities and we had to remove them. Here is a short list of these problems and how to manage them with the new parser:

- `-` as a valid character for identifiers (property and class names): in the old implementation you could define a property name like "simple-name" and do `SELECT simple-name FROM Foo` . This is not allowed anymore, because `-` character is used for arithmetic operations (subtract). To use names with `-` character, use backticks. Example: ``SELECT `simple-name` FROM Foo``
- reserved keywords as identifiers: words like `select` , `from` , `where` ... could be used as property or class name, eg. this query was valid `SELECT FROM FROM FROM` . In v 2.1 all the reserved keywords have to be quoted with a backtick to be used as valid

identifiers: `SELECT `FROM` FROM `FROM``

Object database

Before 2.1 entity class cache was static, so you could not manage multiple OObjectDatabase connections in the same VM. In 2.1 registerEntityClass() works at storage level, so you can open multiple OObjectDatabase connections in the same VM.

IMPORTANT: in 2.1 if you close and re-open the storage, you have to re-register your POJO classes.

Distributed architecture

Starting from release 2.1.6 it's not possible to hot upgrade a distributed architecture node by node, because the usage of the last recent version of Hazelcast that breaks such network compatibility. If you're upgrading a distributed architecture you should power off the entire cluster and restart it with the new release.

API changes

ODatabaseDocumentTx.activateOnCurrentThread()

If by upgading to v2.1 you see errors of kind "Database instance is not set in current thread...", this means that you used the same ODatabase instance across multiple threads. This was always forbidden, but some users did it with unpredictable results and random errors. For this reason in v2.1 OrientDB always checks that the ODatabase instance was bound to the current thread.

We introduced a new API to allow moving a ODatabase instance across threads. Before to use a ODatabase instance call the method `ODatabaseDocumentTx.activateOnCurrentThread()` and the ODatabase instance will be bound to the current thread. Example:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal/temp/mydb").open("admin", "admin");
new Thread(){
  public void run() {
    db.activateOnCurrentThread(); // <---- BINDS THE DATABASE ON CURRENT THREAD
    db.command(new OCommandSQL("select from MyProject where thisSummerIsVeryHot = true")).execute();
  }
}.start();
```

Migration from 1.7.x to 2.0.x

Databases created with release 1.7.x are compatible with 2.0, so you don't have to export/import the database like in the previous releases. Check your database directory: if you have a file *.wal, delete it before migration.

Use the new binary serialization

To use the new binary protocol you have to export and reimport the database into a new one. This will boost up your database performance of about +20% against old database.

To export and reimport your database follow these steps:

1) Stop any OrientDB server running

2) Open a new shell (Linux/Mac) or a Command Prompt (Windows)

2) Export the database using the console. Move into the directory where you've installed OrientDB 2.0 and execute the following commands:

```
> cd bin
> ./console.sh (or bin/console.bat under Windows)
orientdb> CONNECT plocal:/temp/mydb admin admin
orientdb> EXPORT DATABASE /temp/mydb.json.gz
orientdb> DISCONNECT
orientdb> CREATE DATABASE plocal:/temp/newdb
orientdb> IMPORT DATABASE /temp/mydb.json.gz
```

Now your new database is: /temp/newdb.

API changes

ODocument pin() and unpin() methods

We removed pin() and unpin() methods to force the cache behavior.

ODocument protecting of internal methods

We have hidden some methods considered internal to avoid users call them. However, if your usage of OrientDB is quite advanced and you still need them, you can access from Internal helper classes. Please still consider them as internals and could change in the future. Below the main ones:

- ORecordAbstract.addListener(), uses ORecordListenerManager.addListener() instead

ODatabaseRecord.getStorage()

We moved getStorage() method to ODatabaseRecordInternal.

ODatabaseDocumentPool

We replaced ODatabaseDocumentPool Java class (now deprecated) with the new, more efficient com.orientechnologies.orient.core.db.OPartitionedDatabasePool.

Caches

We completely removed Level2 cache. Now only Level1 and Storage DiskCache are used. This change should be transparent with code that run on previous versions, unless you enable/disable Level2 cache in your code.

Furthermore it's not possible anymore to disable Cache, so method `setEnable()` has been removed.

Changes

Context	1.7.x	2.0.x
API	ODatabaseRecord.getLevel1Cache()	ODatabaseRecord.getLocalCache()
API	ODatabaseRecord.getLevel2Cache()	Not available
Configuration	OGlobalConfiguration.CACHE_LEVEL1_ENABLED	OGlobalConfiguration.CACHE_LOCAL_ENABLED
Configuration	OGlobalConfiguration.CACHE_LEVEL2_ENABLED	Not available

No more LOCAL engine

We completely dropped the long deprecated LOCAL Storage. If your database were created using "LOCAL:" then you have to export it with the version you were using, then import it in a fresh new database created with OrientDB 2.0.

Server

First run ask for root password

At first run, OrientDB asks for the root's password. Leave it blank to auto generate it (like with 1.7.x). This is the message:

```
+---------------------------------------------------+
|          WARNING: FIRST RUN CONFIGURATION         |
+---------------------------------------------------+
| This is the first time the server is running.     |
| Please type a password of your choice for the     |
| 'root' user or leave it blank to auto-generate it. |
+---------------------------------------------------+

Root password [BLANK=auto generate it]: _
```

If you set the system setting or environment variable `ORIENTDB_ROOT_PASSWORD`, then its value will be taken as root password. If it's defined, but empty, a password will be automatically generated.

Distributed

First run ask for node name

At first run as distributed, OrientDB asks for the node name. Leave it blank to auto generate it (like with 1.7.x). This is the message:

```
+---------------------------------------------------+
|    WARNING: FIRST DISTRIBUTED RUN CONFIGURATION    |
+---------------------------------------------------+
| This is the first time that the server is running  |
| as distributed. Please type the name you want      |
| to assign to the current server node.              |
+---------------------------------------------------+

Node name [BLANK=auto generate it]: _
```

If you set the system setting or environment variable `ORIENTDB_NODE_NAME`, then its value will be taken as node name. If it's defined, but empty, a name will be automatically generated.

Multi-Master replication

With OrientDB 2.0 each record cluster selects assigns the first server node in the `servers` list node as master for insertion only. In 99% of the cases you insert per class, not per cluster. When you work per class, OrientDB auto-select the cluster where the local node is the master. In this way we completely avoid conflicts (like in 1.7.x).

Example of configuration with 2 nodes replicated (no sharding):

```
INSERT INTO Customer (name, surname) VALUES ('Jay', 'Miner')
```

If you execute this command against a node1, OrientDB will assign the cluster-id where node1 is master, i.e. #13:232. With node2 would be different: it couldn't never be #13.

For more information look at: http://www.orientechnologies.com/docs/last/orientdb.wiki/Distributed-Sharding.html.

Asynchronous replication

OrientDB 2.0 supports configurable execution mode through the new variable **`executionMode`**. It can be:

- `undefined`, the default, means synchronous
- `synchronous`, to work in synchronous mode
- `asynchronous`, to work in asynchronous mode

```
{
    "autoDeploy": true,
    "hotAlignment": false,
    "executionMode": "undefined",
    "readQuorum": 1,
    "writeQuorum": 2,
    "failureAvailableNodesLessQuorum": false,
    "readYourWrites": true,
    "clusters": {
        "internal": {
        },
        "index": {
        },
        "*": {
            "servers" : [ "<NEW_NODE>" ]
        }
    }
}
```

Set to "asynchronous" to speed up the distributed replication.

Graph API

Multi-threading

Starting from OrientDB 2.0, instances of both classes OrientGraph and OrientGraphNoTx can't be shared across threads. Create and destroy instances from the same thread.

Edge collections

OrientDB 2.0 disabled the auto scale of edge. In 1.7.x, if a vertex had 1 edge only, a LINK was used. As soon as a new edge is added the LINK is auto scaled to a LINKSET to host 2 edges. If you want this setting back you have to call these two methods on graph instance (or OrientGraphFactory before to get a Graph instance):

```
graph.setAutoScaleEdgeType(true);
graph.setEdgeContainerEmbedded2TreeThreshold(40);
```

Migration from 1.6.x to 1.7.x

Databases created with release 1.6.x are compatible with 1.7, so you don't have to export/import the database like in the previous releases.

Engine

OrientDB 1.7 comes with the PLOCAL engine as default one. For compatibility purpose we still support "local" database, but this will be removed soon. So get the chance to migrate your old "local" database to the new "plocal" follow the steps in: Migrate from local storage engine to plocal.

Migration from 1.5.x to 1.6.x

Databases created with release 1.5.x need to be exported and reimported in OrientDB 1.6.x.

From OrientDB 1.5.x:

- Open the console under "bin/" directory calling:
  - > ./console.sh (or .bat on Windows)
- Connect to the database and export it, example:
  - orientdb> connect plocal:/temp/db admin admin
  - orientdb> export database /temp/db.zip
- Run OrientDB 1.6.x console
  - > ./console.sh (or .bat on Windows)
- Create a new database and import it, example:
  - orientdb> create database plocal:/temp/db admin admin plocal
  - orientdb> import database /temp/db.zip

For any problem on import, look at Import Troubleshooting.

Engine

OrientDB 1.6.x comes with the new PLOCAL engine. To migrate a database create with the old "local" to such engine follow the steps in: Migrate from local storage engine to plocal

Migration from 1.4.x to 1.5.x

OrientDB 1.5.x automatic upgrades any databases created with version 1.4.x, so export and import is not needed.

Engine

OrientDB 1.5.x comes with the new PLOCAL engine. To migrate to such engine follow the steps in: Migrate from local storage engine to plocal.

Migration from 1.3.x to 1.4.x

GraphDB

OrientDB 1.4.x uses a new optimized structure to manage graphs. You can use the new OrientDB 1.4.x API against graph databases created with OrientDB 1.3.x setting few properties at database level. In this way you can continue to work with your database but remember that this doesn't use the new structure so it's strongly suggested to export and import the database.

The new Engine uses some novel techniques based on the idea of a dynamic Graph that change shape at run-time based on the settings and content. The new Engine is much faster than before and needs less space in memory and disk. Below the main improvements:

- avoid creation of edges as document if haven't properties. With Graphs wit no properties on edges this can save more than 50% of space on disk and therefore memory with more chances to have a big part of database in cache. Furthermore this speed up traversal too because requires one record load less. As soon as the first property is set the edge is converted transparently
- Vertex "in" and "out" fields aren't defined in the schema anymore because can be of different types and change at run-time adapting to the content:
    - no connection = null (no space taken)
    - 1 connection = store as LINK (few bytes)
    - > 1 connections = use the Set of LINKS (using the MVRBTreeRIDSet class)
- binding of Blueprints "label" concept to OrientDB sub-classes. If you create an edge with label "friend", then the edge sub-type "friend" will be used (created by the engine transparently). This means: 1 field less in document (the field "label") and therefore less space and the ability to use the technique 1 (see above)
- edges are stored on different files at file system level because are used different clusters
- better partitioning against multiple disks (and in the future more parallelism)
- direct queries like "select from friend" rather than "select from E" and then filtering the result-set looking for the edge with the wanted label property
- multiple properties for edges of different labels. Not anymore a "in" and "out" in Vertex but "out_friend" to store all the outgoing edges of class "friend". This means faster traversal of edges giving one or multiple labels avoiding to scan the entire Set of edges to find the right one

Blueprints changes

If you was using Blueprints look also to the Blueprints changes 1.x and 2.x.

Working with database created with 1.3.x

Execute these commands against the open database:

```
ALTER DATABASE custom useLightweightEdges=false
ALTER DATABASE custom useClassForEdgeLabel=false
ALTER DATABASE custom useClassForVertexLabel=false
ALTER DATABASE custom useVertexFieldsForEdgeLabels=false
```

Base class changed for Graph elements

Before 1.4.x the base classes for Vertices was "OGraphVertex" with alias "V" and for Edges was "OGraphEdge" with alias "E". Starting from v1.4 the base class for Vertices is "V" and "E" for Edges. So if in your code you referred "V" and "E" for inheritance nothing is changed (because "V" and "E" was the aliases of OGraphVertex and "OGraphEdge"), but if you used directly "OGraphVertex" and "OGraphEdge" you need to replace them into "V" and "E".

If you don't export and import the database you can rename the classes by hand typing these commands:

```
ALTER CLASS OGraphVertex shortname null
ALTER CLASS OGraphVertex name V
ALTER CLASS OGraphEdge shortname=null
ALTER CLASS OGraphEdge name E
```

Export and re-import the database

Use GREMLIN and GraphML format.

If you're exporting the database using the version 1.4.x you've to set few configurations at database level. See above Working with database created with 1.3.x.

Export the database

```
$ cd $ORIENTDB_HOME/bin
$ ./gremlin.sh

         \,,,/
         (o o)
-----oOOo-(_)-oOOo-----
gremlin> g = new OrientGraph("local:/temp/db");
==>orientgraph[local:/temp/db]
gremlin> g.saveGraphML("/temp/export.xml")
==>null
```

Import the exported database

```
gremlin> g = new OrientGraph("local:/temp/newdb");
==>orientgraph[local:/temp/newdb]
gremlin> g.loadGraphML("/temp/export.xml");
==>null
gremlin>
```

Your new database will be created under "/temp/newdb" directory.

General Migration

If you want to migrate from release 1.3.x to 1.4.x you've to export the database using the 1.3.x and re-import it using 1.4.x. Example:

Export the database using 1.3.x

```
$ cd $ORIENTDB_HOME/bin
$ ./console.sh
OrientDB console v.1.3.0 - www.orientechnologies.com
Type 'help' to display all the commands supported.

orientdb> CONNECT local:../databases/mydb admin admin
Connecting to database [local:../databases/mydb] with user 'admin'...
OK

orientdb> EXPORT DATABASE /temp/export.json.gz
Exporting current database to: database /temp/export.json.gz...

Started export of database 'mydb' to /temp/export.json.gz...
Exporting database info...OK
Exporting clusters...OK (24 clusters)
Exporting schema...OK (23 classes)
Exporting records...
- Cluster 'internal' (id=0)...OK (records=3/3)
- Cluster 'index' (id=1)...OK (records=0/0)
- Cluster 'manindex' (id=2)...OK (records=1/1)
- Cluster 'default' (id=3)...OK (records=0/0)
- Cluster 'orole' (id=4)...OK (records=3/3)
- Cluster 'ouser' (id=5)...OK (records=3/3)
- Cluster 'ofunction' (id=6)...OK (records=1/1)
- Cluster 'oschedule' (id=7)...OK (records=0/0)
- Cluster 'orids' (id=8)............OK (records=428/428)
- Cluster 'v' (id=9)............OK (records=809/809)
- Cluster 'e' (id=10)...OK (records=0/0)
- Cluster 'followed_by' (id=11)............OK (records=7047/7047)
- Cluster 'sung_by' (id=12)...OK (records=2/2)
- Cluster 'written_by' (id=13)...OK (records=1/1)
- Cluster 'testmodel' (id=14)...OK (records=2/2)
- Cluster 'vertexwithmandatoryfields' (id=15)...OK (records=1/1)
- Cluster 'artist' (id=16)...OK (records=0/0)
- Cluster 'album' (id=17)...OK (records=0/0)
- Cluster 'track' (id=18)...OK (records=0/0)
- Cluster 'sing' (id=19)...OK (records=0/0)
- Cluster 'has' (id=20)...OK (records=0/0)
- Cluster 'person' (id=21)...OK (records=2/2)
- Cluster 'restaurant' (id=22)...OK (records=1/1)
- Cluster 'eat' (id=23)...OK (records=0/0)

Done. Exported 8304 of total 8304 records

Exporting index info...
- Index dictionary...OK
OK (1 indexes)
Exporting manual indexes content...
- Exporting index dictionary ...OK (entries=0)
OK (1 manual indexes)

Database export completed in 1913ms
```

Re-import the exported database using OrientDB 1.4.x:

```
$ cd $ORIENTDB_HOME/bin
$ ./console.sh
OrientDB console v.1.3.0 - www.orientechnologies.com
Type 'help' to display all the commands supported.

orientdb> CREATE DATABASE local:../databases/newmydb admin admin local

Creating database [local:../databases/newmydb] using the storage type [local]...
Database created successfully.

Current database is: local:../databases/newmydb

orientdb> IMPORT DATABASE /temp/export.json.gz
Importing database database /temp/export.json.gz...

Started import of database 'local:../databases/newmydb' from /temp/export.json.gz...
Importing database info...OK
Importing clusters...
- Creating cluster 'internal'...OK, assigned id=0
- Creating cluster 'default'...OK, assigned id=3
- Creating cluster 'orole'...OK, assigned id=4
- Creating cluster 'ouser'...OK, assigned id=5
- Creating cluster 'ofunction'...OK, assigned id=6
- Creating cluster 'oschedule'...OK, assigned id=7
- Creating cluster 'orids'...OK, assigned id=8
- Creating cluster 'v'...OK, assigned id=9
- Creating cluster 'e'...OK, assigned id=10
- Creating cluster 'followed_by'...OK, assigned id=11
- Creating cluster 'sung_by'...OK, assigned id=12
- Creating cluster 'written_by'...OK, assigned id=13
- Creating cluster 'testmodel'...OK, assigned id=14
- Creating cluster 'vertexwithmandatoryfields'...OK, assigned id=15
- Creating cluster 'artist'...OK, assigned id=16
- Creating cluster 'album'...OK, assigned id=17
- Creating cluster 'track'...OK, assigned id=18
- Creating cluster 'sing'...OK, assigned id=19
- Creating cluster 'has'...OK, assigned id=20
- Creating cluster 'person'...OK, assigned id=21
- Creating cluster 'restaurant'...OK, assigned id=22
- Creating cluster 'eat'...OK, assigned id=23
Done. Imported 22 clusters
Importing database schema...OK (23 classes)
Importing records...
- Imported records into cluster 'internal' (id=0): 3 records
- Imported records into cluster 'orole' (id=4): 3 records
- Imported records into cluster 'ouser' (id=5): 3 records
- Imported records into cluster 'internal' (id=0): 1 records
- Imported records into cluster 'v' (id=9): 809 records
- Imported records into cluster 'followed_by' (id=11): 7047 records
- Imported records into cluster 'sung_by' (id=12): 2 records
- Imported records into cluster 'written_by' (id=13): 1 records
- Imported records into cluster 'testmodel' (id=14): 2 records
- Imported records into cluster 'vertexwithmandatoryfields' (id=15): 1 records
- Imported records into cluster 'person' (id=21): 2 records

Done. Imported 7874 records

Importing indexes ...
- Index 'dictionary'...OK
Done. Created 1 indexes.
Importing manual index entries...
- Index 'dictionary'...OK (0 entries)
Done. Imported 1 indexes.
Delete temporary records...OK (0 records)

Database import completed in 2383 ms
orientdb>
```

Your new database will be created under "../databases/newmydb" directory.

Backup & Restore

OrientDB supports backup and and restore operations, like any database management system.

The `BACKUP DATABASE` command executes a complete backup on the currently open database. It compresses the backup the backup using the ZIP algorithm. To restore the database from the subsequent `.zip` file, you can use the `RESTORE DATABASE` command.

Backups and restores are much faster than the `EXPORT DATABASE` and `IMPORT DATABASE` commands. You can also automate backups using the Automatic Backup server plugin. Additionally, beginning with version 2.2 of Enterprise Edition OrientDB introduces major support for incremental backups.

> **NOTE**: OrientDB Community Edition does not support backing up remote databases. OrientDB Enterprise Edition does support this feature. For more information on how to implement this with Enterprise Edition, see Remote Backups.

Backups versus Exports

During backups, the `BACKUP DATABASE` command produces a consistent copy of the database. During this process, the database locks all write operations, waiting for the backup to finish. If you need perform reads and writes on the database during backups, set up a distributed cluster of nodes. To access to the non blocking backup feature, use the Enterprise Edition.

By contrast, the `EXPORT DATABASE` command doesn't lock the database, allowing concurrent writes to occur during the export process. Consequentially, the export may include changes made after you initiated the export, which may result in inconsistencies.

Using the Backup Script

Beginning in version 1.7.8, OrientDB introduces a `backup.sh` script found in the `$ORIENTDB_HOME/bin` directory. This script allows you to initiate backups from the system console.

Syntax

```
./backup.sh <db-url> <user> <password> <destination> [<type>]
```

- **`<db-url>`** Defines the URL for the database to backup.
- **`<user>`** Defines the user to run the backup.
- **`<password>`** Defines the password for the user.
- **`<destination>`** Defines the path to the backup file the script creates, (use the `.zip` extension).
- **`<type>`** Defines the backup type. Supported types:
  - *`default`* Locks the database during the backup.
  - *`lvm`* Executes an LVM copy-on-write snapshot in the background.

> **NOTE** Non-blocking backups require that the operating system support LVM. For more information, see
>
> - LVM
> - File system snapshots with LVM
> - LVM snapshot backup

Examples

- Backup a database opened using `plocal`:

  ```
  $ $ORIENTDB_HOME/bin/backup.sh plocal:../database/testdb \
        admin adminpasswd \
        /path/to/backup.zip
  ```

- Perform a non-blocking LVM backup, using `plocal`:

```
$ $ORIENTDB_HOME/bin/backup.sh plocal:../database/testdb \
      admin adminpasswd \
      /path/to/backup.zip \
      lvm
```

- Perform a backup using the OrientDB Console with the `BACKUP` command:

```
orientdb> CONNECT PLOCAL:../database/testdb/ admin adminpasswd
orientdb> BACKUP DATABASE /path/to/backup.zip

Backup executed in 0.52 seconds.
```

Restoring Databases

Once you have created your `backup.zip` file, you can restore it to the database either through the OrientDB Console, using the `RESTORE DATABASE` command.

```
orientdb> RESTORE DATABASE /backups/mydb.zip

Restore executed in 6.33 seconds
```

Bear in mind that OrientDB does not support merging during restores. If you need to merge the old data with new writes, instead use `EXPORT DATABASE` and `IMPORT DATABASE` commands, instead.

> For more information, see
>
> - `BACKUP DATABASE`
> - `RESTORE DATABASE`
> - `EXPORT DATABASE`
> - `IMPORT DATABASE`
> - Console Commands

Incremental Backup and Restore

(Since v2.2 - Enteprise Edition only)

An incremental backup generates smaller backup files by storing only the delta between two versions of the database. This is useful when you execute a backup on a regular basis and you want to avoid having to back up the entire database each time. The easiest way to execute a backup and a restore is using Studio.

NOTE: *This feature is available only in the OrientDB Enterprise Edition. If you are interested in a commercial license look at OrientDB Subscription Packages*.

NOTE: Lucene Indexes are not supported yet in the incremental backup/restore process. Once the incremental restore is finished the indexes rebuild is necessary see (here)[https://github.com/orientechnologies/orientdb/issues/5958]

See also

- Backup and Restore
- BACKUP DATABASE console command
- RESTORE DATABASE console command

How does it work?

Every time a backup is executed, OrientDB writes a file named `last-backup.json` in the database directory. This is an example of the content:

```
{
  "lsn": 8438432,
  "startedOn": "2015-08-17 10:33:23.943",
  "completedOn": "2015-08-17 10:33:45.121"
}
```

The most important information is the `lsn` field that is the WAL LSN (Last Serial Number). Thanks to this number, OrientDB is able to understand the last change in the database, so the next incremental backup will be done starting from last `lsn` + 1.

Executing an Incremental Backup

Incremental Backup via Console

Backup Database console command accepts `-incremental` as an optional parameter to execute an incremental backup. In this case the new backup is executed from the last backup (file `last-backup.json` is read if present). If this is the first incremental backup, a full backup is executed. Example:

```
orientdb> connect plocal:/databases/mydb admin admin
orientdb {db=Whisky}> backup database /tmp/backup -incremental
```

The incremental backup setting also allows you to specify an LSN version to start with. Example:

```
orientdb> connect plocal:/databases/mydb admin admin
orientdb {db=Whisky}> backup database /tmp/backup -incremental=93222
```

Incremental Backup via Java API

You can perform an incremental backup via the Java API too.

NOTE The `remote` protocol is supported, but the specified *path* is relative to the server.

If you are managing an ODocumentDatabase you have to call the `incrementalBackup()` method that accepts a String *path* parameter to the backup directory:

```
ODatabaseDocumentTx documentDatabase = new ODatabaseDocumentTx(dbURL);
documentDatabase.open("root", "password");
documentDatabase.incrementalBackup("/tmp/backup");
```

If you are using the OrientGraph interface you have to get the raw graph before calling the `incrementalBackup()` method:

```
OrientGraph graphDatabase = new OrientGraphNoTx(dbURL);
graphDatabase.open("root", "password");
graphDatabase.getRawGraph().incrementalBackup("/tmp/backup");
```

Executing an Incremental Restore

Incremental Restore via the Console

Restore Database console command automatically recognizes if a backup contains incremental data. Restoring an incremental backup creates a new database with the restored content. You cannot perform an in-place incremental restore on an existing database. The execution of the create database command with the option `-restore` builds a fresh database and performs the incremental restore starting from the backup path.

Example:

```
orientdb> create database remote:localhost/mydb root root plocal graph -restore=/tmp/backup

Creating database [remote:localhost/mydb] using the storage type [plocal]...
Connecting to database [remote:localhost/mydb] with user 'admin'...OK

Database created successfully.

Current database is: remote:localhost/mydb
```

Incremental Restore via the Java API

You can perform an incremental restore via the Java API too. To create a database from an incremental backup you can call from Java `ODatabase#create(path-to-incremental-backup-directory)` .

Incremental Restore in Distributed Architecture

The incremental restore affects only the local node where the restore command is executed.
Let's suppose we have 3 nodes and we execute an incremental restore on node1. If we execute an incremental restore on node1 a new fresh database is created on all the 3 nodes, but only on node1 the restore procedure is performed. Thus we obtain the database correctly restored on node1 but an empty database on node2 and node 3.
You can overcome this inconsistency by executing a shutdown on all the nodes of the cluster not involved in the restore procedure (node2 and node3 in our example), so once restarted they will get the full database from node1.

Distributed Architecture

The incremental backup is used in the Distributed Architecture when a server node restarts. This avoids having to backup and tranfer the entire database across the network.

Internals

File Format

In case of incremental backup, the content of the zip file is not the database directory, but rather meta files needed to update the database with the delta. Example of the content:

```
- Employee.pcl
- Person.pcl.incremental
- Person.pcm.incremental
```

This means only three files are changed. Employee.pcl is a full file, while the other two files with extension ".incremental" are incremental. Incremental files contain all the page changes and have the following format:

```
+----------------+-----------------+
| PAGE NUMBER    | PAGE CONTENT    |
| (long)         | byte[]          |
+----------------+-----------------+
```

Export and Import

OrientDB supports export and import operations, like any database management system.

The `EXPORT DATABASE` command exports the current opened database into a file. The exported file is in the Export JSON format. By default, it compresses the file using the GZIP algorithm.

Using exports with the `IMPORT DATABASE` command, you can migrate the database between different releases of OrientDB without losing data. When doing this, if you receive an error relating to the database version, export the database using the same version of OrientDB on which you created the database.

```
orientdb> EXPORT DATABASE /temp/petshop.export

Exporting current database to: /temp/petshop.export...

Exporting database info...OK
Exporting dictionary...OK
Exporting schema...OK
Exporting clusters...
- Exporting cluster 'metadata' (records=11) -> ...........OK
- Exporting cluster 'index' (records=0) -> OK
- Exporting cluster 'default' (records=779) -> OK
- Exporting cluster 'csv' (records=1000) -> OK
- Exporting cluster 'binary' (records=1001) -> OK
- Exporting cluster 'person' (records=7) -> OK
- Exporting cluster 'animal' (records=5) -> OK
- Exporting cluster 'animalrace' (records=0) -> OK
- Exporting cluster 'animaltype' (records=1) -> OK
- Exporting cluster 'orderitem' (records=0) -> OK
- Exporting cluster 'order' (records=0) -> OK
- Exporting cluster 'city' (records=3) -> OK
Export of database completed.
```

Exports versus Backups

Exports don't lock the database. Instead, they browse the contents. This means that OrientDB can execute concurrent operations during the export, but the exported database may not be an exact replica from the time when you issued the command. If you need a database snapshot, use backups.

The `BACKUP DATABASE` command does create a consistent copy of the database, but it locks the database. During the backup, the database remains in read-only mode, all concurrent write operations are blocked until the backup finishes. In the event that you need a database snapshot *and* the ability to perform read/write operations during the backup, set up a distributed cluster of nodes.

> **NOTE**: Even though the export file is 100% JSON, there are some constraints in the JSON format, where the field order must be kept. Modifying the file to adjust the indentation may make the file unusable in database imports.

Importing Databases

Once you have exported your database, you can import it using the `IMPORT DATABASE` command.

```
orientdb> IMPORT DATABASE /temp/petshop.export.gz -preserveClusterIDs=true

Importing records...
- Imported records into the cluster 'internal': 5 records
- Imported records into the cluster 'index': 4 records
- Imported records into the cluster 'default': 1022 records
- Imported records into the cluster 'orole': 3 records
- Imported records into the cluster 'ouser': 3 records
- Imported records into the cluster 'csv': 100 records
- Imported records into the cluster 'binary': 101 records
- Imported records into the cluster 'account': 1005 records
- Imported records into the cluster 'company': 9 records
- Imported records into the cluster 'profile': 9 records
- Imported records into the cluster 'whiz': 1000 records
- Imported records into the cluster 'address': 164 records
- Imported records into the cluster 'city': 55 records
- Imported records into the cluster 'country': 55 records
- Imported records into the cluster 'animalrace': 3 records
- Imported records into the cluster 'ographvertex': 102 records
- Imported records into the cluster 'ographedge': 101 records
- Imported records into the cluster 'graphcar': 1 records
```

> For more information, see
>
> - JSON Export Format
> - `RESTORE DATABASE`
> - `EXPORT DATABASE`
> - `IMPORT DATABASE`
> - Console Commands

Export Format

When you run the `EXPORT DATABASE` command, OrientDB exports the database into a zipped file using a special JSON format. When you run the `IMPORT DATABASE` command, OrientDB unzips the file and parses the JSON, making the import.

Sections

Export files for OrientDB use the following sections. Note that while the export format is 100% JSON, there are some constraints in the format, where the field order must be kept. Additionally, modifying the file to adjust the indentation (as has been done in the examples below), may make it unusable in database imports.

Info Section

The first section contains the resuming database information as well as all versions used during the export. OrientDB uses this information to check for compatibility during the import.

```
"info": {
  "name": "demo",
  "default-cluster-id": 2,
  "exporter-format": 2,
  "engine-version": "1.7-SNAPSHOT",
  "storage-config-version": 2,
  "schema-version": 4,
  "mvrbtree-version": 0
}
```

Parameter	Description	JSON Type
`"name"`	Defines the name of the database.	String
`"default-cluster-id"`	Defines the Cluster ID to use by default. Range: 0-32,762.	Integer
`"exporter-format"`	Defines the version of the database exporter.	Integer
`"engine-version"`	Defines the version of OrientDB.	String
`"storage-version"`	Defines the version of the Storage layer.	Integer
`"schema-version"`	Defines the version of the schema exporter.	Integer
`"mvrbtree-version"`	Defines the version of the MVRB-Tree.	Integer

Clusters Section

This section defines the database structure in clusters. It is formed from a list with an entry for each cluster in the database.

```
"clusters": [
  {"name": "internal", "id": 0, "type": "PHYSICAL"},
  {"name": "index", "id": 1, "type": "PHYSICAL"},
  {"name": "default", "id": 2, "type": "PHYSICAL"}
]
```

Parameter	Description	JSON Type
`"name"`	Defines the logical name of the cluster.	String
`"id"`	Defines the Cluster ID. Range: 0-32, 767.	Integer
`"type"`	Defines the cluster type: `PHYSICAL` , `LOGICAL` and `MEMORY` .	String

Schema Section

This section defines the database schema as classes and properties.

```
"schema":{
  "version": 210,
  "classes": [
    {"name": "Account", "default-cluster-id": 9, "cluster-ids": [9],
      "properties": [
        {"name": "binary", "type": "BINARY", "mandatory": false, "not-null": false},
        {"name": "birthDate", "type": "DATE", "mandatory": false, "not-null": false},
        {"name": "id", "type": "INTEGER", "mandatory": false, "not-null": false}
      ]
    }
  ]
}
```

Parameter	Description	JSON Type
`"version"`	Defines the version of the record storing the schema. Range: 0-2,147,483,647.	Integer
`"classes"`	Defines a list of entries for each class in the schema.	Array

Parameters for the Classes Subsection:

Parameter	Description	JSON Type
`"name"`	Defines the logical name of the class.	String
`"default-cluster-id"`	Defines the default Cluster ID for the class. It represents the cluster that stores the class records.	Integer
`"cluster-ids"`	Defines an array of Cluster ID's that store the class records. The first ID is always the default Cluster ID.	Array of Integers
`"properties"`	Defines a list of entries for each property for the class in the schema.	Array

Parameters for the Properties Sub-subsection:

Parameter	Description	JSON Type
`"name"`	Defines the logical name of the property.	String
`"type"`	Defines the property type.	String
`"mandatory"`	Defines whether the property is mandatory.	Boolean
`"not-null"`	Defines whether the property accepts a `NULL` value.	Boolean

Records Section

This section defines the exported record with metadata and fields. Entries for metadata are distinguished from fields by the `@` symbol.

```
"records": [
   {"@type": "d", "@rid": "#12:476", "@version": 0, "@class": "Account",
    "account_id": 476,
    "date": "2011-12-09 00:00:00:0000",
    "@fieldTypes": ["account_id=i", "date=t"]
   },
   {"@type": "d", "@rid": "#12:477", "@version": 0,    "@class": "Whiz",
    "id": 477,
    "date": "2011-12-09 00:00:00:000",
    "text": "He in office return He inside electronics for $500,000 Jay",
    "@fieldTypes": "date=t"
   }
]
```

Parameters for Metadata

Parameter	Description	JSON Type
`"@type"`	Defines the record-type: `d` for Document, `b` for Binary.	String
`"@rid"`	Defines the Record ID, using the format: `<cluster-id>:<cluster-position>` .	String
`"@version"`	Defines the record version. Range: 0-2, 147, 483, 647.	Integer
`"@class"`	Defines the logical class name for the record.	String
`"@fieldTypes"`	Defines an array of the types for each field in this record.	Any

Supported Field Types

Value	Type
`l`	Long
`f`	Float
`d`	Double
`s`	Short
`t`	Datetime
`d`	Date
`c`	Decimal
`b`	Byte

Full Example

```
{
  "info":{
    "name": "demo",
    "default-cluster-id": 2,
    "exporter-version": 2,
    "engine-version": "1.0rc8-SNAPSHOT",
    "storage-config-version": 2,
    "schema-version": 4,
    "mvrbtree-version": 0
  },
  "clusters": [
    {"name": "internal", "id": 0, "type": "PHYSICAL"},
    {"name": "index", "id": 1, "type": "PHYSICAL"},
    {"name": "default", "id": 2, "type": "PHYSICAL"},
    {"name": "orole", "id": 3, "type": "PHYSICAL"},
    {"name": "ouser", "id": 4, "type": "PHYSICAL"},
    {"name": "orids", "id": 5, "type": "PHYSICAL"},
    {"name": "csv", "id": 6, "type": "PHYSICAL"},
    {"name": "binary", "id": 8, "type": "PHYSICAL"},
    {"name": "account", "id": 9, "type": "PHYSICAL"},
    {"name": "company", "id": 10, "type": "PHYSICAL"},
    {"name": "profile", "id": 11, "type": "PHYSICAL"},
    {"name": "whiz", "id": 12, "type": "PHYSICAL"},
    {"name": "address", "id": 13, "type": "PHYSICAL"},
    {"name": "city", "id": 14, "type": "PHYSICAL"},
    {"name": "country", "id": 15, "type": "PHYSICAL"},
    {"name": "dummy", "id": 16, "type": "PHYSICAL"},
    {"name": "ographvertex", "id": 26, "type": "PHYSICAL"},
    {"name": "ographedge", "id": 27, "type": "PHYSICAL"},
    {"name": "graphvehicle", "id": 28, "type": "PHYSICAL"},
    {"name": "graphcar", "id": 29, "type": "PHYSICAL"},
    {"name": "graphmotocycle", "id": 30, "type": "PHYSICAL"},
    {"name": "newv", "id": 31, "type": "PHYSICAL"},
    {"name": "mappoint", "id": 33, "type": "PHYSICAL"},
    {"name": "person", "id": 35, "type": "PHYSICAL"},
    {"name": "order", "id": 36, "type": "PHYSICAL"},
    {"name": "post", "id": 37, "type": "PHYSICAL"},
    {"name": "comment", "id": 38, "type": "PHYSICAL"}
  ],
  "schema":{
    "version": 210,
    "classes": [
      {"name": "Account", "default-cluster-id": 9, "cluster-ids": [9],
        "properties": [
          {"name": "binary", "type": "BINARY", "mandatory": false, "not-null": false},
          {"name": "birthDate", "type": "DATE", "mandatory": false, "not-null": false},
          {"name": "id", "type": "INTEGER", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "Address", "default-cluster-id": 13, "cluster-ids": [13]
      },
      {"name": "Animal", "default-cluster-id": 17, "cluster-ids": [17]
      },
      {"name": "AnimalRace", "default-cluster-id": 18, "cluster-ids": [18]
      },
      {"name": "COMMENT", "default-cluster-id": 38, "cluster-ids": [38]
      },
      {"name": "City", "default-cluster-id": 14, "cluster-ids": [14]
      },
      {"name": "Company", "default-cluster-id": 10, "cluster-ids": [10], "super-class": "Account",
        "properties": [
        ]
      },
      {"name": "Country", "default-cluster-id": 15, "cluster-ids": [15]
      },
      {"name": "Dummy", "default-cluster-id": 16, "cluster-ids": [16]
      },
      {"name": "GraphCar", "default-cluster-id": 29, "cluster-ids": [29], "super-class": "GraphVehicle",
        "properties": [
        ]
      },
      {"name": "GraphMotocycle", "default-cluster-id": 30, "cluster-ids": [30], "super-class": "GraphVehicle",
        "properties": [
        ]
      },
      {"name": "GraphVehicle", "default-cluster-id": 28, "cluster-ids": [28], "super-class": "OGraphVertex",
        "properties": [
        ]
      },
      {"name": "MapPoint", "default-cluster-id": 33, "cluster-ids": [33],
        "properties": [
          {"name": "x", "type": "DOUBLE", "mandatory": false, "not-null": false},
          {"name": "y", "type": "DOUBLE", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "OGraphEdge", "default-cluster-id": 27, "cluster-ids": [27], "short-name": "E",
        "properties": [
          {"name": "in", "type": "LINK", "mandatory": false, "not-null": false, "linked-class": "OGraphVertex"},
          {"name": "out", "type": "LINK", "mandatory": false, "not-null": false, "linked-class": "OGraphVertex"}
        ]
      },
      {"name": "OGraphVertex", "default-cluster-id": 26, "cluster-ids": [26], "short-name": "V",
        "properties": [
          {"name": "in", "type": "LINKSET", "mandatory": false, "not-null": false, "linked-class": "OGraphEdge"},
          {"name": "out", "type": "LINKSET", "mandatory": false, "not-null": false, "linked-class": "OGraphEdge"}
        ]
      },
      {"name": "ORIDs", "default-cluster-id": 5, "cluster-ids": [5]
      },
      {"name": "ORole", "default-cluster-id": 3, "cluster-ids": [3],
        "properties": [
          {"name": "mode", "type": "BYTE", "mandatory": false, "not-null": false},
          {"name": "name", "type": "STRING", "mandatory": true, "not-null": true},
          {"name": "rules", "type": "EMBEDDEDMAP", "mandatory": false, "not-null": false, "linked-type": "BYTE"}
        ]
      },
      {"name": "OUser", "default-cluster-id": 4, "cluster-ids": [4],
        "properties": [
          {"name": "name", "type": "STRING", "mandatory": true, "not-null": true},
          {"name": "password", "type": "STRING", "mandatory": true, "not-null": true},
          {"name": "roles", "type": "LINKSET", "mandatory": false, "not-null": false, "linked-class": "ORole"}
        ]
      },
      {"name": "Order", "default-cluster-id": 36, "cluster-ids": [36]
      },
      {"name": "POST", "default-cluster-id": 37, "cluster-ids": [37],
        "properties": [
          {"name": "comments", "type": "LINKSET", "mandatory": false, "not-null": false, "linked-class": "COMMENT"}
        ]
      },
      {"name": "Person", "default-cluster-id": 35, "cluster-ids": [35]
      },
      {"name": "Person2", "default-cluster-id": 22, "cluster-ids": [22],
        "properties": [
          {"name": "age", "type": "INTEGER", "mandatory": false, "not-null": false},
          {"name": "firstName", "type": "STRING", "mandatory": false, "not-null": false},
          {"name": "lastName", "type": "STRING", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "Profile", "default-cluster-id": 11, "cluster-ids": [11],
        "properties": [
          {"name": "hash", "type": "LONG", "mandatory": false, "not-null": false},
          {"name": "lastAccessOn", "type": "DATETIME", "mandatory": false, "not-null": false, "min": "2010-01-01 00:00:00"},
          {"name": "name", "type": "STRING", "mandatory": false, "not-null": false, "min": "3", "max": "30"},
          {"name": "nick", "type": "STRING", "mandatory": false, "not-null": false, "min": "3", "max": "30"},
          {"name": "photo", "type": "TRANSIENT", "mandatory": false, "not-null": false},
          {"name": "registeredOn", "type": "DATETIME", "mandatory": false, "not-null": false, "min": "2010-01-01 00:00:00"},
          {"name": "surname", "type": "STRING", "mandatory": false, "not-null": false, "min": "3", "max": "30"}
        ]
      },
      {"name": "PropertyIndexTestClass", "default-cluster-id": 21, "cluster-ids": [21],
        "properties": [
          {"name": "prop1", "type": "STRING", "mandatory": false, "not-null": false},
          {"name": "prop2", "type": "INTEGER", "mandatory": false, "not-null": false},
          {"name": "prop3", "type": "BOOLEAN", "mandatory": false, "not-null": false},
          {"name": "prop4", "type": "INTEGER", "mandatory": false, "not-null": false},
          {"name": "prop5", "type": "STRING", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "SQLDropIndexTestClass", "default-cluster-id": 23, "cluster-ids": [23],
        "properties": [
          {"name": "prop1", "type": "DOUBLE", "mandatory": false, "not-null": false},
          {"name": "prop2", "type": "INTEGER", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "SQLSelectCompositeIndexDirectSearchTestClass", "default-cluster-id": 24, "cluster-ids": [24],
        "properties": [
          {"name": "prop1", "type": "INTEGER", "mandatory": false, "not-null": false},
          {"name": "prop2", "type": "INTEGER", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "TestClass", "default-cluster-id": 19, "cluster-ids": [19],
        "properties": [
          {"name": "name", "type": "STRING", "mandatory": false, "not-null": false},
          {"name": "testLink", "type": "LINK", "mandatory": false, "not-null": false, "linked-class": "TestLinkClass"}
        ]
      },
      {"name": "TestLinkClass", "default-cluster-id": 20, "cluster-ids": [20],
        "properties": [
          {"name": "testBoolean", "type": "BOOLEAN", "mandatory": false, "not-null": false},
          {"name": "testString", "type": "STRING", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "Whiz", "default-cluster-id": 12, "cluster-ids": [12],
        "properties": [
          {"name": "account", "type": "LINK", "mandatory": false, "not-null": false, "linked-class": "Account"},
          {"name": "date", "type": "DATE", "mandatory": false, "not-null": false, "min": "2010-01-01"},
          {"name": "id", "type": "INTEGER", "mandatory": false, "not-null": false},
          {"name": "replyTo", "type": "LINK", "mandatory": false, "not-null": false, "linked-class": "Account"},
          {"name": "text", "type": "STRING", "mandatory": true, "not-null": false, "min": "1", "max": "140"}
        ]
      },
      {"name": "classclassIndexManagerTestClassTwo", "default-cluster-id": 25, "cluster-ids": [25]
      },
      {"name": "newV", "default-cluster-id": 31, "cluster-ids": [31], "super-class": "OGraphVertex",
        "properties": [
          {"name": "f_int", "type": "INTEGER", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "vertexA", "default-cluster-id": 32, "cluster-ids": [32], "super-class": "OGraphVertex",
        "properties": [
          {"name": "name", "type": "STRING", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "vertexB", "default-cluster-id": 34, "cluster-ids": [34], "super-class": "OGraphVertex",
        "properties": [
          {"name": "map", "type": "EMBEDDEDMAP", "mandatory": false, "not-null": false},
          {"name": "name", "type": "STRING", "mandatory": false, "not-null": false}
        ]
      }
    ]
  },
  "records": [{
          "@type": "d", "@rid": "#12:476", "@version": 0, "@class": "Whiz",
          "id": 476,
          "date": "2011-12-09 00:00:00:000",
          "text": "Los a went chip, of was returning cover, In the",
          "@fieldTypes": "date=t"
        },{
          "@type": "d", "@rid": "#12:477", "@version": 0, "@class": "Whiz",
          "id": 477,
          "date": "2011-12-09 00:00:00:000",
          "text": "He in office return He inside electronics for $500,000 Jay",
          "@fieldTypes": "date=t"
        }
  ]
}
```

Import from RDBMS

NOTE: As of OrientDB 2.0, you can use the OrientDB-ETL module to import data from an RDBMS. You can use ETL also with 1.7.x by installing it as a separate module.

OrientDB supports a subset of SQL, so importing a database created as "Relational" is straightforward. For the sake of simplicity, consider your Relational database having just these two tables:

- POST
- COMMENT

Where the relationship is between Post and comment as One-2-Many.

```
TABLE POST:
+----+----------------+
| id | title          |
+----+----------------+
| 10 | NoSQL movement |
| 20 | New OrientDB   |
+----+----------------+

TABLE COMMENT:
+----+--------+--------------+
| id | postId | text         |
+----+--------+--------------+
|  0 |   10   | First        |
|  1 |   10   | Second       |
| 21 |   10   | Another      |
| 41 |   20   | First again  |
| 82 |   20   | Second Again |
+----+--------+--------------+
```

- Import using the Document Model (relationships as links)
- Import using the Graph Model (relationships as edges)

Import from a Relational Database

Relational databases typically query and manipulate data with SQL. Given that OrientDB supports a subset of SQL, it is relatively straightfoward to import data from a Relational databases to OrientDB. You can manage imports using the Java API, OrientDB Studio or the OrientDB Console. The examples below use the Console.

> This guide covers importing into the Document Model. Beginning with version 2.0, you can import into the Graph Model using the ETL Module. From version 1.7.x you can still use ETL by installing it as a separate module

For these examples, assume that your Relational database, (referred to as `reldb` in the code), contains two tables: `Post` and `Comment`. The relationship between these tables is one-to-many.

```
reldb> SELECT * FROM post;

+----+----------------+
| id | title          |
+----+----------------+
| 10 | NoSQL movement |
| 20 | New OrientDB   |
+----+----------------+


reldb> SELECT * FROM comment;

+----+--------+--------------+
| id | postId | text         |
+----+--------+--------------+
|  0 |   10   | First        |
|  1 |   10   | Second       |
| 21 |   10   | Another      |
| 41 |   20   | First again  |
| 82 |   20   | Second Again |
+----+--------+--------------+
```

Given that the Relational Model doesn't use concepts from Object Oriented Programming, there are some things to consider in the transition from a Relational database to OrientDB.

- In Relational databases there is no concept of class, so in the import to OrientDB you need to create on class per table.
- In Relational databases, one-to-many references invert from the target table to the source table.

  ```
  Table POST    <- (foreign key) Table COMMENT
  ```

  In OrientDB, it follows the Object Oriented Model, so you have a collection of links connecting instances of `Post` and `Comment`.

  ```
  Class POST ->* (collection of links) Class COMMENT
  ```

Exporting Relational Databases

Most Relational database management systems provide a way to export the database into SQL format. What you specifically need from this is a text file that contains the SQL `INSERT` commands to recreate the database from scratch. For example,

- MySQL: the `mysqldump` utility.
- Oracle Database: the Datapump utilities.
- Microsoft SQL Server: the Import and Export Wizard.

When you run this utility on the example database, it produces an `.sql` file that contains the exported SQL of the Relational database.

```
DROP TABLE IF EXISTS post;
CREATE TABLE post (
id INT(11) NOT NULL AUTO_INCREMENT,
title VARCHAR(128),
PRIMARY KEY (id)
);

DROP TABLE IF EXISTS comment;
CREATE TABLE comment (
id INT(11) NOT NULL AUTO_INCREMENT,
postId INT(11),
text TEXT,
PRIMARY KEY (id),
CONSTRAINT `fk_comments`
    FOREIGN KEY (`postId` )
    REFERENCES `post` (`id` )
);

INSERT INTO POST (id, title) VALUES( 10, 'NoSQL movement' );
INSERT INTO POST (id, title) VALUES( 20, 'New OrientDB' );

INSERT INTO COMMENT (id, postId, text) VALUES( 0, 10, 'First' );
INSERT INTO COMMENT (id, postId, text) VALUES( 1, 10, 'Second' );
INSERT INTO COMMENT (id, postId, text) VALUES( 21, 10, 'Another' );
INSERT INTO COMMENT (id, postId, text) VALUES( 41, 20, 'First again' );
INSERT INTO COMMENT (id, postId, text) VALUES( 82, 20, 'Second Again' );
```

Modifying the Export File

Importing from the Relational database requires that you modify the SQL file to make it usable by OrientDB. In order to do this, you need to open the SQL file, (called `export.sql` below), in a text editor and modify the commands there. Once this is done, you can execute the file on the Console using batch mode.

Database

In order to import a data into OrientDB, you need to have a database ready to receive the import. Note that the example `export.sql` file doesn't include statements to create the database. You can either create a new database or use an existing one.

Using New Databases

In creating a database for the import, you can either create a volatile in-memory database, (one that is only available while OrientDB is running), or you can create a persistent disk-based database. For a persistent database, you can create it on a remote server or locally through the PLocal mode.

The recommended method is PLocal, given that it offers better performance on massive inserts.

- Using the embedded Plocal mode:

  ```
  $ vim export.sql

  CREATE DATABASE PLOCAL:/tmp/db/blog admin_user admin_passwd PLOCAL DOCUMENT
  ```

  Here, the `CREATE DATABASE` command creates a new database at `/tmp/db/blog`.

- Using the Remote mode:

  ```
  $ vim export.sql

  CREATE DATABASE REMOTE:localhost/blog root_user dkdf383dhdsj PLOCAL DOCUMENT
  ```

  This creates a database at the URL `http://localhost/blog`.

> **NOTE**: When you create remote databases, you need the server credentials to access it. The user `root` and its password are stored in the `$ORIENTDB_HOME/config/orientdb-server-config.xml` configuration file.

Using Existing Databases

In the event that you already have a database set up and ready for the import, instead of creating a database add a line that connects to that databases, using the `CONNECT` command.

- Using the embedded PLocal mode:

  ```
  $ vim export.sh

  CONNECT PLOCAL:/tmp/db/blog admin_user admin_passwd
  ```

  This connects to the database at `/tmp/db/blog`.

- Using the Remote mode:

  ```
  $ vim export.sql

  CONNECT REMOTE:localhost/blog admin_user admin_passwd
  ```

  This connects to the database at the URL `http://localhost/blog`.

Declaring Intent

In the SQL file, after you create or connect to the database, you need to declare your intention to perform a massive insert. Intents allow you to utilize automatic tuning OrientDB for maximum performance on particular operations, such as large inserts or reads.

```
$ vim export.sh
...
DECLARE INTENT MASSIVEINSERT
```

Creating Classes

Relational databases have no parallel to concepts in Object Oriented programming, such as classes. Conversely, OrientDB doesn't have a concept of tables in the Relational sense.

Modify the SQL file, changing `CREATE TABLE` statements to `CREATE CLASS` commands:

```
$ vim export.sql
...
CREATE CLASS Post
CREATE CLASS Comment
```

> **NOTE**: In cases where your Relational database was created using Object Relational Mapping, or ORM, tools, such as Hibernate or Data Nucleus, you have to rebuild the original Object Oriented Structure directly in OrientDB.

Create Links

In the Relational database, the relationship between the `post` and `comment` was handled through foreign keys on the `id` fields. OrientDB handles relationships differently, using links between two or more records of the Document type.

By default, the `CREATE LINK` command creates a direct relationship in your object model. Navigation goes from `Post` to `Comment` and not vice versa, which is the case for the Relational database. You'll need to use the `INVERSE` keyword to make the links work in both directions.

Add the following line after the `INSERT` statements.

```
$ vim export.sql
...
CREATE LINK comments TYPE LINKSET FROM comment.postId TO post.id INVERSE
```

Remove Constraints

Unlike how Relational databases handle tables, OrientDB does not require you to create a strict schema on your classes. The properties on each class are defined through the `INSERT` statements. That is, `id` and `title` on `Post` and `id` , `postId` and `text` on `Comment` .

Given that you created a link in the above section, the property `postId` is no longer necessary. Instead of modifying each `INSERT` statement, you can use the `UPDATE` command to remove them at the end:

```
$ vim export.sql
...
UPDATE comment REMOVE postId
```

Bear in mind, this is an optional step. The database will still function if you leave this field in place.

Expected Output

When you've finished, remove any statements that OrientDB does not support. With the changes above this leaves you with a file similar to the one below:

```
$ cat export.sql

CONNECT plocal:/tmp/db/blog admin admin

DECLARE INTENT MASSIVEINSERT

CREATE CLASS Post
CREATE CLASS Comment

INSERT INTO Post (id, title) VALUES( 10, 'NoSQL movement' )
INSERT INTO Post (id, title) VALUES( 20, 'New OrientDB' )

INSERT INTO Comment (id, postId, text) VALUES( 0, 10, 'First' )
INSERT INTO Comment (id, postId, text) VALUES( 1, 10, 'Second' )
INSERT INTO Comment (id, postId, text) VALUES( 21, 10, 'Another' )
INSERT INTO Comment (id, postId, text) VALUES( 41, 20, 'First again' )
INSERT INTO Comment (id, postId, text) VALUES( 82, 20, 'Second Again' )

CREATE LINK comments TYPE LINKSET FROM Comment.postId TO Post.id INVERSE
UPDATE Comment REMOVE postId
```

Importing Databases

When you finish modifying the SQL file, you can execute it through the Console in batch mode. This is done by starting the Console with the SQL file given as the first argument.

```
$ $ORIENTDB_HOME/bin/console.sh export.sql
```

When the OrientDB starts, it executes each of the commands given in the SQL files, creating or connecting to the database, creating the classes and inserting the data from the Relational database. You now have a working instance of OrientDB to use.

Using the Database

You now have an OrientDB Document database where relationships are direct and handled without the use of joins.

- Query for all posts with comments:

  ```
  orientdb> SELECT FROM Post WHERE comments.size() > 0
  ```

- Query for all posts where the comments contain the word "flame" in the `text` property:

  ```
  orientdb> SELECT FROM Post WHERE comments CONTAINS(text
            LIKE '%flame%')
  ```

- Query for all posts with comments made today, assuming that you have added a `date` property to the `Comment` class:

  ```
  orientdb> SELECT FROM Post WHERE comments CONTAINS(date >
            '2011-04-14 00:00:00')
  ```

> For more information, see
>
> - SQL commands
> - Console-Commands

Import from RDBMS to Graph Model

To import from RDBMS to OrientDB using the Graph Model the ETL tool is the suggested way to do it. Take a look at: Import from CSV to a Graph.

Import from Neo4j

Neo4j is an open-source graph database that queries and manipulates data using its own Cypher Query Language.

> For more information on the differences between Neo4j and OrientDB, please refer to the OrientDB vs. Neo4j page.
>
> Neo4j and Cypher are registered trademark of Neo Technology, Inc.

Migration Strategies

Importing data from Neo4j into OrientDB is a straightforward process.

To migrate, please choose one of the following strategies:

1. **Use the *Neo4j to OrientDB Importer***
    - Starting from OrientDB version 2.2, this is the preferred way to migrate from Neo4j, especially for large and complex datasets. The *Neo4j to OrientDB Importer* allows you to migrate Neo4j's nodes, relationships, unique constraints and indexes. For more details, please refer to the Neo4j to OrientDB Importer section
2. **Use GraphML**
    - GraphML is an XML-based file format for graphs. For more details, please refer to the section Import from Neo4j using GraphML

Note: if your data is in CSV format, you can migrate to OrientDB using the OrientDB's ETL tool.

Neo4j to OrientDB Importer

The *Neo4j to OrientDB Importer* is a tool that can help you importing in a quick way a Neo4j graph database into OrientDB.

Imported Neo4j items are:

- nodes
- relationships
- unique constraints
- indexes

> Neo4j and Cypher are registered trademark of Neo Technology, Inc.

Supported Versions

Currently, the *Neo4j to OrientDB Importer* supports, and has been tested with, the following versions:

- OrientDB: 2.2.x, 3.0.x
- Neo4j: 3.x

Limitations

The following limitations apply:

- Currently only `local` migrations are allowed.
- Schema limitations:
  - In case a node in Neo4j has multiple *Labels*, it will be imported into a single OrientDB `Class` (*"MultipleLabelNeo4jConversion"*).
    - Note that the information about the original set of Labels is not lost but stored into an internal property of the imported vertex (*"Neo4jLabelList"*). As a result, it will be possible to query nodes with a specific Neo4j *Label*. Note also that the nodes imported into the single class *"MultipleLabelNeo4jConversion"* can then be moved to other `Classes`, according to your specific needs, using the MOVE VERTEX command. For more information, please refer to this Section.
  - Neo4j Nodes with same *Label* but different case, e.g. *LABEL* and *LAbel* will be aggregated into a single OrientDB vertex `Class`.
  - Neo4j Relationship with same name but different case, e.g. *relaTIONship* and *RELATIONSHIP* will be aggregated into a single OrientDB edge `Class`
  - Migration of Neo4j's *"existence"* constraints (only available in the Neo4j's Enterprise Edition) is currently not implemented.

Installation

The *Neo4j to OrientDB Importer* is provided as an external plugin for the OrientDB Server, and is available as a `zip` or `tar.gz` archive.

Please download the plugin from maven central:

```
http://central.maven.org/maven2/com/orientechnologies/orientdb-neo4j-importer/2.2.37/orientdb-neo4j-importer-2.2.37.tar.gz
```

Replace `tar.gz` with `zip` for the `zip` archive.

To install the plugin, please unpack the archive on your OrientDB server directory (please make sure that the version of your OrientDB server and the version of the Neo4j to OrientDB Importer are the same. Upgrade your OrientDB server, if necessary). On Linux systems, to unpack the archive you can use a command like the following:

```
tar xfv orientdb-neo4j-importer-2.2.37.tar.gz -C path_to_orientDB/ --strip-components=1
```

Migration Scenarios

A typical migration scenario consists of the following steps:

- A copy of the Neo4j's database graph directory (typically `graph.db`) is created into a safe place
- OrientDB is installed
- The *Neo4j to OrientDB Importer* is installed
- The migration process is started from the command line, passing to the *Neo4j to OrientDB Importer* the copy of the Neo4j's database directory created earlier
- OrientDB (embedded or server) is started and the newly imported graph database can be used

Notes:

- Since currently only exclusive, `local`, connections are allowed, during the migration there must be no running servers on the Neo4j's database directory and on the target OrientDB's import directory.

- As an alternative of creating a copy of the Neo4j's database directory, and in case you can schedule a Neo4j shutdown, you can:

  - Shutdown your Neo4j Server
  - Start the migration by passing the original Neo4j's database directory to the *Neo4j to OrientDB Importer* (a good practice is to create a back-up first)

Usage

After Installation, the *Neo4j to OrientDB Importer* can be launched using the provided `orientdb-neo4j-importer.sh` script (or `orientdb-neo4j-importer.bat` for Windows systems).

Syntax

```
OrientDB-Neo4j-Importer
    -neo4jlibdir <neo4jlibdir> (mandatory)
    -neo4jdbdir <neo4jdbdir> (mandatory)
    [-odbdir <odbdir>]
    [-o true|false]
    [-i true|false]
```

Where:

- **neo4jlibdir** (mandatory option) is the full path to the Neo4j lib directory (e.g. `D:\neo4j\neo4j-community-3.0.7\lib`). On Windows systems, this parameter must be the first passed parameter.

- **neo4jdbdir** (mandatory option) is the full path to the Neo4j’s graph database directory (e.g. `D:\neo4j\neo4j-community-3.0.7\data\databases\graph.db`).

- **odbdir** (optional) is the full path to a directory where the Neo4j database will be migrated. The directory will be created by the import tool. In case the directory exists already, the *Neo4j to OrientDB Importer* will behave accordingly to the value of the option `o` (see below). The default value of `odbdir` is `$ORIENTDB_HOME/databases/neo4j_import`.

- **o** (optional). If `true` the `odbdir` directory will be overwritten, if it exists. If `false` and the `odbdir` directory exists, a warning will be printed and the program will exit. The default value of `o` is `false`.

- **i** (optional). If `true` a unique index on the property `Neo4jRelID` will be created, for all migrated edge classes. This allows you to query relationships by original Neo4j relationship Ids. The default value of `i` is `false`.

If the *Neo4j to OrientDB Importer* is launched without parameters, it fails because **-neo4jlibdir** and **-neo4jdbdir** are mandatory.

Example

A typical import command looks like the following (please adapt the value of the `-neo4jlibdir` and `-neo4jdbdir` parameters to your specific case):

Windows:

```
orientdb-neo4j-importer.bat -neo4jlibdir="D:\neo4j\neo4j-community-3.0.7\lib" -neo4jdbdir="D:\neo4j\neo4j-community-3.0.7\data
\databases\graph.db"
```

Linux / Mac:

```
./orientdb-neo4j-importer.sh -neo4jlibdir /mnt/d/neo4j/neo4j-community-3.0.7/lib -neo4jdbdir /mnt/d/neo4j/neo4j-community-3.0.
7/data/databases/graph.db
```

Migration Details

Internally, the *Neo4j to OrientDB Importer* makes use of the Neo4j's `java` API to read the graph database from Neo4j and of the OrientDB's `java` API to store the graph into OrientDB.

The import consists of four phases:

- **Phase 1:** Initialization of the Neo4j and OrientDB servers
- **Phase 2:** Migration of nodes and relationships
- **Phase 3:** Schema migration
- **Phase 4:** Shutdown of the servers and summary info

General Migration Details

The following are some general migration details that is good to keep in mind:

- During the import, OrientDB's `WAL` and `WAL_SYNC_ON_PAGE_FLUSH` are disabled, and OrientDB is prepared for massive inserts (*OIntentMassiveInsert*).

- In case a node in Neo4j has no *Label*, it will be imported in OrientDB into the Class *"GenericClassNeo4jConversion"*.

- Starting from version 2.2.14, in case a node in Neo4j has multiple *Labels*, it will be imported into the `Class` *"MultipleLabelNeo4jConversion"*. Before 2.2.14, only the first *Label* was imported.

- List of original Neo4j *Labels* are stored as properties in the imported OrientDB vertices (property: *"Neo4jLabelList"*).

- During the import, a not unique index is created on the property *"Neo4jLabelList"*. This allows you to query by *Label* even over nodes migrated into the single `Class` *"MultipleLabelNeo4jConversion"*, using queries like: `SELECT FROM V WHERE Neo4jLabelList CONTAINS 'your_label_here'` or the equivalent with the MATCH syntax: `MATCH {class: V, as: your_alias, where: (Neo4jLabelList CONTAINS 'your_label'} RETURN your_alias` .

- Original Neo4j `IDs` are stored as properties in the imported OrientDB vertices and edges (`Neo4jNodeID` for vertices and `Neo4jRelID` for edges). Such properties can be (manually) removed at the end of the import, if not needed.

- During the import, an OrientDB index is created on the property `Neo4jNodeID` for all imported vertex `classes` (node's *Labels* in Neo4j). This is to speed up vertices lookup during edge creation. The created indexes can be (manually) removed at the end of the import, if not needed.

- In case a Neo4j Relationship has the same name of a Neo4j *Label*, e.g. *"RelationshipName"*, the *Neo4j to OrientDB Importer* will import that relationship into OrientDB in the class `E_RelationshipName` (i.e. prefixing the Neo4j's `RelationshipType` with an `E_`).

- During the creation of properties in OrientDB, Neo4j `Char` data type is mapped to a `String` data type.

Details on Schema Migration

The following are some schema-specific migration details that is good to keep in mind:

- If in Neo4j there are no constraints or indexes, and if we exclude the properties and indexes created for internal purposes (`Neo4jNodeID` , `Neo4jRelID` , `Neo4jLabelList` and corresponding indexes), the imported OrientDB database is schemaless.

- If in Neo4j there are constraints or indexes, the imported OrientDB database is schema-hybrid (with some properties defined). In particular, for any constraint and index:

    - The Neo4j property where the constraint or index is defined on, is determined.
    - A corresponding property is created in OrientDB (hence the schema-hybrid mode).
- If a Neo4j unique constraint is found, a corresponding unique index is created in OrientDB
    - In case the creation of the unique index fails, a not unique index will be created. Note: this scenario can happen, by design, when migrating nodes that have multiple *Labels*, as they are imported into a single vertex `Class`).
- If a Neo4j index is found, a corresponding (not unique) OrientDB index is created.

Migration Best Practices

Below some migration best practices.

1. Check if you are using *Labels* with same name but different case, e.g. *LABEL* and *LAbel* and if you really need them. If the correct *Label* is *Label*, change *LABEL* and *LAbel* to *Label* in the original Neo4j database before the import. If you really cannot change them, be aware that with the current version of the Neo4j to OrientDB Importer such nodes will be aggregated into a single OrientDB vertex `Class` .
2. Check if you are using relationships with same name but different case, e.g. *relaTIONship* and *RELATIONSHIP* and if you really need them. If the correct relationship is *Relationship*, change *relaTIONship* and *RELATIONSHIP* to *Relationship* before the import. If you really cannot change them, be aware that with the current version of the Neo4j to OrientDB Importer such relationships will be aggregated into a single OrientDB edge `Class` .
3. Check your constraints and indexes before starting the import. Sometime you have more constraints or indexes than needed, e.g. old ones that you created on *Labels* that you are not using anymore. These constraints will be migrated as well, so a best practice is to check that you have defined, in Neo4j, only those that you really want to import. To check constraints and indexes in Neo4j, you can type `:schema` in the Browser and then click on the "play" icon. Please delete the not needed items.
4. Check if you are using nodes with multiple *Labels*, and if you really need more than one *Label* on them. Be aware that with current version of the Neo4j to OrientDB Importer such nodes with multiple *Labels* will be imported into a single OrientDB `Class` ("*MultipleLabelNeo4jConversion*").

Migration Log

During the migration, a log file is created.

The log can be found at `path_to_orientDB/log/orientdb-neo4j-importer.log` .

Migration Monitoring

During the migration, for each imported Neo4j items (nodes, relationships, constraints and indexes) a completion percentage is written in the shell from where the import has been started, thus allowing to monitor progresses.

For large imports, a best practice is to monitor also the produced import log, using a program like `tail` , e.g.

```
tail -f -n 100 -f path_to_orientDB/log/orientdb-neo4j-importer.log
```

Migration Troubleshooting

In case of problems, the details of the occurred errors are written in the migration log file. Please use this file to troubleshoot the migration.

Connecting to the newly imported Database

After the migration process, you may start an OrientDB server using the `server.sh` or `server.bat` scripts.

You can connect to the newly imported database through Studio or the Console using the OrientDB's default database users, e.g. using the user *admin* and password *admin*.

Please secure your database by removing the default users, if you don't need them, or by creating new users.

For further information on using OrientDB, please refer to the Getting Started Guide.

Query Strategies

This section includes a few strategies that you can use to query your data after the import.

As first thing, please be aware that in OrientDB you can query your data using both SQL or pattern matching. In case you are familiar with Neo4j's Cypher query language, it may be more easy for you to use our pattern matching (see our MATCH syntax for more details). However, keep in mind that depending on your specific use case, our SQL can be of great help.

Counting all nodes

To count all nodes (vertices):

Neo4j's Cypher	OrientDB's SQL
`MATCH (n) RETURN count(n)`	`SELECT COUNT(*) FROM V`

Counting all relationships

To count all relationships (edges):

Neo4j's Cypher	OrientDB's SQL
`MATCH ()-->() RETURN count(*)`	`SELECT COUNT(*) FROM E`

Querying nodes by original Neo4j ID

If you would like to query nodes by their original Neo4j Node ID, you can use the property *Neo4jNodeID*, which is created automatically for you during the import, and indexed as well.

To query a node that belongs to a specific `Class` with name *ClassName*, you can execute a query like:

```
SELECT FROM ClassName WHERE Neo4jNodeID = your_id_here
```

To query a node regardless of the `Class` where it has been included in, you can use a query like:

```
SELECT FROM V WHERE Neo4jNodeID = your_id_here
```

Querying relationships by original Neo4j ID

The strategy to query relationships by their original Neo4j Relationship ID, will be improved in the next hotfix (see GitHub Issue #9, which also includes a workaround).

Querying nodes by original Neo4j Labels

In case the original nodes have just one *Label*, they will be migrated in OrientDB into a `Class` that has name equals to the Neo4j's *Label* name. In this simple case, to query nodes by *Label* you can execute a query like the following:

Neo4j's Cypher	OrientDB's SQL

`MATCH (n:LabelName) RETURN n`	`SELECT FROM LabelName`
	
	or using our MATCH syntax:
	
	`MATCH {class: LabelName, as: n} RETURN n`

More generally speaking, since the original Neo4j *Label* is stored inside the property *Neo4jLabelList*, to query imported nodes (vertices) using their original Neo4j *Label*, you can use queries like the following:

Neo4j's Cypher	OrientDB's SQL

`MATCH (n:LabelName) RETURN n`	`SELECT * FROM V WHERE Neo4jLabelList CONTAINS 'LabelName'`
	
	or using our MATCH syntax:
	
	`MATCH {class: V, as: n, where: (Neo4jLabelList CONTAINS 'LabelName')} RETURN n`

This is, in particular, the strategy that must be followed in case the original Neo4j's nodes have multiple *Labels* (and are hence migrated into the single OrientDB `Class` *MultipleLabelNeo4jConversion*).

Note that the property *Neo4jLabelList* has an index on it.

Migration Example

A complete example of a migration from Neo4j to OrientDB using the *Neo4j to OrientDB Importer* can be found in the section Tutorial: Importing the *northwind* Database from Neo4j.

Roadmap

A list of prioritized enhancements for the Neo4j to OrientDB Importer, along with some other project information can be found here.

FAQ

1. In case original nodes in Neo4j have multiple *Labels*, they are imported into a single OrientDB vertex Class. Depending on the specific use case, after the migration, it may be useful to manually move vertices to other Classes. How can this be done?

First, please note that there is an open enhancement request about having a customized mapping between Neo4j *Labels* and OrientDB `Classes`. Until it is implemented, a possible strategy to quickly move vertices into other `Classes` is to use the `MOVE VERTEX` syntax.

The following are the steps to follow:

A - Create the `Classes` where you want to move the vertices.

When creating the `Classes`, please keep in mind the following:

- Define the following properties:
  - *Neo4jNodeID* of type *LONG*
  - *Neo4jLabelList* of type *EmbeddedList String*

Example:

```
CREATE CLASS YourNewClassHere EXTENDS V
CREATE PROPERTY YourNewClassHere.Neo4jNodeID LONG
CREATE PROPERTY YourNewClassHere.Neo4jLabelList EMBEDDEDLIST STRING
```

B - Select all vertices that have a specific Neo4j *Label*, and then move them to your new `Class`. To do this you can use a query like:

```
MOVE VERTEX (
  SELECT FROM MultipleLabelNeo4jConversion
    WHERE Neo4jLabelList CONTAINS 'Your Neo4j Label here'
  )
TO CLASS:YourNewClassHere BATCH 10000
```

(use a batch size appropriate to your specific case).

C - Create the following indexes in your new `Classes`:

- A unique index on the property *Neo4jNodeID*
- A not unique index on the property *Neo4jLabelList*

Important: creation of the indexes above is crucial in case you will want to query vertices using their original Neo4j node *IDs* or *Labels*.

Example:

```
CREATE INDEX YourNewClassHere.Neo4jNodeID ON YourNewClassHere(Neo4jNodeID) UNIQUE
CREATE INDEX YourNewClassHere.Neo4jLabelList ON YourNewClassHere(Neo4jLabelList) NOTUNIQUE
```

2. Not all constraints have been imported. Why?

By design, there are certain cases where not all the constraints can be imported. It may be that you are in one of these cases. When nodes are aggregated into a single `Class` (either because that node has multiple *Labels* or because there are *Labels* with the same name but different case, e.g. *LABEL* and *LAbel*) not all constraints can be imported: the creation of unique indices in OrientDB will probably fail; as a workaround the Importer will try to create not unique indexes, but when aggregating nodes into a single `Class`, number of created constraints will be probably less than number of constraints in Neo4j, even after the creation of the not unique indexes. This in general may or may not be a problem depending on your specific case. Please feel free to open an issue if you believe you incurred into a bug.

Tutorial: Importing the *northwind* Database from Neo4j

In this tutorial we will use the *Neo4j to OrientDB Importer* to import the Neo4j *northwind* example database into OrientDB.

For general information on the possible Neo4j to OrientDB migration strategies, please refer to the Import from Neo4j section.

> Neo4j and Cypher are registered trademark of Neo Technology, Inc.

Preparing for the migration

Please download and install OrientDB:

```
$ wget https://s3.us-east-2.amazonaws.com/orientdb3/releases/2.2.37/orientdb-community-2.2.37.zip -O orientdb-community-2.2.37
.zip
$ unzip orientdb-community-2.2.37
```

Download and install the *Neo4j to OrientDB Importer*:

```
$ wget http://central.maven.org/maven2/com/orientechnologies/orientdb-neo4j-importer/2.2.37/orientdb-neo4j-importer-2.2.37.tar
.gz
$ tar xfv orientdb-neo4j-importer-2.2.37.tar.gz -C orientdb-community-2.2.37 --strip-components=1
```

For further information on the OrientDB's installation, please refer to this section.

For further information on the *Neo4j to OrientDB Importer* installation, please refer to this section.

Starting the migration

Assuming that:

- `/home/santo/neo4j/neo4j-community-3.0.7/lib` is the full path to the directory that includes the Neo4j's libraries
- `/home/santo/data/graph.db_northwind` is the full path to the directory that contains the Neo4j's *northwind* database
- `/home/santo/orientdb/orientdb-community-2.2.12/databases/northwind_import` is the full path to the directory where you would like to migrate the *northwind* database
- that no Neo4j and OrientDB servers are running on those directories

you can import the *northwind* database with a command similar to the following:

```
./orientdb-neo4j-importer.sh \
  -neo4jlibdir /home/santo/neo4j/neo4j-community-3.0.7/lib \
  -neo4jdbdir /home/santo/neo4j/data/graph.db_northwind \
  -odbdir /home/santo/orientdb/orientdb-community-2.2.12/databases/northwind_import
```

For further information on how to use the *Neo4j to OrientDB Importer*, please refer to this section.

Migration output

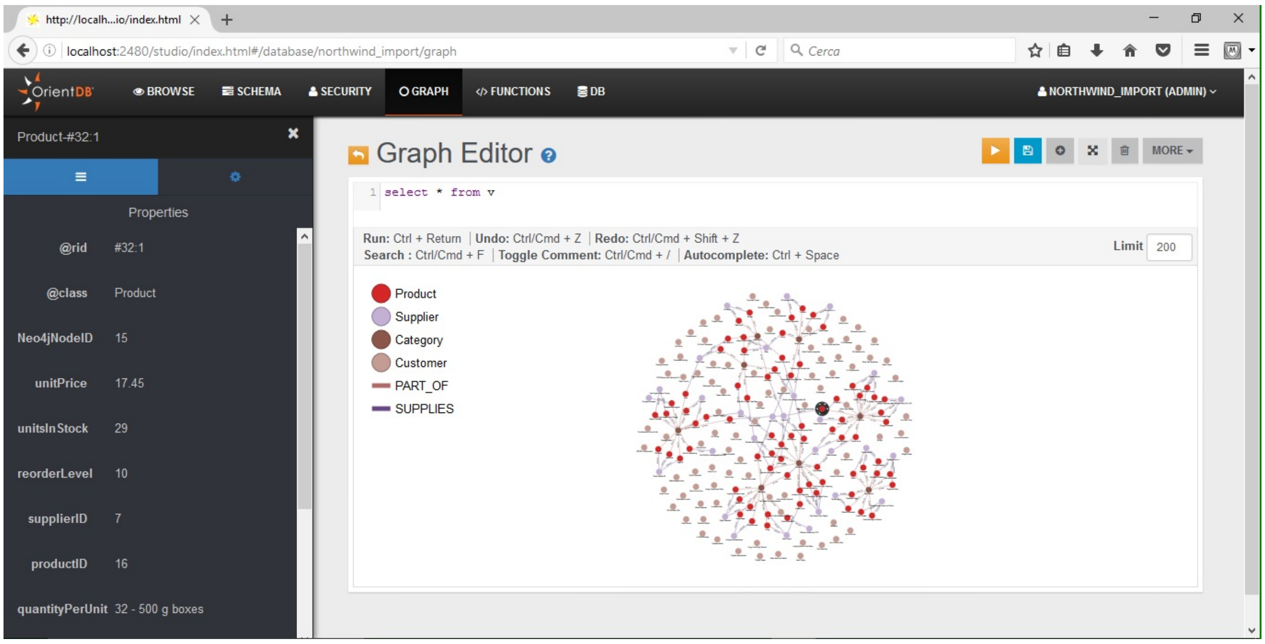
The following is the output that is written by the *Neo4j to OrientDB Importer* during the `northwind` database migration:

```
Neo4j to OrientDB Importer v.2.2.12-SNAPSHOT - Copyrights (c) 2016 OrientDB LTD

WARNING: 'o' option not found. Defaulting to 'false'.

Please make sure that there are no running servers on:
  '/home/santo/neo4j/data/graph.db_northwind' (Neo4j)
and:
  '/home/santo/orientdb/orientdb-community-2.2.12/databases/northwind_import' (OrientDB)

Initializing Neo4j...Done

Initializing OrientDB...Done

Importing Neo4j database:
  '/home/santo/neo4j/data/graph.db_northwind'
into OrientDB database:
  '/home/santo/orientdb/orientdb-community-2.2.12/databases/northwind_import'

Getting all Nodes from Neo4j and creating corresponding Vertices in OrientDB...
  1035 OrientDB Vertices have been created (100% done)
Done

Creating internal Indices on property 'Neo4jNodeID' on all OrientDB Vertices Classes...
  5 OrientDB Indices have been created (100% done)
Done

Getting all Relationships from Neo4j and creating corresponding Edges in OrientDB...
  3139 OrientDB Edges have been created (100% done)
Done

Getting Constraints from Neo4j and creating corresponding ones in OrientDB...
  0 OrientDB Indices have been created
Done

Getting Indices from Neo4j and creating corresponding ones in OrientDB...
  5 OrientDB Indices have been created (100% done)
Done

Import completed!

Shutting down OrientDB...Done
Shutting down Neo4j...Done

===============
Import Summary:
===============

- Found Neo4j Nodes                                                 : 1035
-- With at least one Label                                          :  1035
--- With multiple Labels                                            :   0
-- Without Labels                                                   :  0
- Imported OrientDB Vertices                                        : 1035 (100%)

- Found Neo4j Relationships                                         : 3139
- Imported OrientDB Edges                                           : 3139 (100%)

- Found Neo4j Constraints                                           : 0
- Imported OrientDB Constraints (Indices created)                   : 0

- Found Neo4j (non-constraint) Indices                              : 5
- Imported OrientDB Indices                                         : 5 (100%)

- Additional created Indices (on vertex properties 'Neo4jNodeID')   : 5

- Total Import time:                                                : 29 seconds
-- Initialization time                                              :  7 seconds
-- Time to Import Nodes                                             :  6 seconds (181.67 nodes/sec)
-- Time to Import Relationships                                     :  7 seconds (459.79 rels/sec)
-- Time to Import Constraints and Indices                           :  4 seconds (1.21 indices/sec)
-- Time to create internal Indices (on vertex properties 'Neo4jNodeID')    :  4 seconds (1.22 indices/sec)
```

Connecting to the newly imported Database

General information on how to connect to a newly imported database can be found in this section.

The following is a partial visualization of the *northwind* database done with the Graph Editor included in the OrientDB's Studio tool:


http://localh...io/index.html
localhost:2480/studio/index.html#/database/northwind_import/graph
Cerca
OrientDB
BROWSE
SCHEMA
SECURITY
GRAPH
FUNCTIONS
DB
NORTHWIND_IMPORT (ADMIN)
Graph Editor
MORE
Run: Ctrl + Return | Undo: Ctrl/Cmd + Z | Redo: Ctrl/Cmd + Shift + Z
Search : Ctrl/Cmd + F | Toggle Comment: Ctrl/Cmd + / | Autocomplete: Ctrl + Space
Product
Supplier
Category
Customer
PART_OF
SUPPLIES

Product-#32:1

Properties

@rid	#32:1
@class	Product
Neo4jNodeID	15
unitPrice	17.45
unitsInStock	29
reorderLevel	10
supplierID	7
productID	16
quantityPerUnit	32 - 500 g boxes

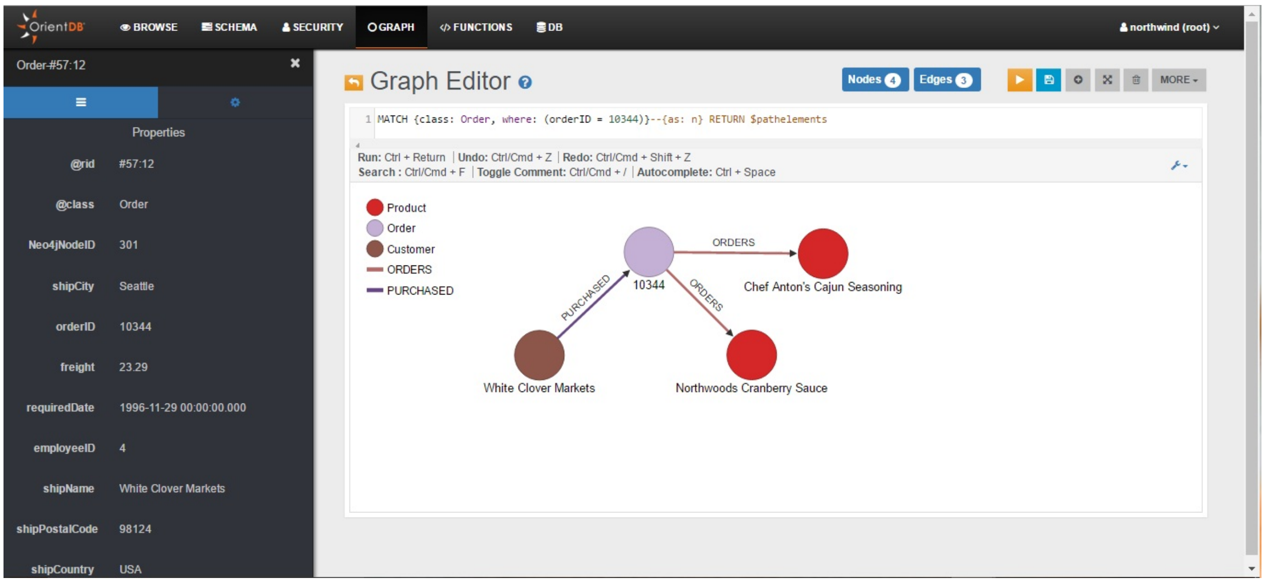
```
select * from v
```

Limit 200

As you can see from the *Limit* field, the visualization is limited to 200 vertices.

The following, instead, is the graph returned by the following MATCH query (the query returns all nodes connected to the Order with `orderID` 10344):

```
MATCH {class: Order, where: (orderID = 10344)}--{as: n} RETURN $pathelements
```


OrientDB
BROWSE
SCHEMA
SECURITY
GRAPH
FUNCTIONS
DB
northwind (root)
Graph Editor
Nodes 4
Edges 3
MORE
Run: Ctrl + Return | Undo: Ctrl/Cmd + Z | Redo: Ctrl/Cmd + Shift + Z
Search : Ctrl/Cmd + F | Toggle Comment: Ctrl/Cmd + / | Autocomplete: Ctrl + Space
Product
Order
Customer
ORDERS
PURCHASED
ORDERS
PURCHASED
ORDERS
10344
Chef Anton's Cajun Seasoning
White Clover Markets
Northwoods Cranberry Sauce

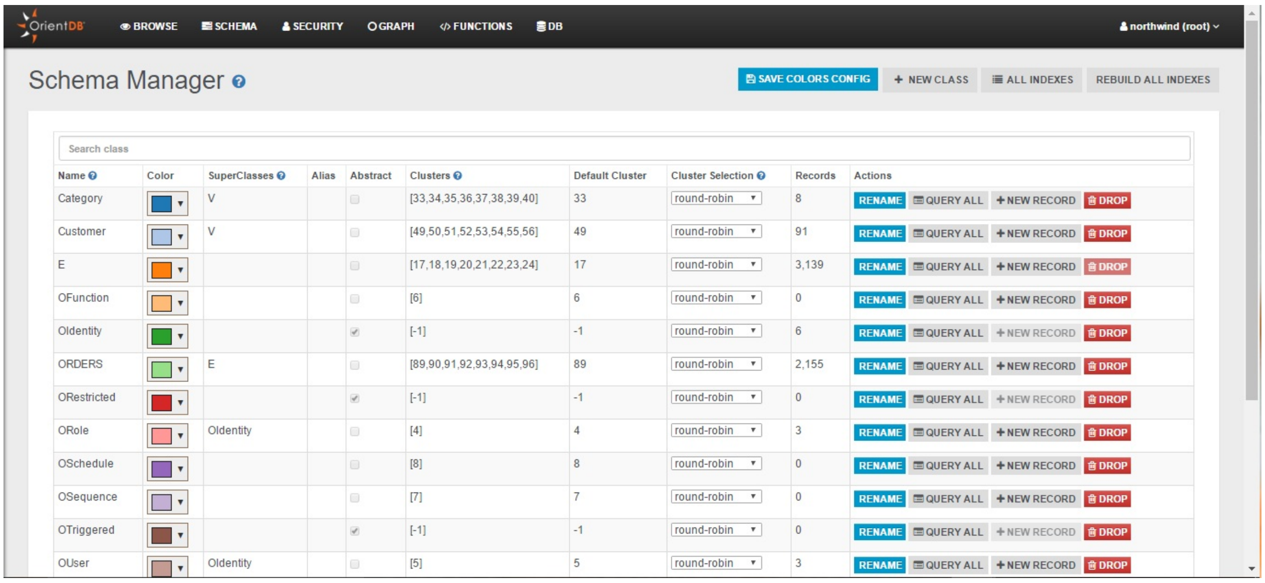
Order-#57:12

Properties

@rid	#57:12
@class	Order
Neo4jNodeID	301
shipCity	Seattle
orderID	10344
freight	23.29
requiredDate	1996-11-29 00:00:00.000
employeeID	4
shipName	White Clover Markets
shipPostalCode	98124
shipCountry	USA

```
MATCH {class: Order, where: (orderID = 10344)}--{as: n} RETURN $pathelements
```

From Studio's Schema Manager, you can check all imported Vertex Classes (node Labels in Neo4j), Edge Classes (Relationship Types in Neo4j), and Indexes:



Schema Manager

Name	Color	SuperClasses	Alias	Abstract	Clusters	Default Cluster	Cluster Selection	Records	Actions
Category	🟦	V		☐	[33,34,35,36,37,38,39,40]	33	round-robin	8	RENAME QUERY ALL NEW RECORD DROP
Customer	🟦	V		☐	[49,50,51,52,53,54,55,56]	49	round-robin	91	RENAME QUERY ALL NEW RECORD DROP
E	🟧			☐	[17,18,19,20,21,22,23,24]	17	round-robin	3,139	RENAME QUERY ALL NEW RECORD DROP
OFunction	🟧			☐	[6]	6	round-robin	0	RENAME QUERY ALL NEW RECORD DROP
OIdentity	🟩			✓	[-1]	-1	round-robin	6	RENAME QUERY ALL NEW RECORD DROP
ORDERS	🟩	E		☐	[89,90,91,92,93,94,95,96]	89	round-robin	2,155	RENAME QUERY ALL NEW RECORD DROP
ORestricted	🟥			✓	[-1]	-1	round-robin	0	RENAME QUERY ALL NEW RECORD DROP
ORole	🟥	OIdentity		☐	[4]	4	round-robin	3	RENAME QUERY ALL NEW RECORD DROP
OSchedule	🟪			☐	[8]	8	round-robin	0	RENAME QUERY ALL NEW RECORD DROP
OSequence	🟪			☐	[7]	7	round-robin	0	RENAME QUERY ALL NEW RECORD DROP
OTriggered	🟫			✓	[-1]	-1	round-robin	0	RENAME QUERY ALL NEW RECORD DROP
OUser	🟫	OIdentity		☐	[5]	5	round-robin	3	RENAME QUERY ALL NEW RECORD DROP

`V` and `E` are special classes: they include all Vertices and all Edges.

Import from Neo4j using GraphML

This section describes the process of importing data from Neo4j to OrientDB using GraphML. For general information on the possible Neo4j to OrientDB migration strategies, please refer to the Import from Neo4j section.

Neo4j can export in GraphML, an XML-based file format for graphs. Given that OrientDB can read GraphML, you can use this file format to import data from Neo4j into OrientDB, using the Console or the Java API.

Note:

> For large and complex datasets, the preferred way to migrate from Neo4j is using the Neo4j to OrientDB Importer.
>
> Neo4j and Cypher are registered trademark of Neo Technology, Inc.

Exporting GraphML

In order to export data from Neo4j into GraphML, you need to install the Neo4j Shell Tools plugin. Once you have this package installed, you can use the `export-graphml` utility to export the database.

1. Change into the Neo4j home directory:

```
$ cd /path/to/neo4j-community-2.3.2
```

2. Download the Neo4j Shell Tools:

```
$ curl http://dist.neo4j.org/jexp/shell/neo4j-shell-tools_2.3.2.zip \
      -o neo4j-shell-tools.zip
```

3. Unzip the `neo4j-shell-tools.zip` file into the `lib` directory:

```
$ unzip neo4j-shell-tools.zip -d lib
```

4. Restart the Neo4j Server. In the event that it's not running, `start` it:

```
$ ./bin/neo4j restart
```

5. Once you have Neo4j restarted with the Neo4j Shell Tools, launch the Neo4j Shell tool, located in the `bin/` directory:

```
$ ./bin/neo4j-shell
Welcome to the Neo4j Shell! Enter 'help' for a list of commands
NOTE: Remote Neo4j graph database service 'shell' at port 1337

neo4j-sh (0)$
```

6. Export the database into GraphML:

```
neo4j-sh (0)$ export-graphml -t -o /tmp/out.graphml
Wrote to GraphML-file /tmp/out.graphml 0. 100%: nodes = 302 rels = 834
properties = 4221 time 59 sec total 59 sec
```

This exports the database to the path `/tmp/out.graphml`.

Importing GraphML

There are three methods available in importing the GraphML file into OrientDB: through the Console, through Gremlin or through the Java API.

Importing through the OrientDB Console

For more recent versions of OrientDB, you can import data from GraphML through the OrientDB Console. If you have version 2.0 or greater, this is the recommended method given that it can automatically translate the Neo4j labels into classes.

1. Log into the OrientDB Console.

   ```
   $ $ORIENTDB_HOME/bin/console.sh
   ```

2. In OrientDB, create a database to receive the import:

   ```
   orientdb> CREATE DATABASE PLOCAL:/tmp/db/test
   Creating database [plocal:/tmp/db/test] using the storage type [plocal]...
   Database created successfully.

   Current database is: plocal:/tmp/db/test
   ```

3. Import the data from the GraphML file:

   ```
   orientdb {db=test}> IMPORT DATABASE /tmp/out.graphml

   Importing GRAPHML database database from /tmp/out.graphml...
   Transaction 8 has been committed in 12ms
   ```

This imports the Neo4j database into OrientDB on the `test` database.

Importing through the Gremlin Console

For older versions of OrientDB, you can import data from GraphML through the Gremlin Console. If you have a version 1.7 or earlier, this is the method to use. It is not recommended on more recent versions, given that it doesn't consider labels declared in Neo4j. In this case, everything imports as the base vertex and edge classes, (that is, `V` and `E`). This means that, after importing through Gremlin you need to refactor you graph elements to fit a more structured schema.

To import the GraphML file into OrientDB, complete the following steps:

1. Launch the Gremlin Console:

   ```
   $ $ORIENTDB_HOME/bin/gremlin.sh

            \,,,/
            (o o)
   -----oOOo-(_)-oOOo-----
   ```

2. From the Gremlin Console, create a new graph, specifying the path to your Graph database, (here `/tmp/db/test`):

   ```
   gremlin> g = new OrientGraph("plocal:/tmp/db/test");
   ==>orientgraph[plocal:/db/test]
   ```

3. Load the GraphML file into the graph object (that is, `g`):

   ```
   gremlin> g.loadGraphML("/tmp/out.graphml");
   ==>null
   ```

4. Exit the Gremlin Console:

```
gremlin> quit
```

This imports the GraphML file into your OrientDB database.

Importing through the Java API

OrientDB Console calls the Java API. Using the Java API directly allows you greater control over the import process. For instance,

```
new OGraphMLReader(new OrientGraph("plocal:/temp/bettergraph")).inputGraph("/temp/neo4j.graphml");
```

This line imports the GraphML file into OrientDB.

Defining Custom Strategies

Beginning in version 2.1, OrientDB allows you to modify the import process through custom strategies for vertex and edge attributes. It supports the following strategies:

- `com.orientechnologies.orient.graph.graphml.OIgnoreGraphMLImportStrategy` Defines attributes to ignore.
- `com.orientechnologies.orient.graph.graphml.ORenameGraphMLImportStrategy` Defines attributes to rename.

Examples

- Ignore the vertex attribute `type` :

```
new OGraphMLReader(new OrientGraph("plocal:/temp/bettergraph")).defineVertexAttributeStrategy("__type__", new OIgnoreGrap
hMLImportStrategy()).inputGraph("/temp/neo4j.graphml");
```

- Ignore the edge attribute `weight` :

```
new OGraphMLReader(new OrientGraph("plocal:/temp/bettergraph")).defineEdgeAttributeStrategy("weight", new OIgnoreGraphMLI
mportStrategy()).inputGraph("/temp/neo4j.graphml");
```

- Rename the vertex attribute `type` in just `type` :

```
new OGraphMLReader(new OrientGraph("plocal:/temp/bettergraph")).defineVertexAttributeStrategy("__type__", new ORenameGrap
hMLImportStrategy("type")).inputGraph("/temp/neo4j.graphml");
```

Import Tips and Tricks

Dealing with Memory Issues

In the event that you experience memory issues while attempting to import from Neo4j, you might consider reducing the batch size. By default, the batch size is set to `1000` . Smaller value causes OrientDB to process the import in smaller units.

- Import with adjusted batch size through the Console:

```
orientdb {db=test}> IMPORT DATABASE /tmp/out.graphml batchSize=100
```

- Import with adjusted batch size through the Java API:

```
new OGraphMLReader(new OrientGraph("plocal:/temp/bettergraph")).setBatchSize(100).inputGraph("/temp/neo4j.graphml");
```

Storing the Vertex ID's

By default, OrientDB updates the import to use its own ID's for vertices. If you want to preserve the original vertex ID's from Neo4j, use the `storeVertexIds` option.

- Import with the original vertex ID's through the Console:

  ```
  orientdb {db=test}> IMPORT DATABASE /tmp/out.graphml storeVertexIds=true
  ```

- Import with the original vertex ID's through the Java API:

  ```
  new OGraphMLReader(new OrientGraph("plocal:/temp/bettergraph")).setStoreVertexIds(true).inputGraph("/temp/neo4j.graphml")
  ;
  ```

Example

A complete example of a migration from Neo4j to OrientDB using the GraphML method can be found in the section Tutorial: Importing the *movie* Database from Neo4j.

Tutorial: Importing the *movie* Database from Neo4j

In this tutorial we will follow the steps described in the Import from Neo4j using GraphML section to import the Neo4j's *movie* example database into OrientDB.

We will also provide some examples of queries using the OrientDB's MATCH syntax, making a comparison with the corresponding Neo4j's Cypher query language.

For general information on the possible Neo4j to OrientDB migration strategies, please refer to the Import from Neo4j section.

> Neo4j and Cypher are registered trademark of Neo Technology, Inc.

Exporting from Neo4j

Assuming you have already downloaded and unpacked the Neo4j Shell Tools, and restarted the Neo4j Server, as described in the Section Exporting GraphML, you can export the *movie* database using `neo4j-shell` with a command like the following one:

```
D:\neo4j\neo4j-community-3.0.6\bin>neo4j-shell.bat

Welcome to the Neo4j Shell! Enter 'help' for a list of commands
NOTE: Remote Neo4j graph database service 'shell' at port 1337

neo4j-sh (?)$ export-graphml -t -o d:/movie.graphml
Wrote to GraphML-file d:/movies.graphml 0. 100%: nodes = 171 rels = 253 properties = 564 time 270 ms total 270 ms
```

In the example above the exported *movie* graph is stored under `D:\movie.graphml` .

Importing into OrientDB

In this tutorial we will import in OrientDB the file `movie.graphml` using the OrientDB's Console. For other GraphML import methods, please refer to the section Importing GraphML.

The OrientDB's Console output generated during the import process is similar to the following (note that first we create a *movie* database using the command `CREATE DATABASE` , and then we do the actual import using the command `IMPORT DATABASE`):

```
D:\orientdb\orientdb-enterprise-2.2.8\bin>console.bat

OrientDB console v.2.2.8-SNAPSHOT (build 2.2.x@r39259e190e16045fe1425b1c0485f8562fca055b; 2016-08-23 14:38:49+0000) www.orient
db.com
Type 'help' to display all the supported commands.
Installing extensions for GREMLIN language v.2.6.0

orientdb> CREATE DATABASE PLOCAL:D:/orientdb/orientdb-enterprise-2.2.8/databases/movie

Creating database [PLOCAL:D:/orientdb/orientdb-enterprise-2.2.8/databases/movie] using the storage type [PLOCAL]...
Database created successfully.

Current database is: PLOCAL:D:/orientdb/orientdb-enterprise-2.2.8/databases/movie
orientdb {db=movie}> IMPORT DATABASE D:/movie.graphml

Importing GRAPHML database from D:/movie.graphml with options ()...
Done: imported 171 vertices and 253 edges
orientdb {db=movie}>
```

As you can see from the output above, as a result of the import 171 vertices and 253 edges have been created in OrientDB. This is exactly the same number of nodes and relationships exported from Neo4j.

For more tips and tricks related to the import process, please refer to this section.

Query Comparison

Once the *movie* database has been imported into OrientDB, you may use several ways to access its data.

The `MATCH` syntax and the tool Studio can be used, for instance, in a similar way to the Neo4j's Cypher and Browser.

The following sections include a comparison of the Neo4j's Cypher and OrientDB's `MATCH` syntax for some queries that you can execute against the *movie* database.

Find the actor named "Tom Hanks"

Neo4j's Cypher:

```
MATCH (tom:Person {name: "Tom Hanks"})
RETURN tom
```

OrientDB's MATCH:

```
MATCH {class: Person, as: tom, where: (name = 'Tom Hanks')}
RETURN $pathElements
```

Find the movie with title "Cloud Atlas"

Neo4j's Cypher:

```
MATCH (cloudAtlas:Movie {title: "Cloud Atlas"})
RETURN cloudAtlas
```

OrientDB's MATCH:

```
MATCH {class: Movie, as: cloudAtlas, where: (title = 'Cloud Atlas')}
RETURN $pathElements
```

Find 10 people

Neo4j's Cypher:

```
MATCH (people:Person)
RETURN people.name
LIMIT 10
```

OrientDB's MATCH:

```
MATCH {class: Person, as: people}
RETURN people.name
LIMIT 10
```

Find the movies released in the 1990s

Neo4j's Cypher:

```
MATCH (nineties:Movie)
WHERE nineties.released > 1990 AND nineties.released < 2000
RETURN nineties.title
```

OrientDB's MATCH:

```
MATCH {class: Movie, as: nineties, WHERE: (released > 1990 AND released < 2000 )}
RETURN nineties.title
```

List all Tom Hanks movies

Neo4j's Cypher:

```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovies)
RETURN tom, tomHanksMovies
```

OrientDB's MATCH:

```
MATCH {class: Person, as: tom, where: (name = 'Tom Hanks')}-ACTED_IN->{as: tomHanksMovies}
RETURN $pathElements
```

Find out who directed "Cloud Atlas"

Neo4j's Cypher:

```
MATCH (cloudAtlas {title: "Cloud Atlas"})<-[:DIRECTED]-(directors)
RETURN directors.name
```

OrientDB's MATCH:

```
MATCH {class: Movie, as: cloudAtlas, where: (title = 'Cloud Atlas')}<-DIRECTED-{as: directors}
RETURN directors.name
```

Find Tom Hanks' co-actors

Neo4j's Cypher:

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors)
RETURN DISTINCT coActors.name
```

OrientDB's MATCH:

```
MATCH {class: Person, as: tom, where: (name = 'Tom Hanks')}-ACTED_IN->{as: m}<-ACTED_IN-{class: Person,as: coActors}
RETURN coActors.name
```

Find how people are related to "Cloud Atlas"

Neo4j's Cypher:

```
MATCH (people:Person)-[relatedTo]-(:Movie {title: "Cloud Atlas"})
RETURN people.name, Type(relatedTo), relatedTo
```

OrientDB's MATCH:

```
MATCH {class: Person, as: people}--{as: m, where: (title = 'Cloud Atlas')}
RETURN $pathElements
```

ETL

The Extractor Transformer and Loader, or ETL, module for OrientDB provides support for moving data to and from OrientDB databases using ETL processes.

- Configuration: The ETL module uses a configuration file, written in JSON.
- Extractor Pulls data from the source database.
- Transformers Convert the data in the pipeline from its source format to one accessible to the target database.
- Loader loads the data into the target database.

How ETL Works

The ETL module receives a backup file from another database, it then converts the fields into an accessible format and loads it into OrientDB.

```
EXTRACTOR => TRANSFORMERS[] => LOADER
```

For example, consider the process for a CSV file. Using the ETL module, OrientDB loads the file, applies whatever changes it needs, then stores the record as a document into the current OrientDB database.

```
+-----------+-----------------------+-----------+
|           |                PIPELINE           |
+ EXTRACTOR +-----------------------+-----------+
|           |     TRANSFORMERS      |  LOADER   |
+-----------+-----------------------+-----------+
|   FILE   ==>  CSV->FIELD->MERGE  ==> OrientDB |
+-----------+-----------------------+-----------+
```

You can modify this pipeline, allowing the transformation and loading phases to run in parallel by setting the configuration variable `"parallel"` to `true`.

```
{"parallel": true}
```

Installation

Beginning with version 2.0, OrientDB bundles the ETL module with the official release.

Usage

To use the ETL module, run the `oetl.sh` script with the configuration file given as an argument.

```
$ $ORIENTDB_HOME/bin/oetl.sh config-dbpedia.json
```

❗	*NOTE: If you are importing data for use in a distributed database, then you must set* `ridBag.embeddedToSbtreeBonsaiThreshold=Integer.MAX\_VALUE` *for the ETL process to avoid replication errors, when the database is updated online.*

Run-time Configuration

When you run the ETL module, you can define its configuration variables by passing it a JSON file, which the ETL module resolves at run-time by passing them as it starts up.

You could also define the values for these variables through command-line options. For example, you could assign the database URL as `${databaseURL}`, then pass the relevant argument through the command-line:

```
$ $ORIENTDB_HOME/bin/oetl.sh config-dbpedia.json \
      -databaseURL=plocal:/tmp/mydb
```

When the ETL module initializes, it pulls `/tmp/mydb` from the command-line to define this variable in the configuration file.

Available Components

- Blocks
- Sources
- Extractors
- Transformers
- Loaders

Examples:

- Import the database of Beers
- Import from CSV to a Graph
- Import from JSON
- Import DBPedia
- Import from a DBMS
- Import from Parse (Facebook)

ETL - Configuration

OrientDB manages configuration for the ETL module through a single JSON configuration file, called at execution.

Syntax

```
{
  "config": {
    <name>: <value>
  },
  "begin": [
    { <block-name>: { <configuration> } }
  ],
  "source" : {
    { <source-name>: { <configuration> } }
  },
  "extractor" : {
    { <extractor-name>: { <configuration> } }
  },
  "transformers" : [
    { <transformer-name>: { <configuration> } }
  ],
  "loader" : { <loader-name>: { <configuration> } },
  "end": [
   { <block-name>: { <configuration> } }
  ]
}
```

- `"config"` Manages all settings and context variables used by any component of the process.
- `"source"` Manages the source data to process.
- `"begin"` Defines a list of blocks to execute in order when the process begins.
- `"extractor"` Manages the extractor settings.
- `"transformers"` Defines a list of transformers to execute in the pipeline.
- `"loader"` Manages the loader settings.
- `"end"` Defines a list of blocks to execute in order when the process finishes.

Example

```
{
  "config": {
    "log": "debug",
    "fileDirectory": "/temp/databases/dbpedia_csv/",
    "fileName": "Person.csv.gz"
  },
  "begin": [
   { "let": { "name": "$filePath",  "value": "$fileDirectory.append( $fileName )"} },
   { "let": { "name": "$className", "value": "$fileName.substring( 0, $fileName.indexOf(".") )"} }
  ],
  "source" : {
    "file": { "path": "$filePath", "lock" : true }
  },
  "extractor" : {
    "row": {}
  },
  "transformers" : [
   { "csv": { "separator": ",", "nullValue": "NULL", "skipFrom": 1, "skipTo": 3 } },
   { "merge": { "joinFieldName":"URI", "lookup":"V.URI" } },
   { "vertex": { "class": "$className"} }
  ],
  "loader" : {
    "orientdb": {
      "dbURL": "plocal:/temp/databases/dbpedia",
      "dbUser": "admin",
      "dbPassword": "admin",
      "dbAutoCreate": true,
      "tx": false,
      "batchCommit": 1000,
      "dbType": "graph",
      "indexes": [{"class":"V", "fields":["URI:string"], "type":"UNIQUE" }]
    }
  }
}
```

General Rules

In developing a configuration file for ETL module processes, consider the following:

- You can use context variables by prefixing them with the `$` sign.
- It assigns the `$input` context variable before each transformation.
- You can execute an expression in OrientDB SQL with the `={<expression>}` syntax. For instance,

```
"field": ={EVAL('3 * 5)}
```

Conditional Execution

In conditional execution, OrientDB only runs executable blocks, such as transformers and blocks, when a condition is found true, such as with a `WHERE` clause.

For example,

```
{ "let": {
    "name": "path",
    "value": "C:/Temp",
    "if": "${os.name} = 'Windows'"
  }
},
{ "let": {
    "name": "path",
    "value": "/tmp",
    "if": "${os.name}.indexOf('nux')"
  }
}
```

Log setting

Most blocks, such transformers and blocks, support the `"log"` setting. Logs take one of the following logging levels, (which are case-insensitive),: `NONE` , `ERROR` , `INFO` , `DEBUG` . By default, it uses the `INFO` level.

Setting the log-level to `DEBUG` displays more information on execution. It also slows down execution, so use it only for development and debugging purposes.

```
{ "http": {
    "url": "http://ip.jsontest.com/",
    "method": "GET",
    "headers": {
      "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.12
5 Safari/537.36"
    },
    "log": "DEBUG"
  }
}
```

Configuration Variables

The ETL module binds all values declared in the `"config"` block to the execution context and are accessible to ETL processing. There are also some special variables used by the ETL process.

Variable	Description	Type	Default value
`"log"`	Defines the global logging level. The accepted levels are: `NONE` , `ERROR` , `INFO` , and `DEBUG` . This parameter is useful to debug a ETL process or single component.	string	`INFO`
`"maxRetries"`	Defines the maximum number of retries allowed, in the event that the loader raises an `ONeedRetryException` , for concurrent modification of the same record.	integer	10
`"parallel"`	Defines whether the ETL module executes pipelines in parallel, using all available cores.	boolean	`false`
`"haltOnError"`	Defines whether the ETL module halts the process when it encounters unmanageable errors. When set to `false` , the process continues in the event of errors. It reports the number of errors it encounters at the end of the import. This feature was introduced in version 2.0.9.	boolean	`true`

Split Configuration on Multiple Files

You can split the configuration into several files allowing for the composition of common parts such as paths, URL's and database references.

For example, you might split the above configuration into two files: one with the input paths for `Person.csv` specifically, while the other would contain common configurations for the ETL module.

```
$ cat personConfig.json

{
  "config": {
    "log": "debug",
    "fileDirectory": "/temp/databases/dbpedia_csv/",
    "fileName": "Person.csv.gz"
  }
}


$ cat commonConfig.json

{
  "begin": [
   { "let": { "name": "$filePath",  "value": "$fileDirectory.append( $fileName )"} },
   { "let": { "name": "$className", "value": "$fileName.substring( 0, $fileName.indexOf(".") )"} }
  ],
  "source" : {
    "file": { "path": "$filePath", "lock" : true }
  },
  "extractor" : {
    "row": {}
  },
  "transformers" : [
   { "csv": { "separator": ",", "nullValue": "NULL", "skipFrom": 1, "skipTo": 3 } },
   { "merge": { "joinFieldName":"URI", "lookup":"V.URI" } },
   { "vertex": { "class": "$className"} }
  ],
  "loader" : {
    "orientdb": {
      "dbURL": "plocal:/temp/databases/dbpedia",
      "dbUser": "admin",
      "dbPassword": "admin",
      "dbAutoCreate": true,
      "tx": false,
      "batchCommit": 1000,
      "dbType": "graph",
      "indexes": [{"class":"V", "fields":["URI:string"], "type":"UNIQUE" }]
    }
  }
}
```

Then, when you can call both configuration files when you run the ETL module:

```
$ $ORIENTDB_HOME/bin/oetl.sh commonConfig.json personConfig.json
```

Run-time configuration

In the configuration file for the ETL module, you can define variables that the module resolves at run-time by passing them as command-line options. Values passed in this manner *override* the values defined in the `"config"` section, even when you use multiple configuration files.

For instance, you might set the configuration variable in the file to `${databaseURL}`, then define it through the command-line using:

```
$ $ORIENTDB_HOME/bin/oetl.sh config-dbpedia.json \
      -databaseURL=plocal:/tmp/mydb
```

In this case, the `databaseURL` parameter is set in the `"config"` section to `/tmp/mydb`, overriding any value given the file.

```
{
  "config": {
    "log": "debug",
    "fileDirectory": "/temp/databases/dbpedia_csv/",
    "fileName": "Person.csv.gz"
    "databaseUrl": "plocal:/temp/currentDb"
  },
  ...
```

ETL - Blocks

When OrientDB executes the ETL module, blocks in the ETL configuration define components to execute in the process. The ETL module in OrientDB supports the following types of blocks:

- `"let"`
- `"code"`
- `"console"`

Let Blocks

In a `"let"` block, you can define variables to the ETL process context.

- Component name: `let`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"name"`	Defines the variable name. The ETL process ignores any values with the `$` prefix.	string	yes	
`"value"`	Defines the fixed value to assign.	an		
`"expression"`	Defines an expression in the OrientDB SQL language to evaluate and assign.	string		

Examples

- Assign a value to the file path variable

```
{
  "let": {
    "name": "$filePath",
    "value": "/temp/myfile"
  }
}
```

- Concat the `$fileName` variable to the `$fileDirectory` to create a new variable for `$filePath` :

```
{
   "let": {
      "name": "$filePath",
      "expression": "$fileDirectory.append($fileName )"
   }
}
```

Code Blocks

In the `"code"` block, you can configure code snippets to execute in any JVM-supported languages. The default language is JavaScript.

- Component name: `code`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"language"`	Defines the programming language to use.	string		Javascript
`"code"`	Defines the code to execute.	string	yes	

Examples

- Execute a `Hello, World!` program in JavaScript, through the ETL module:

```
{
   "code": {
      "language": "Javascript",
      "code": "print('Hello World!');"
   }
}
```

Console Blocks

In a `"console"` block, you can define commands OrientDB executes through the Console.

- Component name: `console`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"file"`	Defines the path to a file containing the commands you want to execute.	string		
`"commands"`	Defines an array of commands, as strings, to execute in sequence.	string array		

Example

- Invoke the console with a file containing the commands:

```
{
   "console": {
      "file": "/temp/commands.sql"
   }
 }
```

- Invoke the console with an array of commands:

```
{
   "console": {
      "commands": [
         "CONNECT plocal:/temp/db/mydb admin admin",
         "INSERT INTO Account set name = 'Luca'"
      ]
  }
}
```

ETL - Sources

When OrientDB executes the ETL module, source components define the source of the data you want to extract. In the case of some extractors like JDBCExtractor work without source, making this component optional. The ETL module in OrientDB supports the following types of sources:

- `"input"`
- `"file"`
- `"http"`

Input Sources

In the input source component, the ETL module extracts data from console input. You may find this useful in cases where the ETL module operates in a pipe with other tools.

- Component name: `input`

Syntax

```
oetl.sh "<input>"
```

Example

- Cat a file, piping its output into the ETL module:

  ```
  $ cat /etc/csv | $ORIENTDB_HOME/bin/oetl.sh \
        "{transformers:[{csv:{}}]}"
  ```

If the source isn't configured, *input* is used by default. It would be easy to write a bash script that loads multiple files from a directory:

```
for f  in $(ls); do echo "processing $f"; cat $f > ORIENTDB_HOME/bin/oetl.sh  ; done
```

File Sources

In the file source component, the variables represent a source file containing the data you want the ETL module to read. You can use text files or files comprssed to `tar.gz` .

- Component name: `file`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"path"`	Defines the path to the file	string	yes	
`"lock"`	Defines whether to lock the file during the extraction phase.	boolean		`false`
`"encoding"`	Defines the encoding for the file.	string		`UTF-8`

Examples

- Extract data from the file at `/tmp/actor.tar.gz` :

```
{
   "file": {
      "path": "/tmp/actor.tar.gz",
      "lock" : true ,
      "encoding" : "UTF-8"
   }
}
```

HTTP Sources

In the HTTP source component, the ETL module extracts data from an HTTP address as source.

- Component name: `http`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"url"`	Defines the URL to look to for source data.	string	yes	
`"method"`	Defines the HTTP method to use in extracting data. Supported methods are: `GET` , `POST` , `PUT` , `DELETE` , `HEAD` , `OPTIONS` , and `TRACE` .	string		`GET`
`"headers"`	Defines the request headers as an inner document key/value.	document		

Examples

- Execute an HTTP request in a `GET` , setting the user agent in the header:

```
{
   "http": {
      "url": "http://ip.jsontest.com/",
      "method": "GET",
      "headers": {
         "User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0
.1985.125 Safari/537.36"
      }
   }
}
```

ETL - Extractors

When OrientDB executes the ETL module, extractor components handle data extraction from source. They are the first part of the ETL process. The ETL module in OrientDB supports the following extractors:

- Row
- CSV
- JDBC
- JSON
- XML

Row Extractor

When the ETL module runs with a Row Extractor, it extracts content row by row. It outputs a string array class.

- Compnent name: `row`
- Output Class: `[ string ]`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"multiLine"`	Defines whether the process supports multiline. Useful with CSV's supporting linefeed inside of string.	boolean		`true`
`"linefeed"`	Defines the linefeed to use in the event of multiline processing.	string		`\r\n`

> The `"multiLine"` and `"linefeed"` parameters were introduced in version 2.0.9.

Examples

- Use the row extractor with its default configuration:

```
{
   "row": {}
}
```

CSV Extractor

When the ETL module runs the CSV Extractor, it parses a file formated to Apache Commons CSV and extracts the data into OrientDB. This component was introduced in version 2.1.4 and is unavailable in older releases of OrientDB.

- Component name: `csv`
- Output class: `[ ODocument ]`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"separator"`	Defines the column separator.	char		`,`
`"columnsOnFirstLine"`	Defines whether the first line contains column descriptors.	boolean		`true`
`"columns"`	Defines array for names and (optionally) types to write.	string array		
`"nullValue"`	Defines the null value in the file.	string		`NULL`
`"dateFormat"`	Defines the format to use in parsing dates from file.	string		`yyyy-MM-dd`
`"dateTimeFormat"`	Defines the format to use in parsing dates with time from file.	string		`yyyy-MM-dd HH:mm`
`"quote"`	Defines string character delimiter.	char		`"`
`"skipFrom"`	Defines the line number you want to skip from.	integer		
`"skipTo"`	Defines the line number you want to skip to.	integer		
`"ignoreEmptyLines"`	Defines whether it should ignore empty lines.	boolean		`false`
`"ignoreMissingColumns"`	Defines whether it should ignore empty columns.	boolean		`false`
`"predefinedFormat"`	Defines the CSV format you want to use.	string		

- For the `"columns"` parameter, specify the type by postfixing it to the value. Specifying types guarantees better performance.
- For the `"predefinedFormat"` parameter, the available formats are: `Default` , `Excel` , `MySQL` , `RFC4180` , `TDF` .

Examples

- Extract lines from CSV to the `ODocument` class, using commas as the separator, considering `NULL` as the null value and skipping rows two through four:

```
{ "csv":
    {  "separator": ",",
       "nullValue": "NULL",
       "skipFrom": 1,
       "skipTo": 3
    }
}
```

- Extract lines from a CSV exported from MySQL:

```
{ "csv":
    {  "predefinedFormat": "MySQL"}
}
```

- Extract lines from a CSV with the default formatting, using `N/A` as the null value and a custom date format:

```
{ "csv":
    {  "predefinedFormat": "Default",
       "nullValue" : "N/A",
       "dateFormat" : "dd-MM-yyyy",
       "dateTimeFormat" : "dd-MM-yyyy HH:mm"
    }
}
```

- Extract lines from a CSV with the default formatting, using `N/A` as the null value, a custom date format, a custom dateTime format and columns type mapping :

```
{ "csv":
    {  "predefinedFormat": "DEFAULT",
       "nullValue" : "N/A",
       "dateFormat" : "dd-MM-yyyy",
       "dateTimeFormat" : "dd-MM-yyyy HH:mm",
       "columns": ["name:string","createdAt:date","updatedAt:dateTime"]
    }
}
```

JDBC Extractor

When the ETL module runs the JDBC Extractor, it can access any database management system that supports the JDBC driver.

In order for the ETL component to connect to the source database, put the source database's JDBC driver in the classpath, or in the `$ORIENTDB_HOME/lib` directory.

- Component name: `jdbc`
- Output class: `[ ODocument ]`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"driver"`	Defines the JDBC Driver class.	string	yes	
`"url"`	Defines the JDBC URL to connect to.	string	yes	
`"userName"`	Defines the username to use on the source database.	string	yes	
`"userPassword"`	Defines the user password to use on the source database.	string	yes	
`"query"`	Defines the query to extract the record you want to import.	string	yes	
`"queryCount"`	Defines query that returns the count of the fetched records, (used to provide a correct progress indicator).	string		

Example

- Extract the contents of the `client` table on the MySQL database `test` at localhost:

```
{ "jdbc": {
    "driver": "com.mysql.jdbc.Driver",
    "url": "jdbc:mysql://localhost/test",
    "userName": "root",
    "userPassword": "my_mysql_passwd",
    "query": "SELECT * FROM client"
  }
}
```

JSON Extractor

When the ETL module runs with a JSON Extractor, it extracts data by parsing JSON objects. If the data has more than one JSON items, you must enclose the in `[]` brackets.

- Component name: `json`
- Output class: `[ ODocument ]`

Example

- Extract data from a JSON file.

```
{ "json": {} }
```

XML Extractor

When the ETL module runs with the XML extractor, it extracts data by parsing XML elements. This feature was introduced in version 2.2.

- Component name: `xml`
- Output class: `[ ODocument ]`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"rootNode"`	Defines the root node to extract in the XML. By default, it builds from the root element in the file.	string		
`"tagsAsAttribute"`	Defines an array of elements, where child elements are considered as attributes of the document and the attribute values as the text within the element.	string array		

Examples

- Extract data from an XML file, where the XML file reads as:

```
<?xml version="1.0" encoding="UTF-8"?>
<a>
  <b>
      <c name='Ferrari' color='red'>ignore</c>
      <c name='Maserati' color='black'/>
  </b>
</a>
```

While the OrientDB-ETL configuration file reads as:

```
{ "source":
  { "file":
    { "path": "src/test/resources/simple.xml" }
  },
  "extractor" :
    { "xml": {} },
    "loader":
      { "test": {} }
}
```

This extracts the data as:

```
{
  "a": {
    "b": {
      "c": [
        {
          "color": "red",
          "name": "Ferrari"
        },
        {
          "color": "black",
          "name": "Maserati"
        }
      ]
    }
  }
}
```

- Extract a collection from XML, where the XML file reads as:

```
<?xml version="1.0" encoding="UTF-8"?>
<CATALOG>
    <CD>
        <TITLE>Empire Burlesque</TITLE>
        <ARTIST>Bob Dylan</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>Columbia</COMPANY>
        <PRICE>10.90</PRICE>
        <YEAR>1985</YEAR>
    </CD>
    <CD>
        <TITLE>Hide your heart</TITLE>
        <ARTIST>Bonnie Tyler</ARTIST>
        <COUNTRY>UK</COUNTRY>
        <COMPANY>CBS Records</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1988</YEAR>
    </CD>
    <CD>
        <TITLE>Greatest Hits</TITLE>
        <ARTIST>Dolly Parton</ARTIST>
        <COUNTRY>USA</COUNTRY>
        <COMPANY>RCA</COMPANY>
        <PRICE>9.90</PRICE>
        <YEAR>1982</YEAR>
    </CD>
</CATALOG>
```

While the OrientDB-ETL configuration file reads:

```
{ "source":
  { "file":
    { "path": "src/test/resources/music.xml" }
  }, "extractor" :
    { "xml":
      { "rootNode": "CATALOG.CD",
        "tagsAsAttribute": ["CATALOG.CD"]
      }
    },
    "loader": { "test": {} }
}
```

This extracts the data as:

```
{
  "TITLE": "Empire Burlesque",
  "ARTIST": "Bob Dylan",
  "COUNTRY": "USA",
  "COMPANY": "Columbia",
  "PRICE": "10.90",
  "YEAR": "1985"
}
{
  "TITLE": "Hide your heart",
  "ARTIST": "Bonnie Tyler",
  "COUNTRY": "UK",
  "COMPANY": "CBS Records",
  "PRICE": "9.90",
  "YEAR": "1988"
}
{
  "TITLE": "Greatest Hits",
  "ARTIST": "Dolly Parton",
  "COUNTRY": "USA",
  "COMPANY": "RCA",
  "PRICE": "9.90",
  "YEAR": "1982"
}
```

ETL Transformers

When OrientDB runs the ETL module, transformer components execute in a pipeline to modify the data before it gets loaded into the OrientDB database. The operate on received input and return output.

Before execution, it always initalizes the `$input` variable, so that if you need to you can access it at run-time.

- CSV
- FIELD
- MERGE
- VERTEX
- CODE
- LINK
- EDGE
- FLOW
- LOG
- BLOCK
- COMMAND

CSV Transformer

> Beginning with version 2.1.4, the CSV Transformer has been deprecated in favor of the CSV Extractor.

Converts a string in a Document, parsing it as CSV

Component description.

- Component name: **csv**
- Supported inputs types: [**String**]
- Output: **ODocument**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"separator"`	Defines the column separator.	char		`,`
`"columnsOnFirstLine"`	Defines whether the first line contains column descriptions.	boolean		`true`
`"columns"`	Defines array containing column names, you can define types by postfixing the names with `:<type>` .	string array		
`"nullValue"`	Defines the value to interpret as null.	string		
`"stringCharacter"`	Defines string character delimiter.	char		`"`
`"skipFrom"`	Defines the line number to skip from.	integer	yes	
`"skipTo"`	Defines the line number to skip to.	integer	yes	

> For the `"columns"` parameter, specifying type guarantees better performance.

Example

- Transform a row in CSV (as `ODocument` class), using commas as the separator, considering `NULL` as a null value and skipping rows two through four.

```
{ "csv": { "separator": ",", "nullValue": "NULL",
           "skipFrom": 1, "skipTo": 3 } }
```

Field Transformer

When the ETL module calls the Field Transformer, it executes an SQL transformer against the field.

Component description.

- Component name: **vertex**
- Supported inputs types: [**ODocument**]
- Output: **ODocument**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"fieldName"`	Defines the document field name to use.	string		
`"expression"`	Defines the expression you want to evaluate, using OrientDB SQL.	string	yes	
`"value"`	Defines the value to set. If the value is taken or computed at run-time, use `"expression"` instead.	any		
`"operation"`	Defines the operation to execute against the fields: `SET` or `REMOVE` .	string		`SET`
`"save"`	Defines whether to save the vertex, edge or document right after setting the fields.	boolean		`false`

> The `"fieldName"` parameter was introduced in version 2.1.

Examples

- Transform the field `class` into the `ODocument` class, by prefixing it with `_` :

```
{ "field":
  { "fieldName": "@class",
    "expression": "class.prefix('_')"
  }
}
```

- Apply the class name, based on the value of another field:

```
{ "field":
  { "fieldName": "@class",
    "expression": "if( ( fileCount >= 0 ), 'D', 'F')"
  }
}
```

- Assign the last part of a path to the `name` field:

```
{ "field":
  { "fieldName": "name",
    "expression": "path.substring( eval( '$current.path.lastIndexOf(\"/\") + 1') )"
  }
}
```

- Asign the field a fixed value:

```
{ "field":
  { "fieldName": "counter",
    "value": 0
  }
}
```

- Rename the field from `salary` to `renumeration` :

```
{ "field":
  { "fieldName": "remuneration",
    "expression": "salary"
  }
},
{ "field":
  { "fieldName": "salary",
    "operation": "remove"
  }
}
```

- Rename multiple fields in one call.

  ```
  { "field":
    { "fieldNames":
      [ "remuneration", "salary" ],
      "operation": "remove"
    }
  }
  ```

  This feature was introduced in version 2.1.

Merge Transformer

When the ETL module calls the Merge Transformer, it takes input from one `ODocument` instance to output into another, loaded by lookup. THe lookup can either be a lookup against an index or a `SELECT` query.

Component description.

- Component name: **merge**
- Supported inputs types: [**ODocument**, **OrientVertex**]
- Output: **ODocument**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"joinFieldName"`	Defines the field containing the join value.	string	yes	
`"lookup"`	Defines the index on which to execute th elookup, or a `SELECT` query.	string	yes	
`"unresolvedLinkAction"`	Defines the action to execute in the event that the join hasn't been resolved.	string		`NOTHING`

For the `"unresolvedLinkAction"` parameter, the supported actions are:

Action	Description
`NOTHING`	Tells the transformer to do nothing.
`WARNING`	Tells the transformer to increment warnings.
`ERROR`	Tells the transformer to increment errors.
`HALT`	Tells the transformer to interrupt the process.
`SKIP`	Tells the transformer to skip the current row.

Example

- Merge the current record against the record returned by the lookup on index `V.URI`, with the value contained in the field `URI` of the input document:

```
{ "merge":
  { "joinFieldName": "URI",
    "lookup":"V.URI"
  }
}
```

Vertex Transformer

When the ETL module runs the Vertex Transformer, it transforms `ODocument` input to output `OrientVertex`.

Component description.

- Component name: **vertex**
- Supported inputs types: [**ODocument**, **OrientVertex**]
- Output: **OrientVertex**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"class"`	Defines the vertex class to use.	string		`V`
`"skipDuplicates"`	Defines whether it skips duplicates. When class has a `UNIQUE` constraint, ETL ignores duplicates.	boolean		`false`

> The `"skipDuplicates"` parameter was introduced in version 2.1.

Example

- Transform `ODocument` input into a vertex, setting the class value to the `$classname` variable:

```
{ "vertex":
  { "class": "$className",
    "skipDuplicates": true
  }
}
```

Edge Transformer

When the ETL modules calls the Edge Transformer, it converts join values in one or more edges between the current vertex and all vertices returned by the lookup. The lookup can either be made against an index or a `SELECT`.

Component description.

- Component name: **EDGE**
- Supported inputs types: [**ODocument**, **OrientVertex**]
- Output: **OrientVertex**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"joinFieldName"`	Defines the field containing the join value.	string	yes	
`"direction"`	Defines the edge direction.	string		`out`
`"class"`	Defines the edge class.	string		`E`
`"lookup"`	Defines the index on which to execute the lookup or a `SELECT` .	string	yes	
`"targetVertexFields"`	Defines the field on which to set the target vertex.	object		
`"edgeFields"`	Defines the fields to set in th eedge.	object		
`"skipDuplicates"`	Defines whether to skip duplicate edges when the `UNIQUE` constraint is set on both the `out` and `in` properties.	boolean		`false`
`"unresolvedLinkAction"`	Defines the action to execute in the event that the join hasn't been resolved.	string		`NOTHING`

> The `"targetVertexFields"` andx `"edgeFields"` parameter were introduced in version 2.1.

For the `"unresolvedLinkAction"` parameter, the following actions are supported:

Action	Description
`NOTHING`	Tells the transformer to do nothing.
`CREATE`	Tells the transformer to create an instance of `OrientVertex` , setting the primary key to the join value.
`WARNING`	Tells the transformer to increment warnings.
`ERROR`	Tells the transformer to increment errors.
`HALT`	Tells the transformer to interrupt the process.
`SKIP`	Tells the transformer to skup the current row.

Examples

- Create an edge from the current vertex, with the class set to `Parent` , to all vertices returned by the lookup on the `D.inode` index with the value contained in the filed `inode_parent` of the input's vertex:

  ```
  { "edge":
    { "class": "Parent",
      "joinFieldName": "inode_parent",
      "lookup":"D.inode",
      "unresolvedLinkAction": "CREATE"
    }
  }
  ```

- Transformer a single-line CSV that contains both vertices and edges:

```
{ "source":
  { "content":
    { "value": "id,name,surname,friendSince,friendId,friendName,friendSurname\n0,Jay,Miner,1996,1,Luca,Garulli"
    }
  },
  "extractor":
    { "row": {} },
  "transformers":
    [
      { "csv": {} },
      { "vertex":
        { "class": "V1" }
      },
      { "edge":
        { "unresolvedLinkAction": "CREATE",
          "class": "Friend",
          "joinFieldName": "friendId",
          "lookup": "V2.fid",
          "targetVertexFields":
            { "name": "${input.friendName}",
               "surname": "${input.friendSurname}"
            },
            "edgeFields":
              { "since": "${input.friendSince}" }
              }
        },
        { "field":
          { "fieldNames":
            [ "friendSince",
              "friendId",
              "friendName",
              "friendSurname"
            ],
            "operation": "remove"
          }
        }
      ],
      "loader":
        { "orientdb":
          { "dbURL": "memory:ETLBaseTest",
            "dbType": "graph",
            "useLightweightEdges": false
          }
      }
  }
```

Flow Transformer

When the ETL module calls the Flow Transformer, it modifies the flow through the pipeline. Supported operations are `skip` and `halt`. Typically, this transformer operates with the `if` attribute.

Component description.

- Component name: **flow**
- Supported inputs types: **Any**
- Output: same type as input

Syntax

Parameter	Description	Type	Mandatory	Default value
`"operation"`	Defines the flow operation: `skip` or `halt`.	string	yes	

Example

- Skip the current record if `name` is null:

```
{ "flow":
  { "if": "name is null",
    "operation" : "skip"
  }
}
```

Code Transformer

When the ETL module calls the Code Transformer, it executes a snippet of code in any JVM supported language. The default is JavaScript. The last object in the code is returned as output.

In the execution context:

- `input` The input object received.
- `record` The record extracted from the input object, when possible. In the event that input object is a vertex or edge, it assigns the underlying `ODocument` to the variable.

Component description.

- Component name: **code**
- Supported inputs types: [**Object**]
- Output: **Object**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"language"`	Defines the programming language to use.	string		JavaScript
`"code"`	Defines the code to execute.	string	yes	

Example

- Display the current record and return the parent:

```
{ "code":
  { "language": "Javascript",
    "code": "print('Current record: ' + record); record.field('parent');"
  }
}
```

Link Transformer

When the ETL module calls the Link Transformer, it converts join values into links within the current record, using the result of the lookup. The lookup can be made against an index or a `SELECT`.

Component description.

- Component name: **link**
- Supported inputs types: [**ODocument**, **OrientVertex**]
- Output: **ODocument**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"joinFieldName"`	Defines the field containing hte join value.	string		
`"joinValue"`	Defines the value to look up.	string		
`"linkFieldName"`	Defines the field containing the link to set.	string	yes	
`"linkFieldType"`	Defines the link type.	string	yes	
`"lookup"`	Defines the index on which to execute the lookup or a `SELECT` query.	string	yes	
`"unresolvedLinkAction"`	Defines the action to execute in the event that the join doesn't resolve.	string		`NOTHING`

For the `"linkFieldType"` parameter, supported link types are: `LINK` , `LINKSET` and `LINKLIST` .

For the `"unresolvedLinkAction"` parameter the following actions are supported:

Action	Description
`NOTHING`	Tells the transformer to do nothing.
`CREATE`	Tells the transformer to create an `ODocument` instance, setting the primary key as the join value.
`WARNING`	Tells the transformer to increment warnings.
`ERROR`	Tells the transformer to increment errors.
`HALT`	Tells the transformer to interrupt the process.
`SKIP`	Tells the transformer to skip the current row.

Example

- Transform a JSON value into a link within the current record, set as `parent` of the type `LINK` , with the result of the lookup on the index `D.node` with the value contained in the field `inode_parent` on the input document.

```
{ "link":
  { "linkFieldName": "parent",
    "linkFieldType": "LINK",
    "joinFieldName": "inode_parent",
    "lookup":"D.inode",
    "unresolvedLinkAction":"CREATE"
  }
}
```

Log Transformer

When the ETL module uses the Log Transformer, it logs the input object to `System.out` .

Component description.

- Component name: **log**
- Supported inputs types: **Any**
- Output: **Any**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"prefix"`	Defines what it writes before the content.	string		
`"postfix"`	Defines what it writes after the content.	string		

Examples

- Log the current value:

```
{ "log": {} }
```

- Log the currnt value with `->` as the prefix:

```
{ "log":
  { "prefix" : "-> " }
}
```

Block Transformer

When the ETL module calls the Block Transformer, it executes an ETL Block component as a transformation step.

Component description.

- Component name: **block**
- Supported inputs types: [**Any**]
- Output: **Any**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"block"`	Defines the block to execute.	document	yes	

Example

- Log the current value:

```
{ "block":
  { "let":
    { "name": "id",
      "value": "={eval('$input.amount * 2')}"
    }
  }
}
```

Command Transformer

When the ETL module calls the Command Transformer, it executes the given command.

Component description.

- Component name: **command**
- Supported inputs types: [**ODocument**]
- Output: **ODocument**

Syntax

Parameter	Description	Type	Mandatory	Default value
`"language"`	Defines the command language: SQL or Gremlin.	string		`sql`
`"command"`	Defines the command to execute.	string	yes	

Example

- Execute a `SELECT` and output an edge:

```
{ "command" :
  { "command" : "SELECT FROM E WHERE id = ${input.edgeid}",
    "output" : "edge"
  }
}
```

ETL - Loaders

When the ETL module executes, Loaders handle the saving of records. They run at the last stage of the process. The ETL module in OrientDB supports the following loaders:

- Output
- OrientDB

Output Loader

When the ETL module runs the Output Loader, it prints the transformer results to the console output. This is the loader that runs by default.

- Component name: **output**
- Accepted input classes: [**Object**]

OrientDB Loader

When the ETL module runs the OrientDB Loader, it loads the records and vertices from the transformers into the OrientDB database.

- Component name: `orientdb`
- Accepted input classes: `[ ODocument, OrientVertex ]`

Syntax

Parameter	Description	Type	Mandatory	Default value
`"dbURL"`	Defines the database URL.	string	yes	
`"dbUser"`	Defines the user name.	string		`admin`
`"dbPassword"`	Defines the user password.	string		`admin`
`"serverUser"`	Defines the server administrator user name, usually `root`	string		
`"serverPassword"`	Defines the server administrator user password that is provided at server startup	string		
`"dbAutoCreate"`	Defines whether it automatically creates the database, in the event that it doesn't exist already.	boolean		`true`
`"dbAutoCreateProperties"`	Defnes whether it automatically creates properties in the schema.	boolean		`false`
`"dbAutoDropIfExists"`	Defines whether it automatically drops the database if it exists already.	boolean		`false`
`"tx"`	Defines whether it uses transactions	boolean		`false`
`"txUseLog"`	Defines whether it uses log in transactions.	boolean		
`"wal"`	Defines whether it uses write ahead logging. Disable to achieve better performance.	boolean		`true`
`"batchCommit"`	When using transactions, defines the batch of entries it commits. Helps avoid having one large transaction in memory.	integer		`0`
`"dbType"`	Defines the database type: `graph` or `document` .	string		`document`
`"class"`	Defines the class to use in storing new record.	string		
`"cluster"`	Defines the cluster in which to store the new record.	string		
`"classes"`	Defines whether it creates classes, if not defined already in the database.	inner document		
`"indexes"`	Defines indexes to use on the ETL process. Before starting, it creates any declared indexes not present in the database. Indexes must have `"type"` , `"class"` and `"fields"` .	inner document		
`"useLightweightEdges"`	Defines whether it changes the default setting for Lightweight Edges.	boolean		`false`
`"standardELementConstraints"`	Defines whether it changes the default setting for TinkerPop BLueprint constraints. Value cannot be null and you cannot use `id` as a property name.	boolean		`true`

For the `"txUseLog"` parameter, when WAL is disabled you can still achieve reliable transactions through this parameter. You may find it useful to group many operations into a batch, such as `CREATE EDGE` . If `"dbAutoCreate"` or `"dbAutoDropIfExists"` are set to true and remote connection is used, `serverUsername` and `serverPassword` must be provided. Usually the server administrator user name is `root` and the password is provided at the startup of the OrientDB server.

Classes

When using the `"classes"` parameter, it defines an inner document that contains additional configuration variables.

Parameter	Description	Type	Mandatory	Default value
`"name"`	Defines the class name.	string	yes	
`"extends"`	Defines the super-class name.	string		
`"clusters"`	Defines the number of cluster to create under the class.	integer		`1`

> **NOTE**: The `"clusters"` parameter was introduced in version 2.1.

Indexes

Parameter	Description	Type	Mandatory	Default value
`"name"`	Defines the index name.	string		
`"class"`	Defines the class name in which to create the index.	string	yes	
`"type"`	Defines the index type.	string	yes	
`"fields"`	Defines an array of fields to index. To specify the field type, use the syntax: `<field>.<type>` .	string	yes	
`"metadata"`	Defines additional index metadata.	string		

Examples

Configuration to load data into the database `dbpedia` on OrientDB, in the directory `/temp/databases` using the PLocal protocol and a Graph database. The load is transactional, performing commits in thousand insert batches. It creates two lookup vertices with indexes against the property string `URI` in the base vertex class `V` . The index is unique.

```
"orientdb": {
      "dbURL": "plocal:/temp/databases/dbpedia",
      "dbUser": "importer",
      "dbPassword": "IMP",
      "dbAutoCreate": true,
      "tx": false,
      "batchCommit": 1000,
      "wal" : false,
      "dbType": "graph",
      "classes": [
        {"name":"Person", "extends": "V" },
        {"name":"Customer", "extends": "Person", "clusters":8 }
      ],
      "indexes": [
        {"class":"V", "fields":["URI:string"], "type":"UNIQUE" },
        {"class":"Person", "fields":["town:string"], "type":"NOTUNIQUE" ,
            metadata : { "ignoreNullValues" : false }
        }
      ]
    }
```

Tutorial: Importing the Open Beer Database into OrientDB

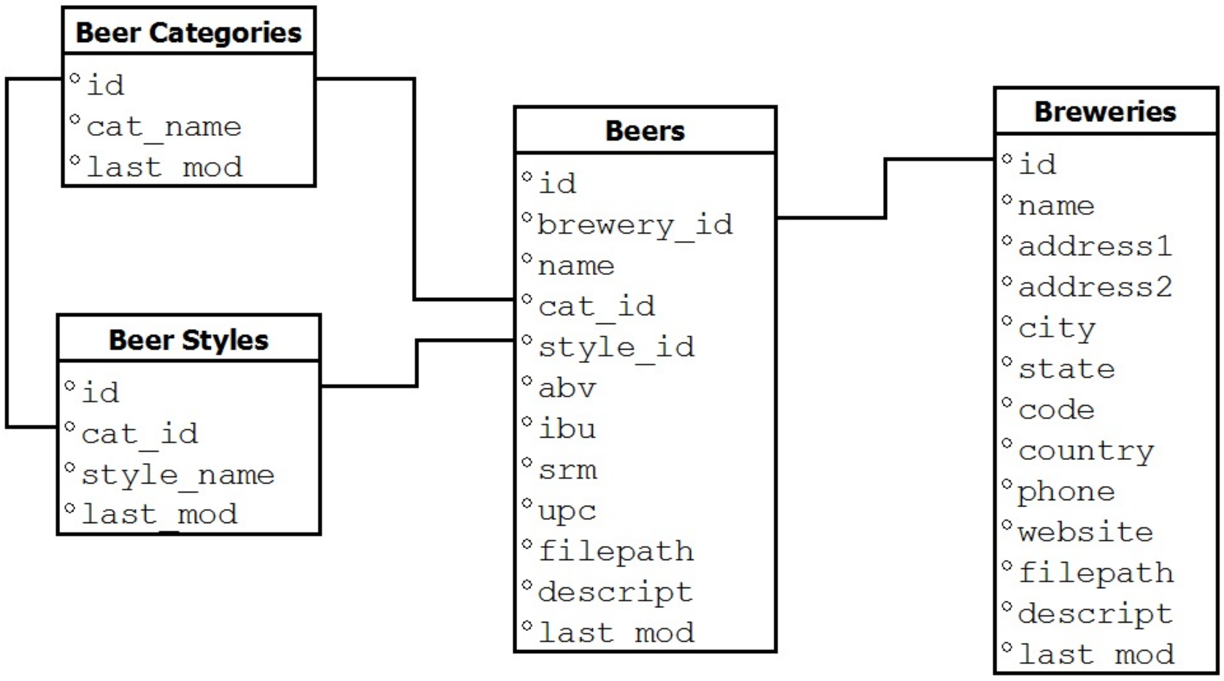
In this tutorial we will use the OrientDB's ETL module to import, as a graph, the Open Beer Database.

Note: You can access directly the converted database, result of this ETL tutorial, in the following ways:

- **Studio**: in the login page press the "Cloud" button, put server's credential and press the download button from the "OpenBeer" row;
- **Direct Download**: download the database from http://orientdb.com/public-databases/OpenBeer.zip and unzip it in a OpenBeer folder inside OrientDB's server "databases" directory.

The Open Beer Database

The Open Beer Database can be downloaded in CSV format from https://openbeerdb.com/. The following image shows its *relational* model:



Preliminary Steps

First, please create a new folder somewhere in your hard drive, and move into it. For this test we will assume `/temp/openbeer` :

```
$ mkdir /temp/openbeer
$ cd /temp/openbeer
```

Download the Open Beer Database in CSV format

Download the Open Beer Database in CSV format and extract the archive:

```
$ curl http://openbeerdb.com/files/openbeerdb_csv.zip > openbeerdb_csv.zip
$ unzip openbeerdb_csv.zip
```

The archive consists of the following files:

- `beers.csv:` contains the beer records
- `breweries.csv:` contains the breweries records
- `breweries_geocode.csv` : contains the geocodes of the breweries. This file is not used in this Tutorial
- `categories.csv` : contains the beer categories
- `styles.csv` : contains the beer styles

Install OrientDB

Download and install OrientDB:

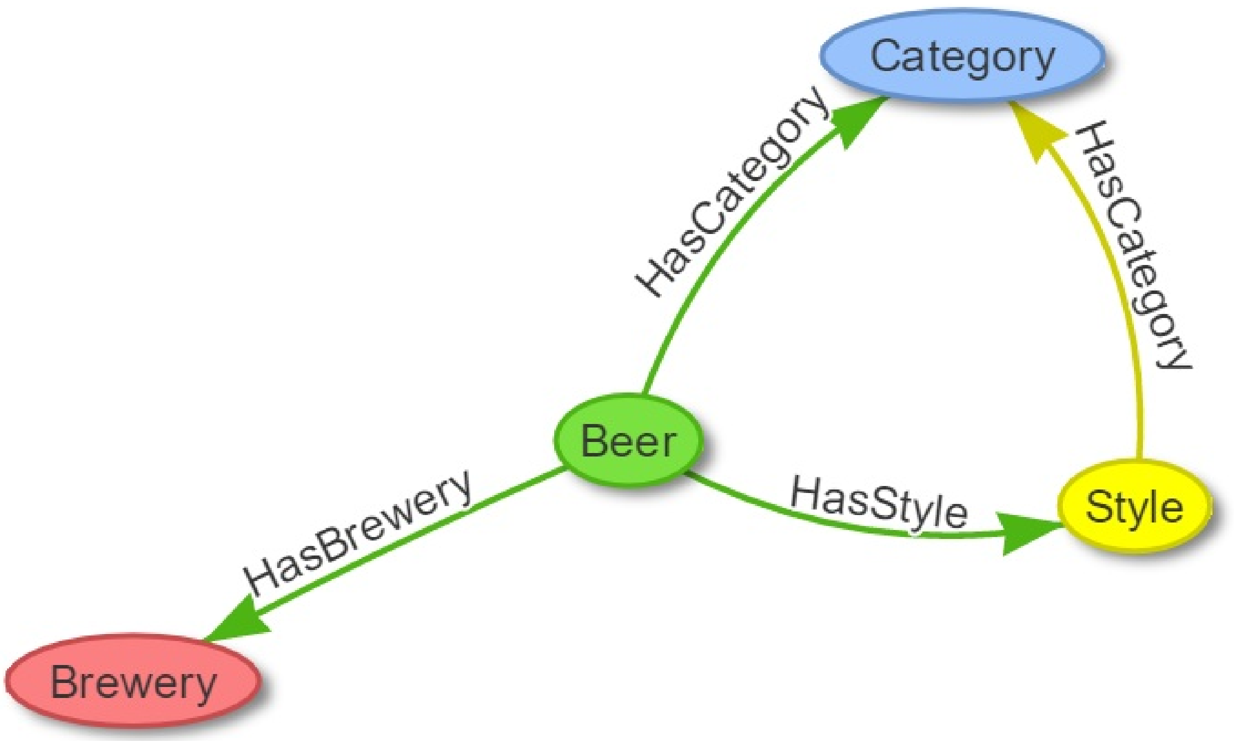
```
$ wget https://s3.us-east-2.amazonaws.com/orientdb3/releases/2.2.37/orientdb-community-2.2.37.zip -O orientdb-community-2.2.37
.zip
$ unzip orientdb-community-2.2.37
```

For more information on how to install OrientDB, please refer to the Installation section.

Graph Data Model

Before starting the ETL process it's important to understand how the Open Beer Database can be modeled as a graph.

The relational model of the Open Beer Database can be easily converted to a *graph* model, as shown below:



The model above consists of the following nodes (or vertices) and relationships (or edges):

- **Nodes**: Beer, Category, Style, Brewery;
- **Relationships**: HasCategory, HasStyle, HasBrewery.

For more informations on the Graph Model in OrientDB, please refer to the Graph Model section.

ETL Process

The ETL module for OrientDB provides support for moving data to and from OrientDB databases using Extract, Transform and Load processes.

The ETL module consists of a script, `oetl.sh`, that takes in input a single JSON configuration file.

For more information on the ETL module, please refer to the ETL section.

Import Beer Categories

The following are the first two lines of the `categories.csv` file:

```
"id","cat_name","last_mod"
"1","British Ale","2010-10-24 13:50:10"
```

In order to import this file in OrientDB, we have to create the following file as `categories.json`:

```
{
  "source": { "file": { "path": "/temp/openbeer/openbeerdb_csv/categories.csv" } },
  "extractor": { "csv": {} },
  "transformers": [
    { "vertex": { "class": "Category" } }
  ],
  "loader": {
    "orientdb": {
       "dbURL": "plocal:../databases/openbeerdb",
       "dbType": "graph",
       "classes": [
         {"name": "Category", "extends": "V"}
       ], "indexes": [
         {"class":"Category", "fields":["id:integer"], "type":"UNIQUE" }
       ]
    }
  }
}
```

To import it into OrientDB, please move into the "bin" directory of the OrientDB distribution:

```
$ cd orientdb-community-2.2.8/bin
```

and run OrientDB ETL:

```
$ ./oetl.sh /temp/openbeer/categories.json

OrientDB etl v.2.0.9 (build @BUILD@) www.orientechnologies.com
BEGIN ETL PROCESSOR
END ETL PROCESSOR
+ extracted 12 rows (0 rows/sec) - 12 rows -> loaded 11 vertices (0 vertices/sec) Total time: 77ms [0 warnings, 0 errors]
```

Import Beer Styles

Now let's import the Beer Styles. These are the first two lines of the `styles.csv` file:

```
"id","cat_id","style_name","last_mod"
"1","1","Classic English-Style Pale Ale","2010-10-24 13:53:31"
```

In this case we will correlate the Style with the Category created earlier.

This is the `styles.json` to use with OrientDB ETL for the next step:

```
{
  "source": { "file": { "path": "/temp/openbeer/openbeerdb_csv/styles.csv" } },
  "extractor": { "csv": {} },
  "transformers": [
    { "vertex": { "class": "Style" } },
    { "edge": { "class": "HasCategory",  "joinFieldName": "cat_id", "lookup": "Category.id" } }
  ],
  "loader": {
    "orientdb": {
       "dbURL": "plocal:../databases/openbeerdb",
       "dbType": "graph",
       "classes": [
         {"name": "Style", "extends": "V"},
         {"name": "HasCategory", "extends": "E"}
       ], "indexes": [
         {"class":"Style", "fields":["id:integer"], "type":"UNIQUE" }
       ]
    }
  }
}
```

Now, to import the styles, please execute the following command:

```
$ ./oetl.sh /temp/openbeer/styles.json

OrientDB etl v.2.0.9 (build @BUILD@) www.orientechnologies.com
BEGIN ETL PROCESSOR
END ETL PROCESSOR
+ extracted 142 rows (0 rows/sec) - 142 rows -> loaded 141 vertices (0 vertices/sec) Total time: 498ms [0 warnings, 0 errors]
```

Import Breweries

Now it's time for the Breweries. These are the first two lines of the `breweries.csv` file:

```
"id","name","address1","address2","city","state","code","country","phone","website","filepath","descript","last_mod"
"1","(512) Brewing Company","407 Radam, F200",,"Austin","Texas","78745","United States","512.707.2337","http://512brewing.com/
",,"(512) Brewing Company is a microbrewery located in the heart of Austin that brews for the community using as many local, d
omestic and organic ingredients as possible.","2010-07-22 20:00:20"
```

Breweries have no outgoing relations with other entities, so this is a plain import similar to the one we did for the categories.

This is the `breweries.json` to use with OrientDB ETL for the next step:

```
{
  "source": { "file": { "path": "/temp/openbeer/openbeerdb_csv/breweries.csv" } },
  "extractor": { "csv": {} },
  "transformers": [
    { "vertex": { "class": "Brewery" } }
  ],
  "loader": {
    "orientdb": {
       "dbURL": "plocal:../databases/openbeerdb",
       "dbType": "graph",
       "classes": [
         {"name": "Brewery", "extends": "V"}
       ], "indexes": [
         {"class":"Brewery", "fields":["id:integer"], "type":"UNIQUE" }
       ]
    }
  }
}
```

Run the import for breweries:

```
$ ./oetl.sh /temp/openbeer/breweries.json

OrientDB etl v.2.0.9 (build @BUILD@) www.orientechnologies.com
BEGIN ETL PROCESSOR
END ETL PROCESSOR
+ extracted 1.395 rows (0 rows/sec) - 1.395 rows -> loaded 1.394 vertices (0 vertices/sec) Total time: 830ms [0 warnings, 0 er
rors]
```

Import Beers

Now it's time for the last and most important file: the Beers! These are the first two lines of the `beers.csv` file:

```
"id","brewery_id","name","cat_id","style_id","abv","ibu","srm","upc","filepath","descript","last_mod",,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,
"1","812","Hocus Pocus","11","116","4.5","0","0","0",,"Our take on a classic summer ale.  A toast to weeds, rays, and summer h
aze.  A light, crisp ale for mowing lawns, hitting lazy fly balls, and communing with nature, Hocus Pocus is offered up as a s
ummer sacrifice to clodless days.
```

As you can see each beer is connected to other entities through the following fields:

- `brewery_id` -> **Brewery**
- `cat_id` -> **Category**
- `style_id` -> **Style**

This is the `beers.json` to use with OrientDB ETL for the next step:

```
{
  "config" : { "haltOnError": false },
  "source": { "file": { "path": "/temp/openbeer/openbeerdb_csv/beers.csv" } },
  "extractor": { "csv": { "columns": ["id","brewery_id","name","cat_id","style_id","abv","ibu","srm","upc","filepath","descrip
t","last_mod"],
                                "columnsOnFirstLine": true } },
  "transformers": [
    { "vertex": { "class": "Beer" } },
    { "edge": { "class": "HasCategory",  "joinFieldName": "cat_id", "lookup": "Category.id" } },
    { "edge": { "class": "HasBrewery",  "joinFieldName": "brewery_id", "lookup": "Brewery.id" } },
    { "edge": { "class": "HasStyle",  "joinFieldName": "style_id", "lookup": "Style.id" } }
  ],
  "loader": {
    "orientdb": {
       "dbURL": "plocal:../databases/openbeerdb",
       "dbType": "graph",
       "classes": [
         {"name": "Beer", "extends": "V"},
         {"name": "HasCategory", "extends": "E"},
         {"name": "HasStyle", "extends": "E"},
         {"name": "HasBrewery", "extends": "E"}
       ], "indexes": [
         {"class":"Beer", "fields":["id:integer"], "type":"UNIQUE" }
       ]
    }
  }
}
```

Run the final import for beers:

```
$ ./oetl.sh /temp/openbeer/beers.json

OrientDB etl v.2.0.9 (build @BUILD@) www.orientechnologies.com
BEGIN ETL PROCESSOR
...
+ extracted 5.862 rows (1.041 rows/sec) - 5.862 rows -> loaded 4.332 vertices (929 vertices/sec) Total time: 10801ms [0 warnin
gs, 27 errors]
END ETL PROCESSOR
```

Note: the 27 errors are due to the 27 wrong content lines that have no id.

Some Queries and Visualizations

Now that the database has been imported we can execute some queries and create some visualizations.

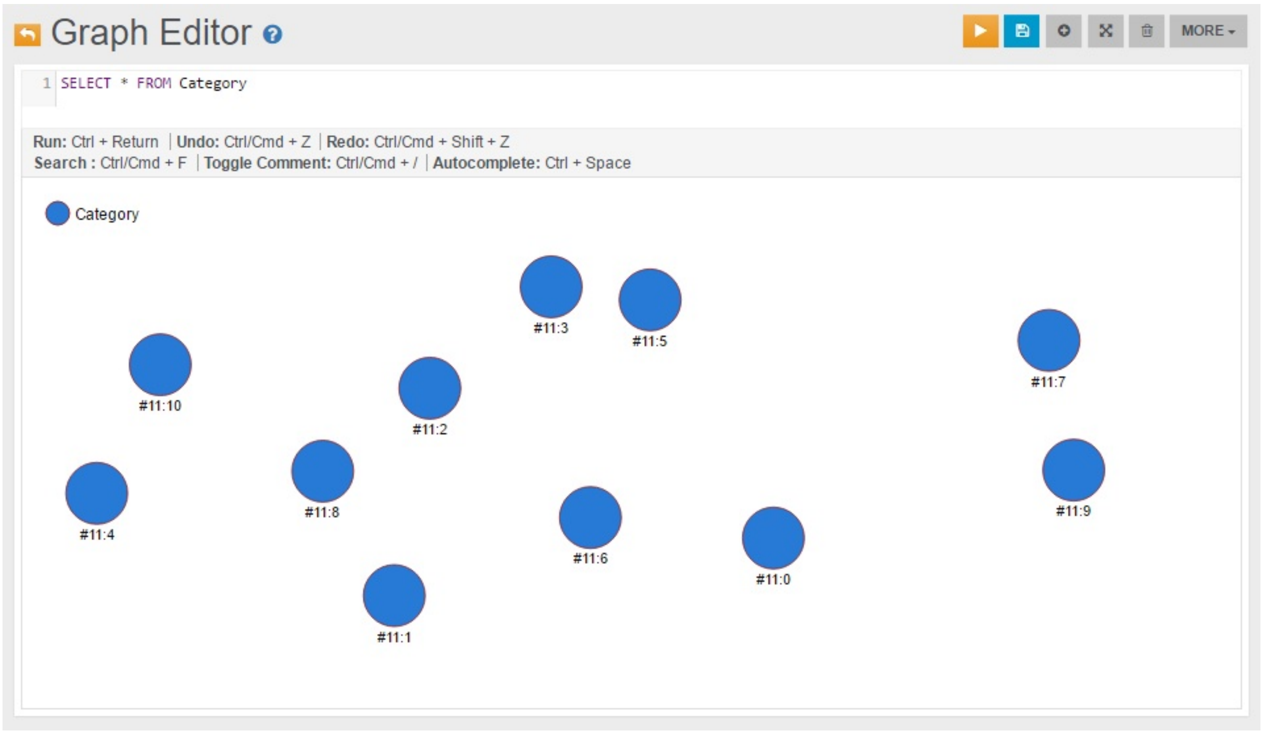
The following are some ways we can use to access the newly imported `OpenBeer` database:

- Console
- Gremlin Console
- Studio
- APIs & Drivers
- some external tools, like Gephy
- some external visualization libraries for graph rendering

If we want to query all *Category* vertices we can execute the following query:

```
SELECT * FROM Category
```

The following is the visualization we can create using the Studio's Graph Editor:



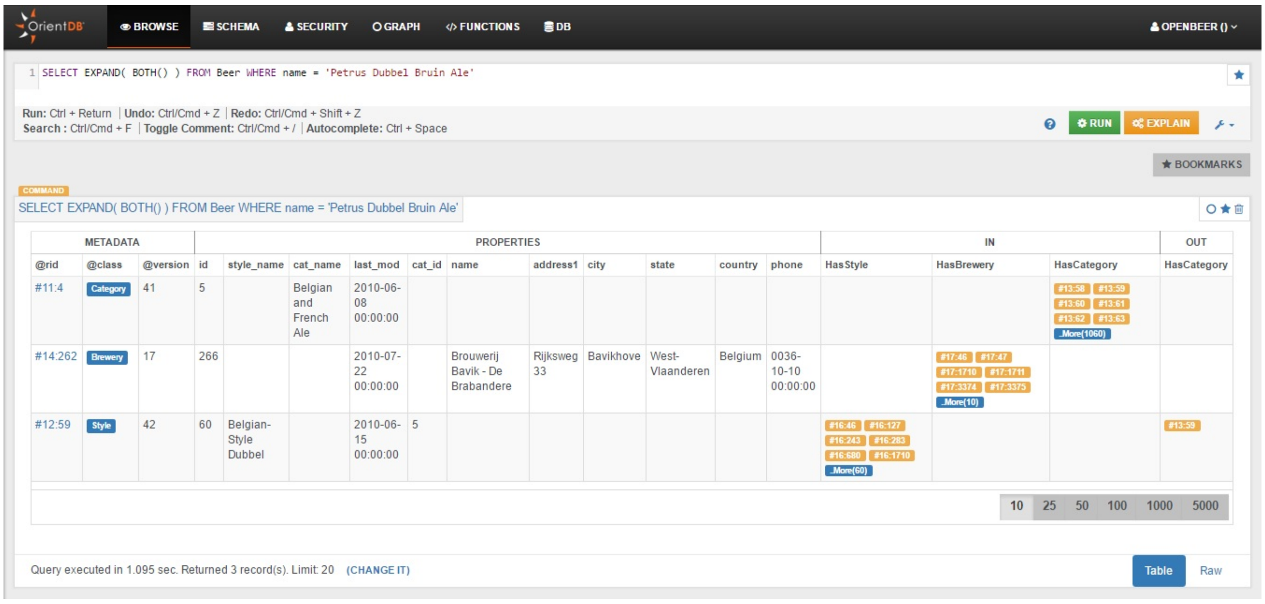
```
SELECT * FROM Category
```

Run: Ctrl + Return | Undo: Ctrl/Cmd + Z | Redo: Ctrl/Cmd + Shift + Z
Search : Ctrl/Cmd + F | Toggle Comment: Ctrl/Cmd + / | Autocomplete: Ctrl + Space

If we want to find all nodes directly connected to a specific beer (e.g. the beer *Petrus Dubbel Bruin Ale*) with either an incoming or outgoing relationship, we can use a query like the following:

```
SELECT EXPAND( BOTH() ) FROM Beer WHERE name = 'Petrus Dubbel Bruin Ale'
```

If we execute this query in the Browse tab of Studio we get the following result, from where we can see that there are three nodes connected to this beer, having *@rid 11:4*, *14:262* and *12:59*:



```
SELECT EXPAND( BOTH() ) FROM Beer WHERE name = 'Petrus Dubbel Bruin Ale'
```

Run: Ctrl + Return | **Undo:** Ctrl/Cmd + Z | **Redo:** Ctrl/Cmd + Shift + Z
Search : Ctrl/Cmd + F | **Toggle Comment:** Ctrl/Cmd + / | **Autocomplete:** Ctrl + Space

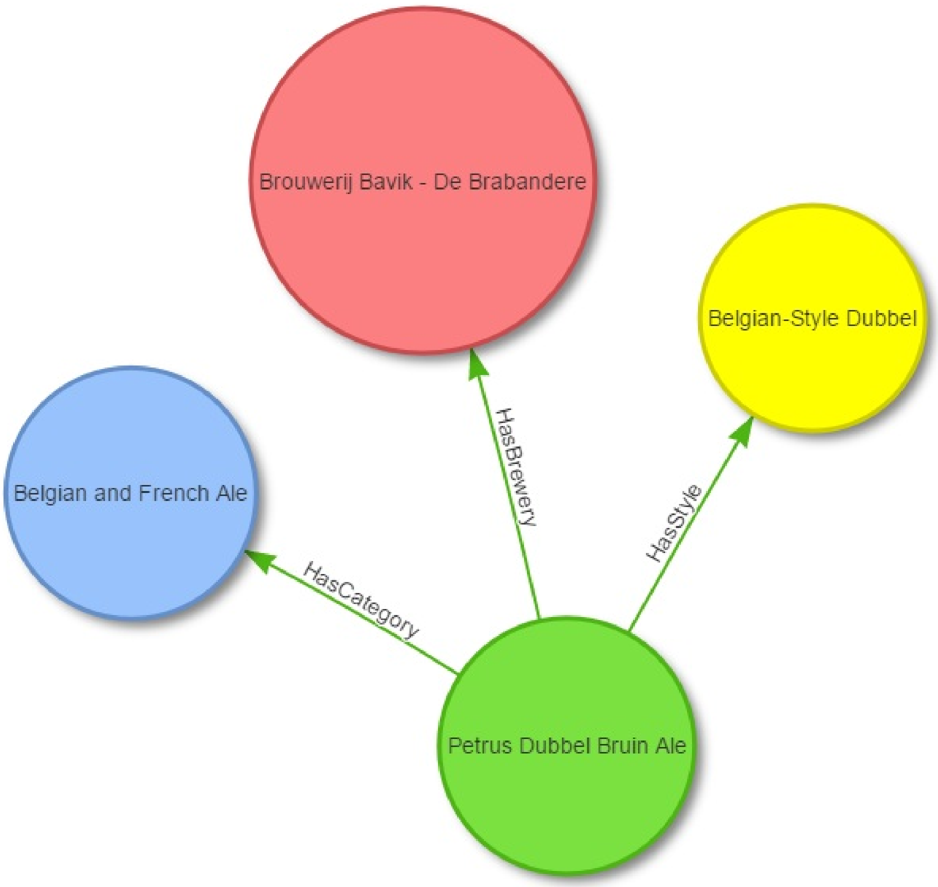
SELECT EXPAND(BOTH()) FROM Beer WHERE name = 'Petrus Dubbel Bruin Ale'

METADATA			PROPERTIES										IN			OUT
@rid	@class	@version	id	style_name	cat_name	last_mod	cat_id	name	address1	city	state	country	phone	HasStyle	HasBrewery	HasCategory	HasCategory
#11:4	Category	41	5		Belgian and French Ale	2010-06-08 00:00:00										#13:58 #13:59 #13:60 #13:61 #13:62 #13:63 ..More(1060)	
#14:262	Brewery	17	266			2010-07-22 00:00:00		Brouwerij Bavik - De Brabandere	Rijksweg 33	Bavikhove	West-Vlaanderen	Belgium	0036-10-10 00:00:00		#17:46 #17:47 #17:1710 #17:1711 #17:3374 #17:3375 ..More(10)		
#12:59	Style	42	60	Belgian-Style Dubbel		2010-06-15 00:00:00	5							#16:46 #16:127 #16:243 #16:283 #16:680 #16:1710 ..More(60)			#13:59

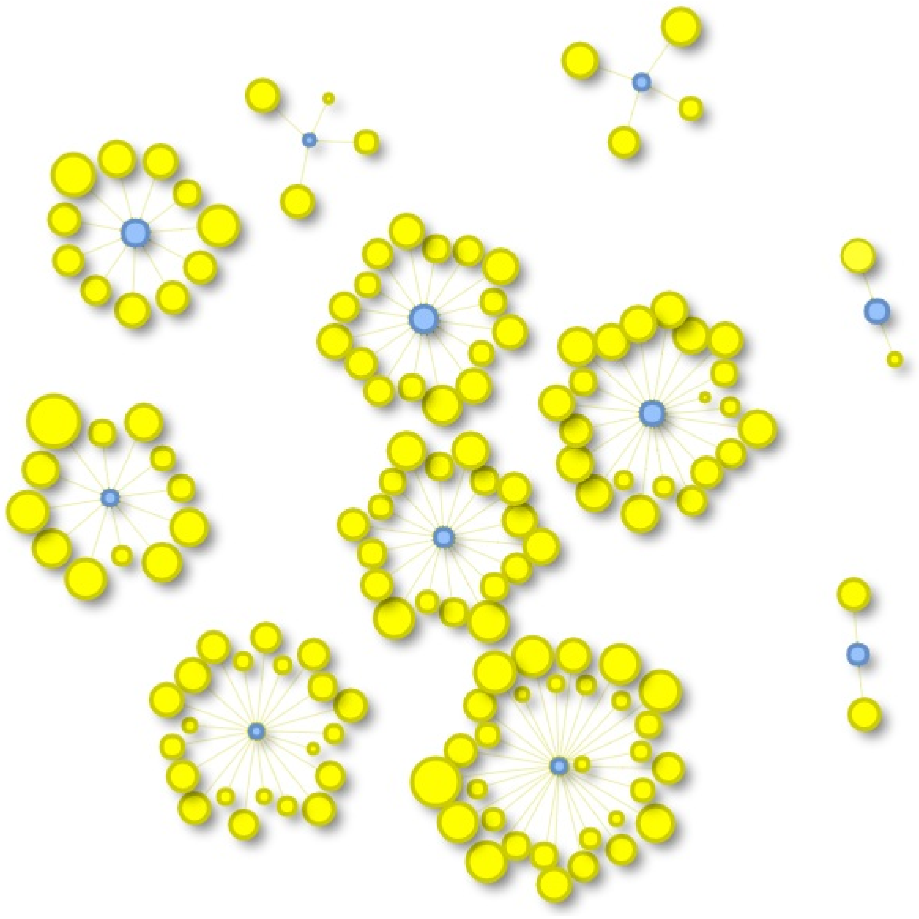
10 25 50 100 1000 5000

Query executed in 1.095 sec. Returned 3 record(s). Limit: 20 (CHANGE IT)

The same result can be visualized using an external graph library. For instance, the following graph has been obtained using the library vis.js where the input *visjs* dataset has been created with a java program created using the OrientDB's Java Graph API:



We can also query bigger portions of the graph. For example, the following image shows all beer *Category* nodes and for each of them all the connected *Style* nodes (the visualization has been created using the library vis.js):



Import from a CSV file to a Graph

This example describes the process for importing from a CSV file into OrientDB as a Graph. For the sake of simplicity, consider only these 2 entities:

- POST
- COMMENT

Also consider the relationship between Post and Comment as One-2-Many. One Post can have multiple Comments. We're representing them as they would appear in an RDBMS, but the source could be anything.

With an RDBMS Post and Comment would be stored in 2 separate tables:

```
TABLE POST:
+----+----------------+
| id | title          |
+----+----------------+
| 10 | NoSQL movement |
| 20 | New OrientDB   |
+----+----------------+

TABLE COMMENT:
+----+--------+--------------+
| id | postId | text         |
+----+--------+--------------+
|  0 |   10   | First        |
|  1 |   10   | Second       |
| 21 |   10   | Another      |
| 41 |   20   | First again  |
| 82 |   20   | Second Again |
+----+--------+--------------+
```

With an RDBMS, one-2-many references are inverted from the target table (Comment) to the source one (Post). This is due to the inability of an RDBMS to handle a collection of values.

In comparison, using the OrientDB Graph model, relationships are modeled as you would think, when you design an application: POSTs have edges to COMMENTs.

So, with an RDBMS you have:

```
Table POST    <- (foreign key) Table COMMENT
```

With OrientDB, the Graph model uses Edges to manage relationships:

```
Class POST ->* (collection of edges) Class COMMENT
```

(1) Export to CSV

If you're using an RDBMS or any other source, export your data in CSV format. The ETL module is also able to extract from JSON and an RDBMS directly through JDBC drivers. However, for the sake of simplicity, in this example we're going to use CSV as the source format.

Consider having 2 CSV files:

File posts.csv

posts.csv file, containing all the posts

```
id,title
10,NoSQL movement
20,New OrientDB
```

File comments.csv

comments.csv file, containing all the comments, with the relationship to the commented post

```
id,postId,text
0,10,First
1,10,Second
21,10,Another
41,20,First again
82,20,Second Again
```

(2) ETL Configuration

The OrientDB ETL tool requires only a JSON file to define the ETL process as Extractor, a list of Transformers to be executed in the pipeline, and a Loader, to load graph elements into the OrientDB database.

Below are 2 files containing the ETL to import Posts and Comments separately.

post.json ETL file

```
{
  "source": { "file": { "path": "/temp/datasets/posts.csv" } },
  "extractor": { "csv": {} },
  "transformers": [
    { "vertex": { "class": "Post" } }
  ],
  "loader": {
    "orientdb": {
       "dbURL": "plocal:/temp/databases/blog",
       "dbType": "graph",
       "classes": [
         {"name": "Post", "extends": "V"},
         {"name": "Comment", "extends": "V"},
         {"name": "HasComments", "extends": "E"}
       ], "indexes": [
         {"class":"Post", "fields":["id:integer"], "type":"UNIQUE" }
       ]
    }
  }
}
```

The Loader contains all the information to connect to an OrientDB database. We have used a plocal database, because it's faster. However, if you have an OrientDB server up & running, use "remote:" instead. Note the classes and indexes declared in the Loader. As soon as the Loader is configured, the classes and indexes are created, if they do not already exist. We have created the index on the Post.id field to assure that there are no duplicates and that the lookup on the created edges (see below) will be fast enough.

comments.json ETL file

```
{
  "source": { "file": { "path": "/temp/datasets/comments.csv" } },
  "extractor": { "csv": {} },
  "transformers": [
    { "vertex": { "class": "Comment" } },
    { "edge": { "class": "HasComments",
                "joinFieldName": "postId",
                "lookup": "Post.id",
                "direction": "in"
            }
        }
  ],
  "loader": {
    "orientdb": {
       "dbURL": "plocal:/temp/databases/blog",
       "dbType": "graph",
       "classes": [
         {"name": "Post", "extends": "V"},
         {"name": "Comment", "extends": "V"},
         {"name": "HasComments", "extends": "E"}
       ], "indexes": [
         {"class":"Post", "fields":["id:integer"], "type":"UNIQUE" }
       ]
    }
  }
}
```

This file is similar to the previous one, but the Edge transformer does the job. Since the link found in the CSV goes in the opposite direction (Comment->Post), while we want to model directly (Post->Comment), we used the direction "in" (default is always "out").

(3) Run the ETL process

Now allow the ETL to run by executing both imports in sequence. Open a shell under the OrientDB home directory, and execute the following steps:

```
$ cd bin
$ ./oetl.sh post.json
$ ./oetl.sh comment.json
```

Once both scripts execute successfully, you'll have your Blog imported into OrientDB as a Graph!

(4) Check the database

Open the database under the OrientDB console and execute the following commands to check that the import is ok:

```
$ ./console.sh

OrientDB console v.2.0-SNAPSHOT (build 2565) www.orientechnologies.com
Type 'help' to display all the supported commands.
Installing extensions for GREMLIN language v.2.6.0

orientdb> connect plocal:/temp/databases/blog admin admin

Connecting to database [plocal:/temp/databases/blog] with user 'admin'...OK

orientdb {db=blog}> select expand( out() ) from Post where id = 10

----+-----+-------+----+------+-------+--------------
#   |@RID |@CLASS |id  |postId|text   |in_HasComments
----+-----+-------+----+------+-------+--------------
0   |#12:0|Comment|0   |10    |First  |[size=1]
1   |#12:1|Comment|1   |10    |Second |[size=1]
2   |#12:2|Comment|21  |10    |Another|[size=1]
----+-----+-------+----+------+-------+--------------

3 item(s) found. Query executed in 0.002 sec(s).
orientdb {db=blog}> select expand( out() ) from Post where id = 20

----+-----+-------+----+------+------------+--------------
#   |@RID |@CLASS |id  |postId|text        |in_HasComments
----+-----+-------+----+------+------------+--------------
0   |#12:3|Comment|41  |20    |First again |[size=1]
1   |#12:4|Comment|82  |20    |Second Again|[size=1]
----+-----+-------+----+------+------------+--------------

2 item(s) found. Query executed in 0.001 sec(s).
```

Import a tree structure

If you have a tree structure in an RDBMS or CSV file and you want to import it in OrientDB, the ETL can come to your rescue. In this example, we use CSV for the sake of simplicity, but it's the same with JDBC input and a SQL query against an RDBMS.

source.csv

```
ID,PARENT_ID,LAST_YEAR_INCOME,DATE_OF_BIRTH,STATE
0,-1,10000,1990-08-11,Arizona
1,0,12234,1976-11-07,Missouri
2,0,21322,1978-01-01,Minnesota
3,0,33333,1960-05-05,Iowa
```

etl.json

```
{
  "source": { "file": { "path": "source.csv" } },
  "extractor": { "row": {} },
  "transformers": [
    { "csv": {} },
    { "vertex": { "class": "User" } },
    { "edge": {
        "class": "ParentOf",
        "joinFieldName": "PARENT_ID",
        "direction": "in",
        "lookup": "User.ID",
                "unresolvedLinkAction": "SKIP"
        }
    }
  ],
  "loader": {
    "orientdb": {
       "dbURL": "plocal:/temp/mydb",
       "dbType": "graph",
       "classes": [
         {"name": "User", "extends": "V"},
         {"name": "ParentOf", "extends": "E"}
       ], "indexes": [
         {"class":"User", "fields":["ID:Long"], "type":"UNIQUE" }
       ]
    }
  }
}
```

Import from JSON

If you are migrating from MongoDB or any other DBMS that exports data in JSON format, the JSON extractor is what you need. For more information look also at: Import-from-PARSE.

This is the input file stored in `/tmp/database.json` file:

```
[
 {
  "name": "Joe",
  "id": 1,
  "friends": [2,4,5],
  "enemies": [6]
 },
 {
  "name": "Suzie",
  "id": 2,
  "friends": [1,4,6],
  "enemies": [5,2]
 }
]
```

Note that `friends` and `enemies` represent relationships with nodes of the same type. They are in the form of an array of IDs. This is what we need:

- Use the Vertex class "Account" to store nodes
- Use the Edge classes "Friend" and "Enemy" to connect vertices
- Merge and Lookups will be on `id` property of Account class that will be unique
- In case the connected friend hasn't been inserted yet, create it ("unresolvedLinkAction": "CREATE")
- To speed up lookups, a unique index will be created on `Account.it`

And this pipeline (log is at `debug` level to show all the messages):

```
{
  "config": {
    "log": "debug"
  },
  "source" : {
    "file": { "path": "/tmp/database.json" }
  },
  "extractor" : {
    "json": {}
  },
  "transformers" : [
    { "merge": { "joinFieldName": "id", "lookup": "Account.id" } },
    { "vertex": { "class": "Account"} },
    { "edge": {
      "class": "Friend",
      "joinFieldName": "friends",
      "lookup": "Account.id",
      "unresolvedLinkAction": "CREATE"
    } },
    { "edge": {
      "class": "Enemy",
      "joinFieldName": "enemies",
      "lookup": "Account.id",
      "unresolvedLinkAction": "CREATE"
    } }
  ],
  "loader" : {
    "orientdb": {
      "dbURL": "plocal:/tmp/databases/db",
      "dbUser": "admin",
      "dbPassword": "admin",
      "dbAutoDropIfExists": true,
      "dbAutoCreate": true,
      "standardElementConstraints": false,
      "tx": false,
      "wal": false,
      "batchCommit": 1000,
      "dbType": "graph",
      "classes": [{"name": "Account", "extends":"V"}, {"name": "Friend", "extends":"E"}, {"name": 'Enemy', "extends":"E"}],
      "indexes": [{"class":"Account", "fields":["id:integer"], "type":"UNIQUE_HASH_INDEX" }]
    }
  }
}
```

Note also the setting

```
      "standardElementConstraints": false,
```

This is needed, in order to allow importing the property "id" in the OrientDB Loader. Without this option, the Blueprints standard would reject it, because "id" is a reserved name.

By executing the ETL process, this is the output:

```
OrientDB etl v.2.1-SNAPSHOT www.orientechnologies.com
feb 09, 2015 2:46:42 AM com.orientechnologies.common.log.OLogManager log
INFORMAZIONI: OrientDB auto-config DISKCACHE=10.695MB (heap=3.641MB os=16.384MB disk=42.205MB)
[orientdb] INFO Dropping existent database 'plocal:/tmp/databases/db'...
BEGIN ETL PROCESSOR
[file] DEBUG Reading from file /tmp/database.json
[orientdb] DEBUG - OrientDBLoader: created vertex class 'Account' extends 'V'
[orientdb] DEBUG orientdb: found 0 vertices in class 'null'
[orientdb] DEBUG - OrientDBLoader: created edge class 'Friend' extends 'E'
[orientdb] DEBUG orientdb: found 0 vertices in class 'null'
[orientdb] DEBUG - OrientDBLoader: created edge class 'Enemy' extends 'E'
[orientdb] DEBUG orientdb: found 0 vertices in class 'null'
[orientdb] DEBUG - OrientDBLoader: created property 'Account.id' of type: integer
[orientdb] DEBUG - OrientDocumentLoader: created index 'Account.id' type 'UNIQUE_HASH_INDEX' against Class 'Account', fields [
id:integer]
[0:merge] DEBUG Transformer input: {name:Joe,id:1,friends:[3],enemies:[1]}
[0:merge] DEBUG joinValue=1, lookupResult=null
[0:merge] DEBUG Transformer output: {name:Joe,id:1,friends:[3],enemies:[1]}
[0:vertex] DEBUG Transformer input: {name:Joe,id:1,friends:[3],enemies:[1]}
[0:vertex] DEBUG Transformer output: v(Account)[#11:0]
[0:edge] DEBUG Transformer input: v(Account)[#11:0]
[0:edge] DEBUG joinCurrentValue=2, lookupResult=null
[0:edge] DEBUG created new vertex=Account#11:1{id:2} v1
[0:edge] DEBUG created new edge=e[#12:0][#11:0-Friend->#11:1]
[0:edge] DEBUG joinCurrentValue=4, lookupResult=null
[0:edge] DEBUG created new vertex=Account#11:2{id:4} v1
[0:edge] DEBUG created new edge=e[#12:1][#11:0-Friend->#11:2]
[0:edge] DEBUG joinCurrentValue=5, lookupResult=null
[0:edge] DEBUG created new vertex=Account#11:3{id:5} v1
[0:edge] DEBUG created new edge=e[#12:2][#11:0-Friend->#11:3]
[0:edge] DEBUG Transformer output: v(Account)[#11:0]
[0:edge] DEBUG Transformer input: v(Account)[#11:0]
[0:edge] DEBUG joinCurrentValue=6, lookupResult=null
[0:edge] DEBUG created new vertex=Account#11:4{id:6} v1
[0:edge] DEBUG created new edge=e[#13:0][#11:0-Enemy->#11:4]
[0:edge] DEBUG Transformer output: v(Account)[#11:0]
[1:merge] DEBUG Transformer input: {name:Suzie,id:2,friends:[3],enemies:[2]}
[1:merge] DEBUG joinValue=2, lookupResult=Account#11:1{id:2,in_Friend:[#12:0]} v2
[1:merge] DEBUG merged record Account#11:1{id:2,in_Friend:[#12:0],name:Suzie,friends:[3],enemies:[2]} v2 with found record={na
me:Suzie,id:2,friends:[3],enemies:[2]}
[1:merge] DEBUG Transformer output: Account#11:1{id:2,in_Friend:[#12:0],name:Suzie,friends:[3],enemies:[2]} v2
[1:vertex] DEBUG Transformer input: Account#11:1{id:2,in_Friend:[#12:0],name:Suzie,friends:[3],enemies:[2]} v2
[1:vertex] DEBUG Transformer output: v(Account)[#11:1]
[1:edge] DEBUG Transformer input: v(Account)[#11:1]
[1:edge] DEBUG joinCurrentValue=1, lookupResult=Account#11:0{name:Joe,id:1,friends:[3],enemies:[1],out_Friend:[#12:0, #12:1, #
12:2],out_Enemy:[#13:0]} v5
[1:edge] DEBUG created new edge=e[#12:3][#11:1-Friend->#11:0]
[1:edge] DEBUG joinCurrentValue=4, lookupResult=Account#11:2{id:4,in_Friend:[#12:1]} v2
[1:edge] DEBUG created new edge=e[#12:4][#11:1-Friend->#11:2]
[1:edge] DEBUG joinCurrentValue=6, lookupResult=Account#11:4{id:6,in_Enemy:[#13:0]} v2
[1:edge] DEBUG created new edge=e[#12:5][#11:1-Friend->#11:4]
[1:edge] DEBUG Transformer output: v(Account)[#11:1]
[1:edge] DEBUG Transformer input: v(Account)[#11:1]
[1:edge] DEBUG joinCurrentValue=5, lookupResult=Account#11:3{id:5,in_Friend:[#12:2]} v2
[1:edge] DEBUG created new edge=e[#13:1][#11:1-Enemy->#11:3]
[1:edge] DEBUG joinCurrentValue=2, lookupResult=Account#11:1{id:2,in_Friend:[#12:0],name:Suzie,friends:[3],enemies:[2],out_Fri
end:[#12:3, #12:4, #12:5],out_Enemy:[#13:1]} v6
[1:edge] DEBUG created new edge=e[#13:2][#11:1-Enemy->#11:1]
[1:edge] DEBUG Transformer output: v(Account)[#11:1]
END ETL PROCESSOR
+ extracted 2 entries (0 entries/sec) - 2 entries -> loaded 2 vertices (0 vertices/sec) Total time: 228ms [0 warnings, 0 error
s]
```

Once ready, let's open the database with Studio and this is the result:

ETL - Import from RDBMS

Most of DBMSs support JDBC driver. All you need is to gather the JDBC driver and put it in classpath or simply in the $ORIENTDB_HOME/lib directory.

With the configuration below all the records from the table "Client" are imported in OrientDB from MySQL database.

Example importing a flat table

```
{
  "config": {
    "log": "debug"
  },
  "extractor" : {
    "jdbc": { "driver": "com.mysql.jdbc.Driver",
              "url": "jdbc:mysql://localhost/mysqlcrm",
              "userName": "root",
              "userPassword": "",
              "query": "select * from Client" }
  },
  "transformers" : [
   { "vertex": { "class": "Client"} }
  ],
  "loader" : {
    "orientdb": {
      "dbURL": "plocal:/temp/databases/orientdbcrm",
      "dbAutoCreate": true
    }
  }
}
```

Example loading records from 2 connected tables

With this example we want to import a database that contains Blog posts in the following tables:

- Authors, in TABLE **Author**, with the following columns: **id** and **name**
- Posts, in TABLE **Post**, with the following columns: **author_id**, **title** and **text**

To import them into OrientDB we'd need 2 ETL processes.

Importing of Authors

```
{
  "config": {
    "log": "debug"
  },
  "extractor" : {
    "jdbc": { "driver": "com.mysql.jdbc.Driver",
              "url": "jdbc:mysql://localhost/mysql",
              "userName": "root",
              "userPassword": "",
              "query": "select * from Author" }
  },
  "transformers" : [
   { "vertex": { "class": "Author"} }
  ],
  "loader" : {
    "orientdb": {
      "dbURL": "plocal:/temp/databases/orientdb",
      "dbAutoCreate": true
    }
  }
}
```

Importing of Posts

```
{
  "config": {
    "log": "debug"
  },
  "extractor" : {
    "jdbc": { "driver": "com.mysql.jdbc.Driver",
              "url": "jdbc:mysql://localhost/mysql",
              "userName": "root",
              "userPassword": "",
              "query": "select * from Post" }
  },
  "transformers" : [
   { "vertex": { "class": "Post"} },
   { "edge": { "class": "Wrote", "direction" : "in",
             "joinFieldName": "author_id",
             "lookup":"Author.id", "unresolvedLinkAction":"CREATE"} }
  ],
  "loader" : {
    "orientdb": {
      "dbURL": "plocal:/temp/databases/orientdb",
      "dbAutoCreate": true
    }
  }
}
```

Note the edge configuration has the direction as "in", that means starts from the Author and finishes to Post.

Import from DB-Pedia

DBPedia exports all the entities as GZipped CSV files. Features:

- First line contains column names, second, third and forth has meta information, which we'll skip (look at `"skipFrom": 1, "skipTo": 3` in CSV transformer)
- The vertex class name is created automatically based on the file name, so we can use the same file against any DBPedia file
- The Primary Key is the "URI" field, where a UNIQUE index has also been created (refer to "ORIENTDB" loader)
- The "merge" transformer is used to allow to re-import or update any file without generating duplicates

Configuration

```
{
  "config": {
    "log": "debug",
    "fileDirectory": "/temp/databases/dbpedia_csv/",
    "fileName": "Person.csv.gz"
  },
  "begin": [
   { "let": { "name": "$filePath",  "value": "$fileDirectory.append( $fileName )"} },
   { "let": { "name": "$className", "value": "$fileName.substring( 0, $fileName.indexOf('.') )"} }
  ],
  "source" : {
    "file": { "path": "$filePath", "lock" : true }
  },
  "extractor" : {
   { "csv": { "separator": ",", "nullValue": "NULL", "skipFrom": 1, "skipTo": 3 } },
  },
  "transformers" : [
   { "merge": { "joinFieldName":"URI", "lookup":"V.URI" } },
   { "vertex": { "class": "$className"} }
  ],
  "loader" : {
    "orientdb": {
      "dbURL": "plocal:/temp/databases/dbpedia",
      "dbUser": "admin",
      "dbPassword": "admin",
      "dbAutoCreate": true,
      "tx": false,
      "batchCommit": 1000,
      "dbType": "graph",
      "indexes": [{"class":"V", "fields":["URI:string"], "type":"UNIQUE" }]
    }
  }
}
```

Import from Parse

Parse is a very popular BaaS (Backend as a Service), acquired by Facebook. Parse uses MongoDB as a database and allows to export the database in JSON format. The format is an array of JSON objects. Example:

```
[
    {
        "user": {
            "__type": "Pointer",
            "className": "_User",
            "objectId": "Ldlskf4mfS"
        },
        "address": {
            "__type": "Pointer",
            "className": "Address",
            "objectId": "lvkDfj4dmS"
        },
        "createdAt": "2013-11-15T18:15:59.336Z",
        "updatedAt": "2014-02-27T23:47:00.440Z",
        "objectId": "Ldk39fDkcj",
        "ACL": {
            "Lfo33mfDkf": {
                "write": true
            },
            "*": {
                "read": true
            }
        }
    }, {
        "user": {
            "__type": "Pointer",
            "className": "_User",
            "objectId": "Lflfem3mFe"
        },
        "address": {
            "__type": "Pointer",
            "className": "Address",
            "objectId": "Ldldjfj3dd"
        },
        "createdAt": "2014-01-01T18:04:02.321Z",
        "updatedAt": "2014-01-23T20:12:23.948Z",
        "objectId": "fkfj49fjFFN",
        "ACL": {
            "dlfnDJckss": {
                "write": true
            },
            "*": {
                "read": true
            }
        }
    }
]
```

Notes:

- Each object has its own `objectId` that identifies the object in the entire database.
- Parse has the concept of `class`, like OrientDB.
- Links are similar to OrientDB RID (but it requires a costly JOIN to be traversed), but made as an embedded object containing:
  - `className` as target class name
  - `objectId` as target objectId
- Parse has ACL at record level, like OrientDB.

In order to import a PARSE file, you need to create the ETL configuration using JSON as Extractor.

Example

In this example, we're going to import the file extracted from Parse containing all the records of the `user` class. Note the creation of the class `User` in OrientDB, which extends `V` (Base Vertex class). We created an index against property `User.objectId` to use the same ID, similar to Parse. If you execute this ETL import multiple times, the records in OrientDB will be updated thanks to the `merge` feature.

```
{
  "config": {
    "log": "debug"
  },
  "source" : {
    "file": { "path": "/temp/parse-user.json", "lock" : true }
  },
  "extractor" : {
    "json": {}
  },
  "transformers" : [
   { "merge": { "joinFieldName":"objectId", "lookup":"User.objectId" } },
   { "vertex": { "class": "User"} }
  ],
  "loader" : {
    "orientdb": {
      "dbURL": "plocal:/temp/databases/parse",
      "dbUser": "admin",
      "dbPassword": "admin",
      "dbAutoCreate": true,
      "tx": false,
      "batchCommit": 1000,
      "dbType": "graph",
      "classes": [
        {"name": "User", "extends": "V"}
      ],
      "indexes": [
        {"class":"User", "fields":["objectId:string"], "type":"UNIQUE_HASH_INDEX" }
      ]
    }
  }
}
```

See also:

Import from JSON.

Logging

OrientDB handles logs using the Java Logging Framework, which is bundled with the JVM. The specific format it uses derives from the `OLogFormatter` class, which defaults to:

```
<date> <level> <message> [<requester>]
```

- **`<date>`** Shows the date of the log entry, using the date format `YYYY-MM-DD HH:MM:SS:SSS` .
- **`<level>`** Shows the log level.
- **`<message>`** Shows the log message.
- **`<class>`** Shows the Java class that made the entry, (optional).

The supported levels are those contained in the JRE class `java.util.logging.Level` . From highest to lowest:

- `SEVERE`
- `WARNING`
- `INFO`
- `CONFIG`
- `FINE`
- `FINER`
- `FINEST`

By default, OrientDB installs two loggers:

- **`console`** : Logs to the shell or command-prompt that starts the application or the server. You can modify it by setting the `log.console.level` variable.
- **`file`** : Logs to the log file. You can modify it by setting the `log.file.level` variable.

Configuration File

You can configure logging strategies and policies by creating a configuration file that follows the Java Logging Messages configuration syntax. For example, consider the following from the `orientdb-server-log.properties` file:

```
# Specify the handlers to create in the root logger
# (all loggers are children of the root logger)
# The following creates two handlers
handlers = java.util.logging.ConsoleHandler, java.util.logging.FileHandler

# Set the default logging level for the root logger
.level = ALL

# Set the default logging level for new ConsoleHandler instances
java.util.logging.ConsoleHandler.level = INFO
# Set the default formatter for new ConsoleHandler instances
java.util.logging.ConsoleHandler.formatter = com.orientechnologies.common.log.OLogFormatter

# Set the default logging level for new FileHandler instances
java.util.logging.FileHandler.level = INFO
# Naming style for the output file
java.util.logging.FileHandler.pattern=../log/orient-server.log
# Set the default formatter for new FileHandler instances
java.util.logging.FileHandler.formatter = com.orientechnologies.common.log.OLogFormatter
# Limiting size of output file in bytes:
java.util.logging.FileHandler.limit=10000000
# Number of output files to cycle through, by appending an
# integer to the base file name:
java.util.logging.FileHandler.count=10
```

When the log properties file is ready, you need to tell the JVM to use t, by setting `java.util.logging.config.file` system property.

```
$ java -Djava.util.logging.config.file=mylog.properties
```

Setting the Log Level

To change the log level without modifying the logging configuration, set the `log.console.level` and `log.file.level` system variables. These system variables are accessible both at startup and at runtime.

Configuring Log Level at Startup

You can configure log level at startup through both the `orientdb-server-config.xml` configuration file and by modifying the JVM before you start the server:

Using the Configuration File

To configure log level from the configuration file, update the following elements in the `<properties>` section:

```
<properties>
   <entry value="info" name="log.console.level" />
   <entry value="fine" name="log.file.level" />
   ...
</properties>
```

Using the JVM

To configure log level from the JVM before starting the server, run the `java` command to configure the `log.console.level` and `log.file.level` variables:

```
$ java -Dlog.console.level=INFO -Dlog.file.level=FINE
```

Configuring Log Level at Runtime

You can configure log level at runtime through both the Java API and by executing an HTTP `POST` against the remote server.

Using Java Code

Through the Java API, you can set the system variables for logging at startup through the `System.setProperty()` method. For instance,

```
public void main(String[] args){
  System.setProperty("log.console.level", "FINE");
  ...
}
```

Using HTTP POST

Through the HTTP requests, you can update the logging system variables by executing a `POST` against the URL: `/server/log.<type>/<level>` .

- `<type>` Defines the log type: `console` or `file` .
- `<level>` Defines the log level.

Examples

The examples below use cURL to execute the HTTP `POST` commands against the OrientDB server. It uses the server `root` user and password.

- Enable the finest tracing level to the console:

```
$ curl -u root:root -X POST http://localhost:2480/server/log.console/FINEST
```

- Enable the finest tracing level to file:

  ```
  $ curl -u root:root -X POST http://localhost:2480/server/log.file/FINEST
  ```

Change logging on the client

On the client is the same as for the server, but you should rather configure the file `config/orientdb-client-log.properties` and add this at your client's JVM:

```
$ java -Djava.util.logging.config.file=config/orientdb-client-log.properties
```

Install Log Formatter

OrientDB Server uses its own log formatter. In order to enable the same for your application, you need to include the following line:

```
OLogManager.installCustomFormatter();
```

The Server automatically installs the log formatter. To disable it, use `orientdb.installCustomFormatter`.

```
$ java -Dorientdb.installCustomFormatter=false
```

Debugging Logger

Java Logging Framework runtime has a known problem with logging from shutdown hooks, sometimes log entries may be lost. OrientDB uses shutdown hooks to properly shutdown its internals, so it also may be affected by this problem, something may go wrong silently on shutdown. To workaround this problem OrientDB provides a special logger – the debugging logger. To activate it provide following command line argument to your JVM:

```
-Djava.util.logging.manager=com.orientechnologies.common.log.OLogManager$DebugLogManager
```

> Use this logger for debugging and troubleshooting purposes only, since it may interfere with your production logging configuration.
>
> Make sure `$DebugLogManager` part is not interpreted as a shell variable substitution. To avoid the substitution apply escaping specific to your shell environment.

Scheduler

OrientDB has a Scheduler of events you can use to fire your events on regular basis. To manage events you can use both SQL and Java API. The scheduler gets the popular CRON expression syntax. The scheduled event, when fired, executes a Database

Resources

- CRON expressions on Wikipedia
- CRON expression maker is an online resource to create CRON expressions

Schedule an event

Via SQL

In order to schedule a new event via SQL, all you need is to create a new record in the `OSchedule` class. Example on scheduling the event "myEvent" that calls the function "myFunction" every second:

```
INSERT INTO oschedule
  SET name = 'myEvent',
      function = (SELECT FROM ofunction WHERE name = 'myFunction'),
      rule = \"0/1 * * * * ?\"
```

Via Java API

```
db.getMetadata().getScheduler().scheduleEvent(
  new OScheduledEventBuilder().setName("myEvent").setRule("0/1 * * * * ?")
  .setFunction(func).build());
```

Update an event

Via SQL

To update the scheduling of an event, update the record with the new `rule`. Example:

```
UPDATE oschedule SET rule = "0/2 * * * * ?" WHERE name = 'myEvent'
```

Via Java API

To update an event, remove it and reschedule it.

```
db.getMetadata().getScheduler().removeEvent("myEvent");
db.getMetadata().getScheduler().scheduleEvent(
  new OScheduledEventBuilder().setName("myEvent").setRule("0/2 * * * * ?")
  .setFunction(func).build());
```

Remove an event

Via SQL

To cancel a scheduled event, just delete the record. Example:

```
DELETE oschedule WHERE name = 'myEvent'
```

Via Java API

```
db.getMetadata().getScheduler().removeEvent("myEvent");
```

Tutorial

In this tutorial we want to purge all the records older than 1 year.

1) Create a Function

First, create a SQL function that delete the records. To have the date of 1y ago you can use the expression `sysdate() - 31536000000`, where 31536000000 represents the number of milliseconds in a year. You can this via SQL or Java API.

Via SQL

```
CREATE FUNCTION purgeHistoricalRecords
  "DELETE FROM Logs WHERE date < ( sysdate() - 31536000000 )"
  LANGUAGE SQL
```

Via Java API

```
OFunction func = db.getMetadata().getFunctionLibrary().createFunction("purgeHistoricalRecords");
func.setLanguage("SQL");
func.setCode("DELETE FROM Logs WHERE date < ( sysdate() - 31536000000 )");
func.save();
return func;
```

2) Schedule the event

The second step is scheduling the event. The CRON expression for "every midnight" is `00 00 * * * ?`. You can this via SQL or Java API.

Via SQL

```
INSERT INTO oschedule
 SET name = 'purgeHistoricalRecordsAtMidnight',
     function = (SELECT FROM ofunction WHERE name = 'purgeHistoricalRecords'),
     rule = \"00 00 * * * ?\"
```

Via Java API

```
db.command(new OCommandSQL(
 "INSERT INTO oschedule SET name = 'purgeHistoricalRecordsAtMidnight', function = ?, rule = \"00 00 * * * ?\""))
 .execute(func.getId());
```

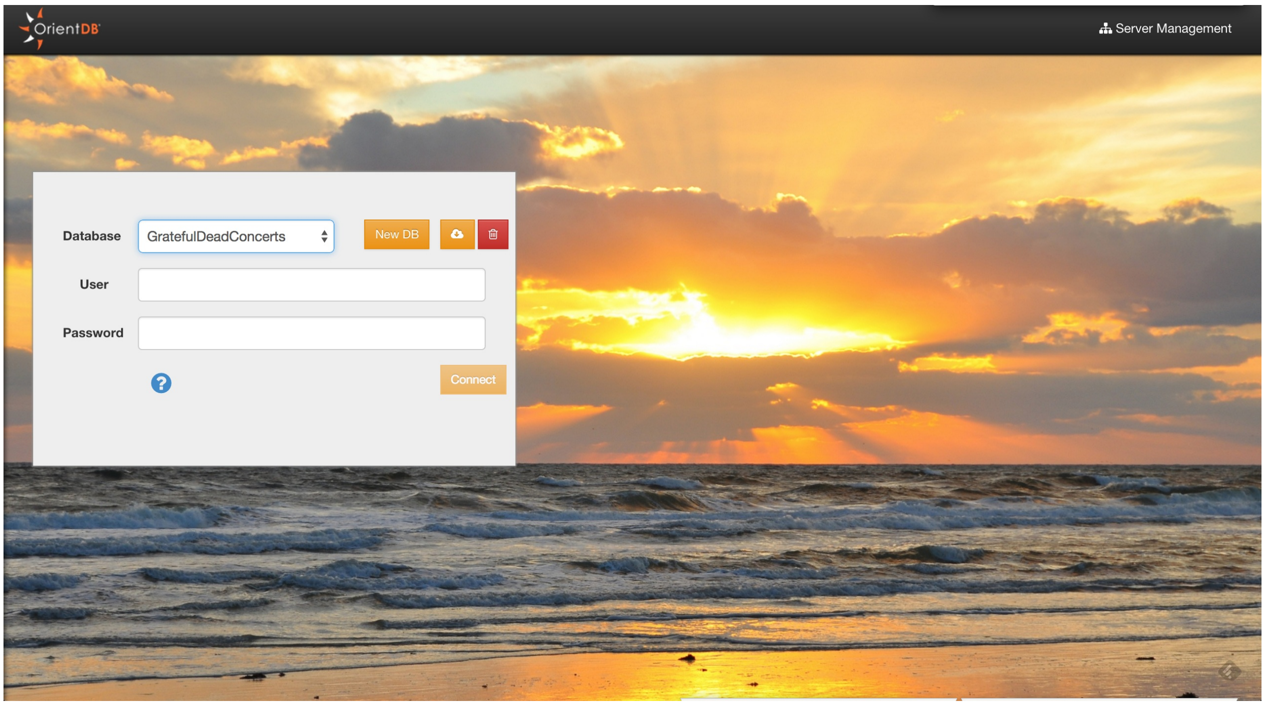
Studio Home page

Studio is a web interface for the administration of OrientDB that comes in bundle with the OrientDB distribution.

If you run OrientDB in your machine the web interface can be accessed via the URL:

```
http://localhost:2480
```

This is the Studio 2.2 Homepage.



From here, you can :

- Connect to an existing database
- Drop an existing database
- Create a new database
- Import a public database
- Go to the Server Management UI

Connect to an existing database

To Login, select a database from the databases list and use any database user. By default **reader/reader** can read records from the database, **writer/writer** can read, create, update and delete records. **admin/admin** has all rights.

Drop an existing database

Select a database from the databases list and click the trash icon. Studio will open a confirmation popup where you have to insert

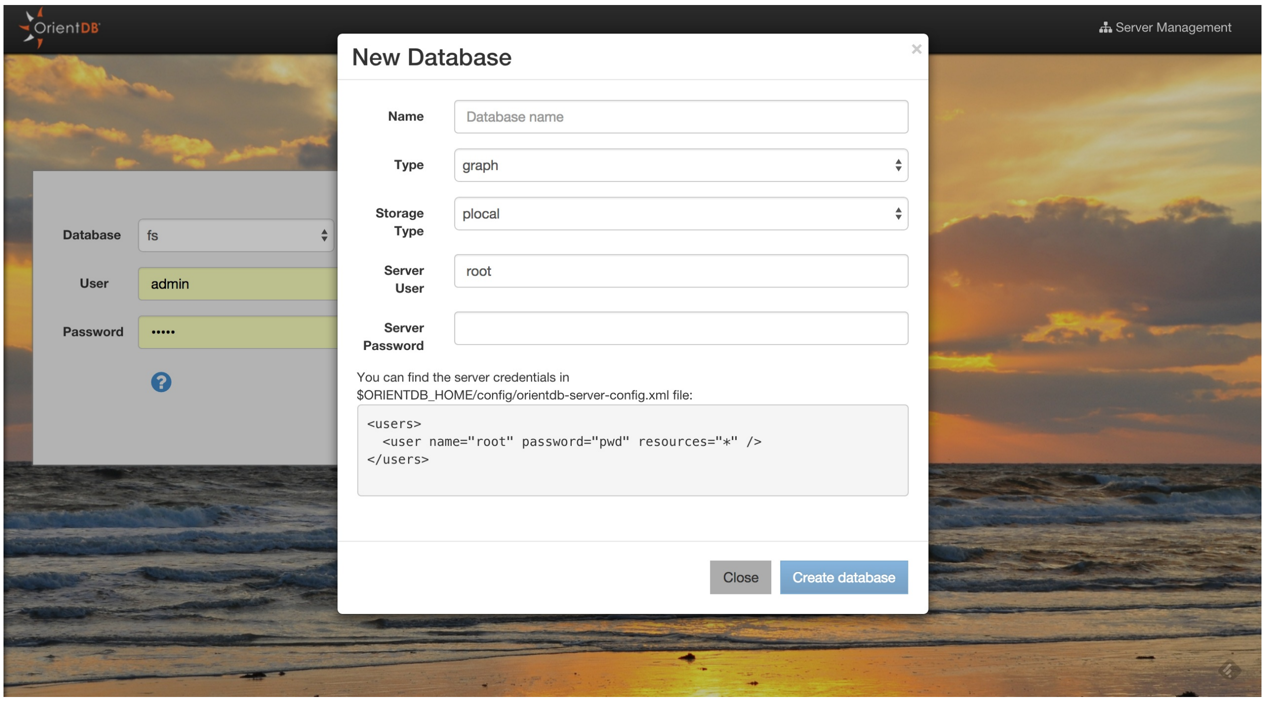
- Server User
- Server Password

and then click the "Drop database" button. You can find the server credentials in the $ORIENTDB_HOME/config/orientdb-server-config.xml file:

```
<users>
  <user name="root" password="pwd" resources="*" />
</users>
```

Create a new database

To create a new database, click the "New DB" button from the Home Page



Name: Database name
Type: graph
Storage Type: plocal
Server User: root
Server Password:

You can find the server credentials in $ORIENTDB_HOME/config/orientdb-server-config.xml file:

```
<users>
  <user name="root" password="pwd" resources="*" />
</users>
```

Some information is needed to create a new database:

- Database name
- Database type (Document/Graph)
- Storage type (plocal/memory)
- Server user
- Server password

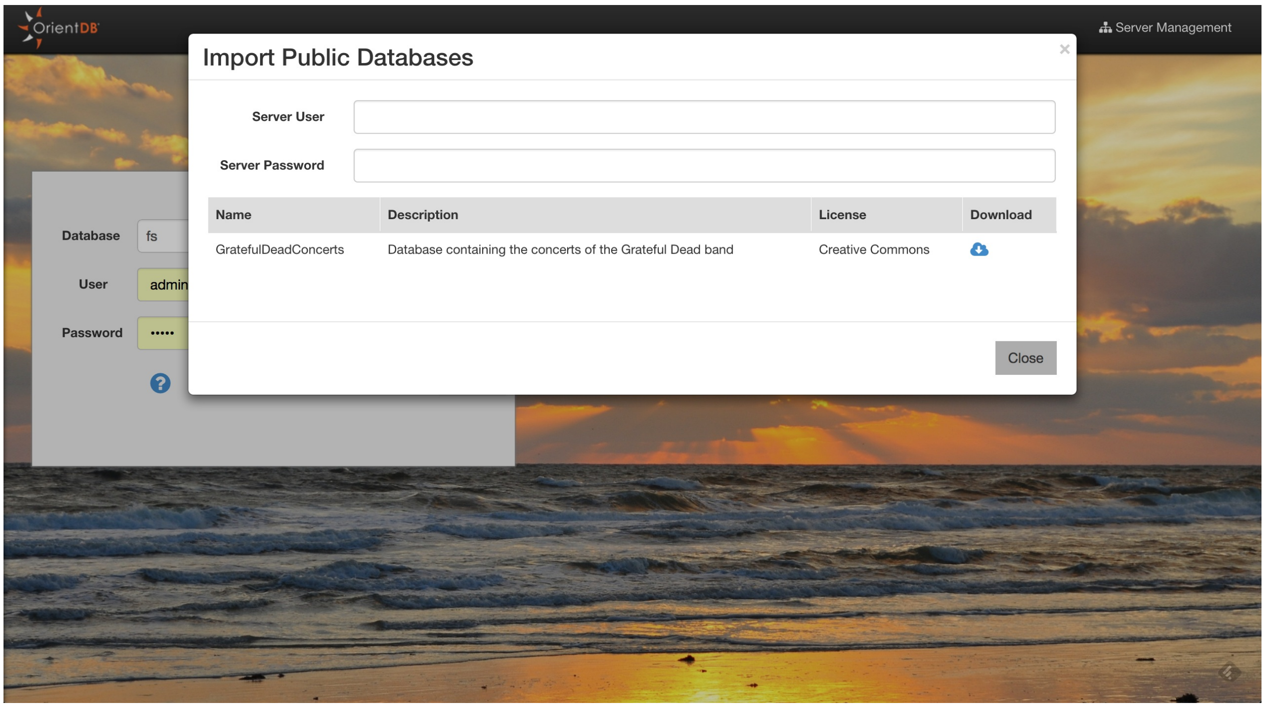
You can find the server credentials in the $ORIENTDB_HOME/config/orientdb-server-config.xml file:

```
<users>
  <user name="root" password="pwd" resources="*" />
</users>
```

Once created, Studio will automatically login to the new database.

Import a public database

Studio 2.2 allows you to import databases from a public repository. These databases contains public data and bookmarked queries that will allow you to start playing with OrientDB and OrientDB SQL. The classic bundle database 'GratefulDeadConcerts' will be moved to this public repository.



Import Public Databases

Server User	
Server Password	

Name	Description	License	Download
GratefulDeadConcerts	Database containing the concerts of the Grateful Dead band	Creative Commons	

Database	fs
User	admin
Password	•••••

To install a public database, you will need the Server Credentials. Then, click the download button of the database that you are interested in. Then Studio will download and install in to your $ORIENTDB_HOME/databases directory. Once finished, Studio will automatically login to the newly installed database.

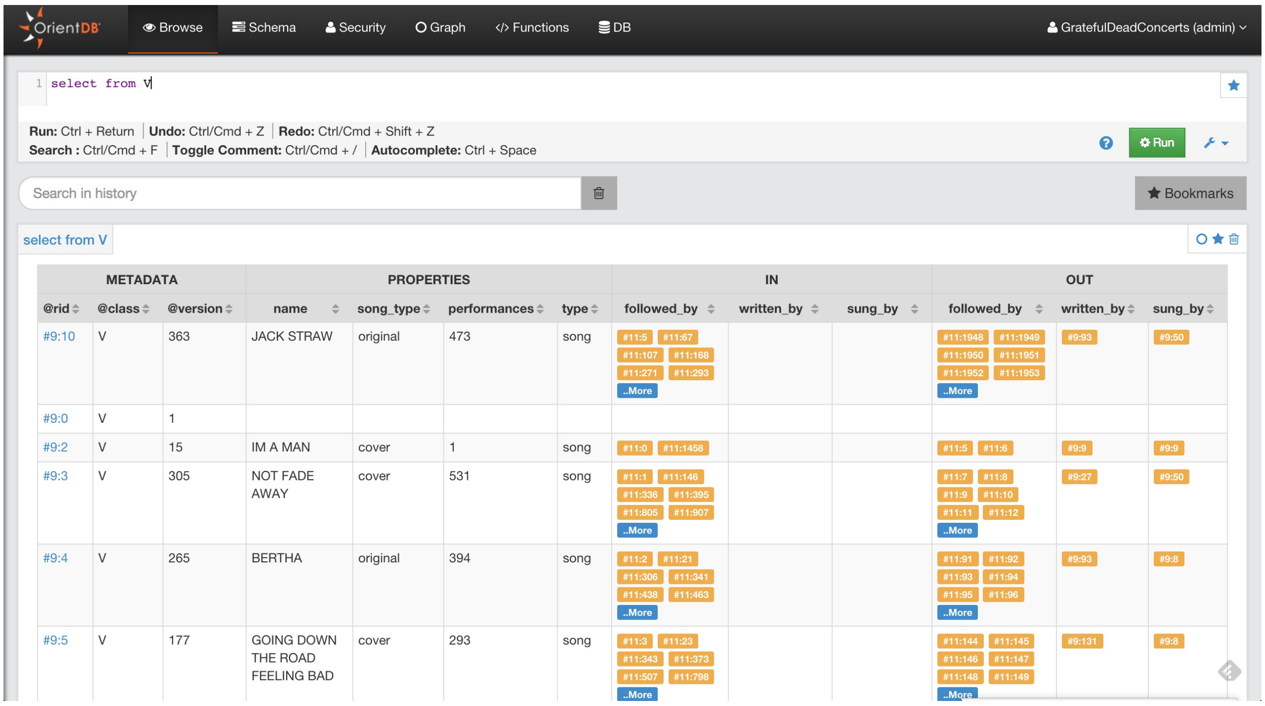
Execute a query

Studio supports auto recognition of the language you're using between those supported: SQL and Gremlin. While writing, use the auto-complete feature by pressing Ctrl + Space.

Other shortcuts are available in the query editor:

- **Ctrl + Return** to execute the query or just click the **Run** button
- **Ctrl/Cmd + Z** to undo changes
- **Ctrl/Cmd + Shift** + Z to redo changes
- **Ctrl/Cmd + F** to search in the editor
- **Ctrl/Cmd +** / to toggle a comment

> **Note:** If you have multiple queries in the editor, you can select a single query with text selection and execute it with **Ctrl** + **Return** or the **Run** button



```
select from V
```

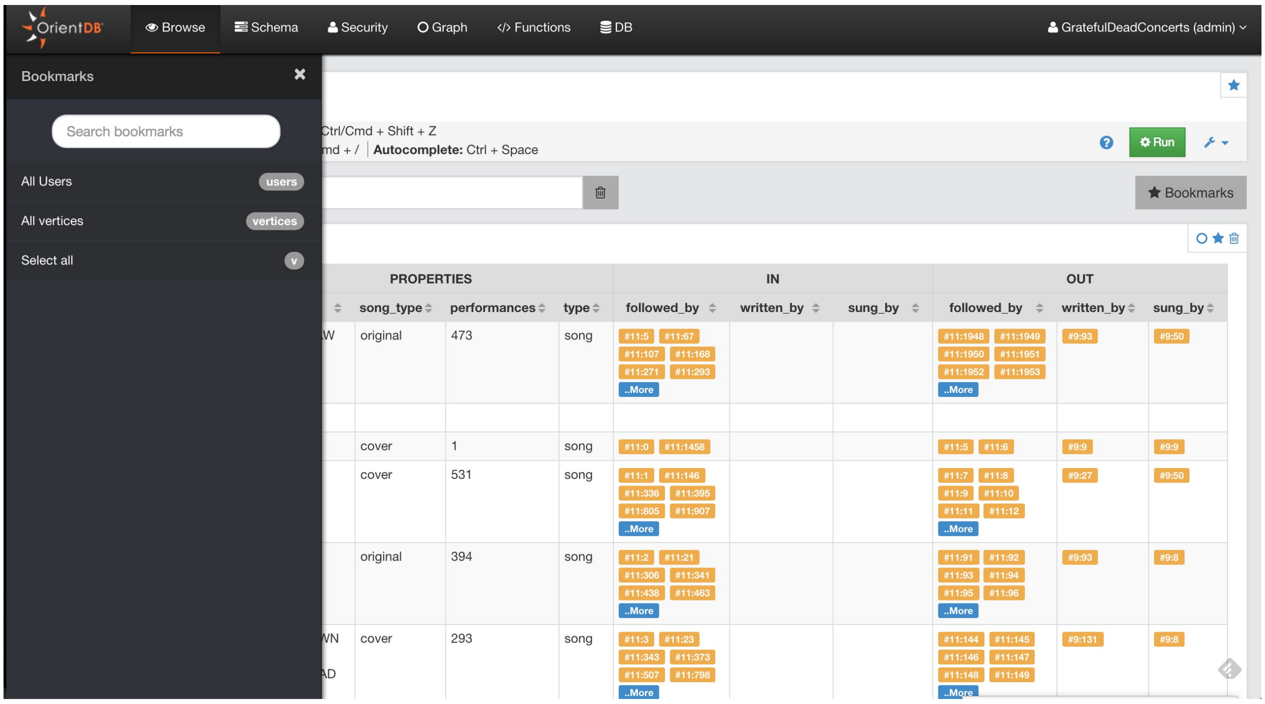
Run: Ctrl + Return | **Undo:** Ctrl/Cmd + Z | **Redo:** Ctrl/Cmd + Shift + Z
Search : Ctrl/Cmd + F | **Toggle Comment:** Ctrl/Cmd + / | **Autocomplete:** Ctrl + Space

select from V

METADATA			PROPERTIES				IN			OUT		
@rid	@class	@version	name	song_type	performances	type	followed_by	written_by	sung_by	followed_by	written_by	sung_by
#9:10	V	363	JACK STRAW	original	473	song	#11:5 #11:67 #11:107 #11:168 #11:271 #11:293 ..More			#11:1948 #11:1949 #11:1950 #11:1951 #11:1952 #11:1953 ..More	#9:93	#9:50
#9:0	V	1										
#9:2	V	15	IM A MAN	cover	1	song	#11:0 #11:1458			#11:5 #11:6	#9:9	#9:9
#9:3	V	305	NOT FADE AWAY	cover	531	song	#11:1 #11:146 #11:336 #11:395 #11:805 #11:907 ..More			#11:7 #11:8 #11:9 #11:10 #11:11 #11:12 ..More	#9:27	#9:50
#9:4	V	265	BERTHA	original	394	song	#11:2 #11:21 #11:306 #11:341 #11:438 #11:463 ..More			#11:91 #11:92 #11:93 #11:94 #11:95 #11:96 ..More	#9:93	#9:8
#9:5	V	177	GOING DOWN THE ROAD FEELING BAD	cover	293	song	#11:3 #11:23 #11:343 #11:373 #11:507 #11:798 ..More			#11:144 #11:145 #11:146 #11:147 #11:148 #11:149 ..More	#9:131	#9:8

By clicking any @rid value in the result set, you will go into document edit mode if the record is a Document, otherwise you will go into vertex edit.

You can bookmark your queries by clicking the star icon in the results set or in the editor. To browse bookmarked queries, click the **Bookmarks** button. Studio will open the bookmarks list on the left, where you can edit/delete or rerun queries.



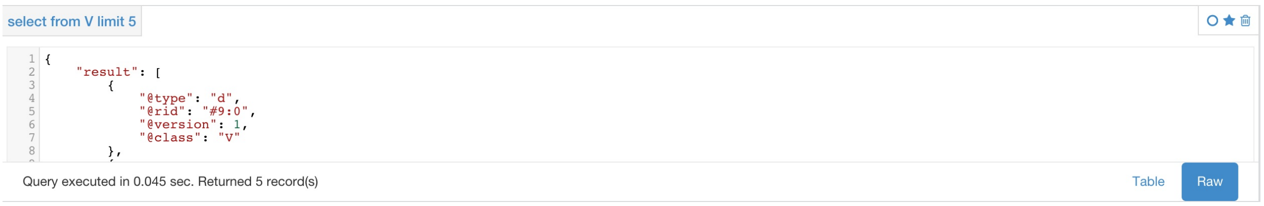
PROPERTIES				IN			OUT		
	song_type	performances	type	followed_by	written_by	sung_by	followed_by	written_by	sung_by
W	original	473	song	#11:5 #11:67 #11:107 #11:168 #11:271 #11:293 ..More			#11:1948 #11:1949 #11:1950 #11:1951 #11:1952 #11:1953 ..More	#9:93	#9:50
	cover	1	song	#11:0 #11:1458			#11:5 #11:6	#9:9	#9:9
	cover	531	song	#11:1 #11:146 #11:336 #11:395 #11:805 #11:907 ..More			#11:7 #11:8 #11:9 #11:10 #11:11 #11:12 ..More	#9:27	#9:50
	original	394	song	#11:2 #11:21 #11:306 #11:341 #11:438 #11:463 ..More			#11:91 #11:92 #11:93 #11:94 #11:95 #11:96 ..More	#9:93	#9:8
WN AD	cover	293	song	#11:3 #11:23 #11:343 #11:373 #11:507 #11:798 ..More			#11:144 #11:145 #11:146 #11:147 #11:148 #11:149 ..More	#9:131	#9:8

Studio saves the executed queries in the Local Storage of the browser, in the query settings, you can configure how many queries studio will keep in history. You can also search a previously executed query, delete all the queries from the history or delete a single query.

From Studio 2.0, you can send the result set of a query to the Graph Editor by clicking on the circle icon in the result set actions. This allows you to visualize your data graphically.

Look at the JSON output

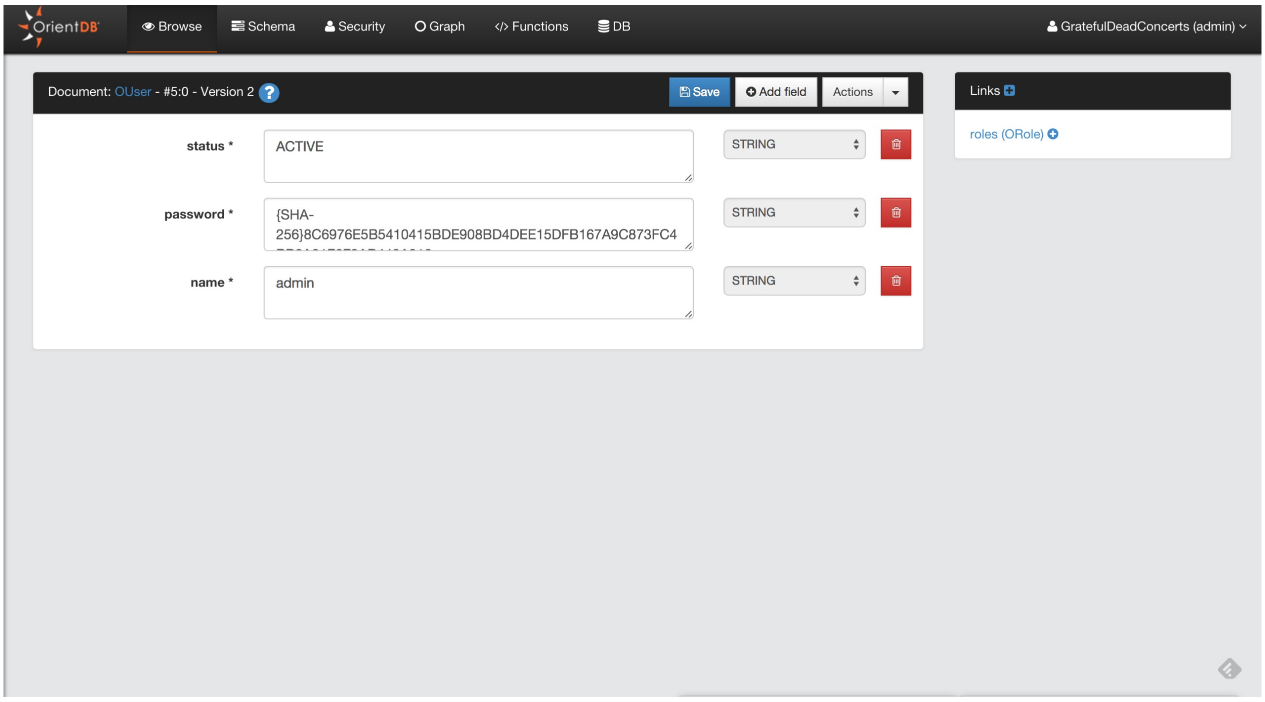
Studio communicates with the OrientDB Server using HTTP/RESt+JSON protocol. To see the output in JSON format, press the **RAW** tab.


select from V limit 5

```
{
    "result": [
        {
            "@type": "d",
            "@rid": "#9:0",
            "@version": 1,
            "@class": "V"
        },
```

Query executed in 0.045 sec. Returned 5 record(s)

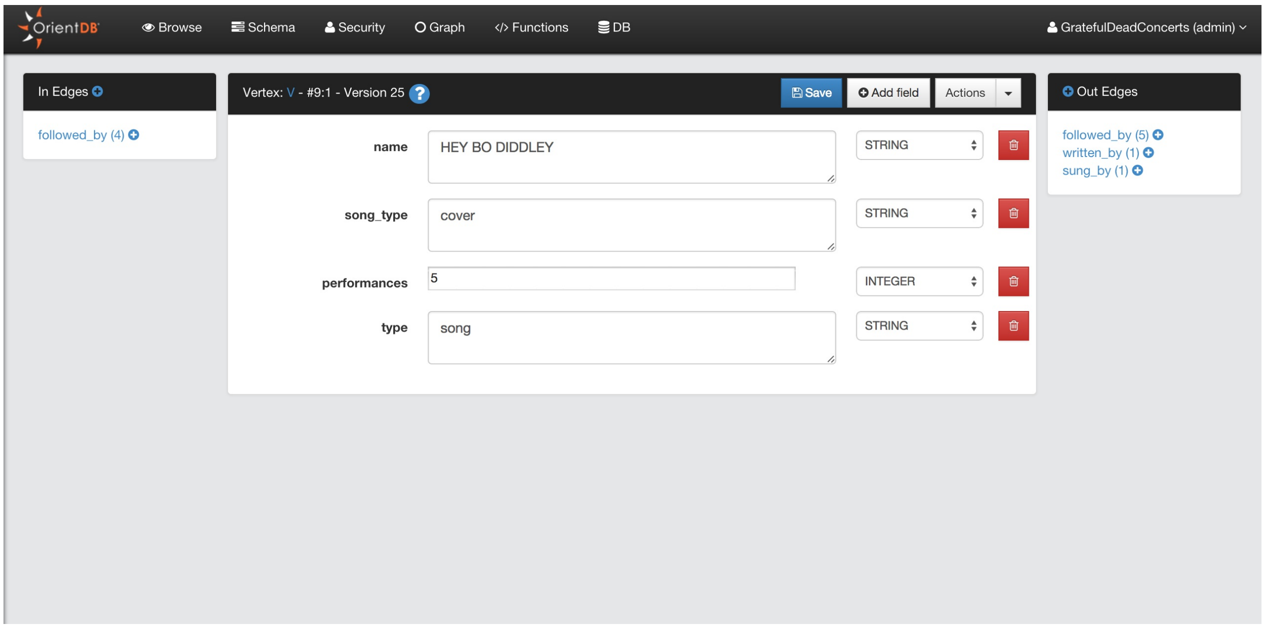
Edit Document



Document: OUser - #5:0 - Version 2

status *	ACTIVE	STRING
password *	{SHA-256}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4	STRING
name *	admin	STRING

Edit Vertex

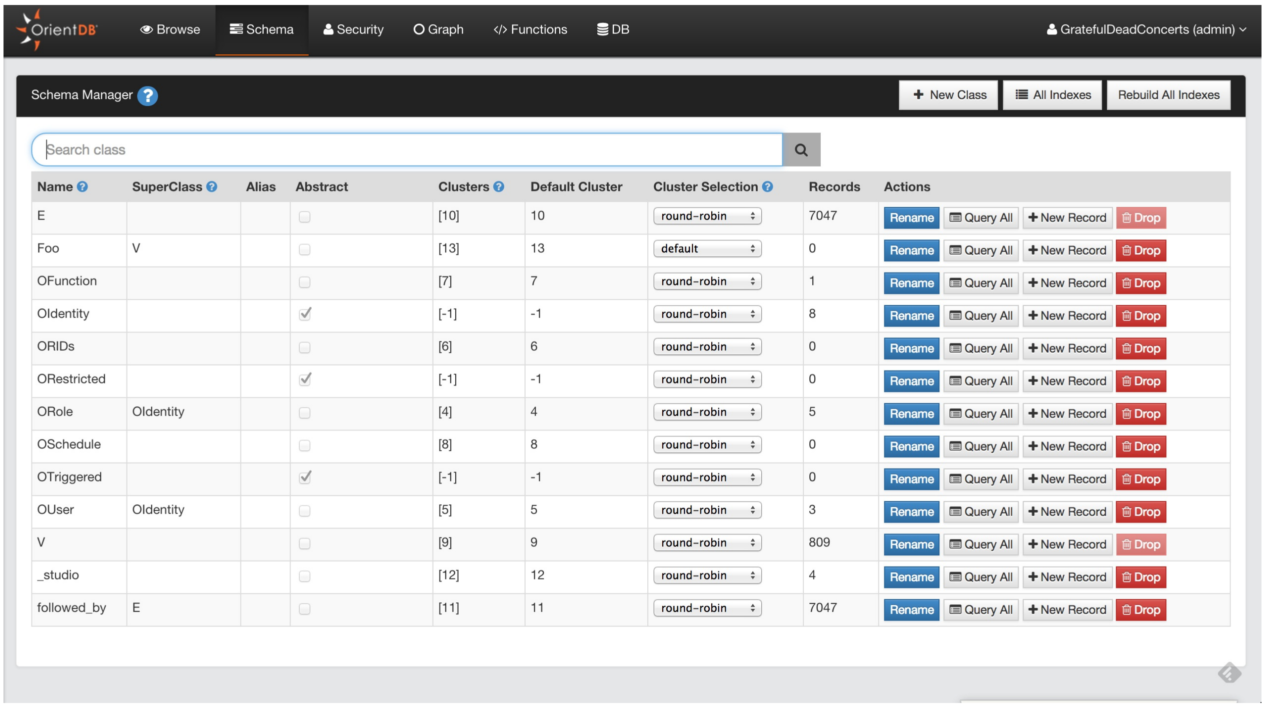


Vertex: V - #9:1 - Version 25

name	HEY BO DIDDLEY	STRING
song_type	cover	STRING
performances	5	INTEGER
type	song	STRING

Schema Manager

OrientDB can work in schema-less mode, schema mode or a mix of both. Here we'll discuss the schema mode. To know more about schema in OrientDB go here



Name	SuperClass	Alias	Abstract	Clusters	Default Cluster	Cluster Selection	Records	Actions
E				[10]	10	round-robin	7047	Rename Query All New Record Drop
Foo	V			[13]	13	default	0	Rename Query All New Record Drop
OFunction				[7]	7	round-robin	1	Rename Query All New Record Drop
OIdentity			✓	[-1]	-1	round-robin	8	Rename Query All New Record Drop
ORIDs				[6]	6	round-robin	0	Rename Query All New Record Drop
ORestricted			✓	[-1]	-1	round-robin	0	Rename Query All New Record Drop
ORole	OIdentity			[4]	4	round-robin	5	Rename Query All New Record Drop
OSchedule				[8]	8	round-robin	0	Rename Query All New Record Drop
OTriggered			✓	[-1]	-1	round-robin	0	Rename Query All New Record Drop
OUser	OIdentity			[5]	5	round-robin	3	Rename Query All New Record Drop
V				[9]	9	round-robin	809	Rename Query All New Record Drop
_studio				[12]	12	round-robin	4	Rename Query All New Record Drop
followed_by	E			[11]	11	round-robin	7047	Rename Query All New Record Drop

Here you can :

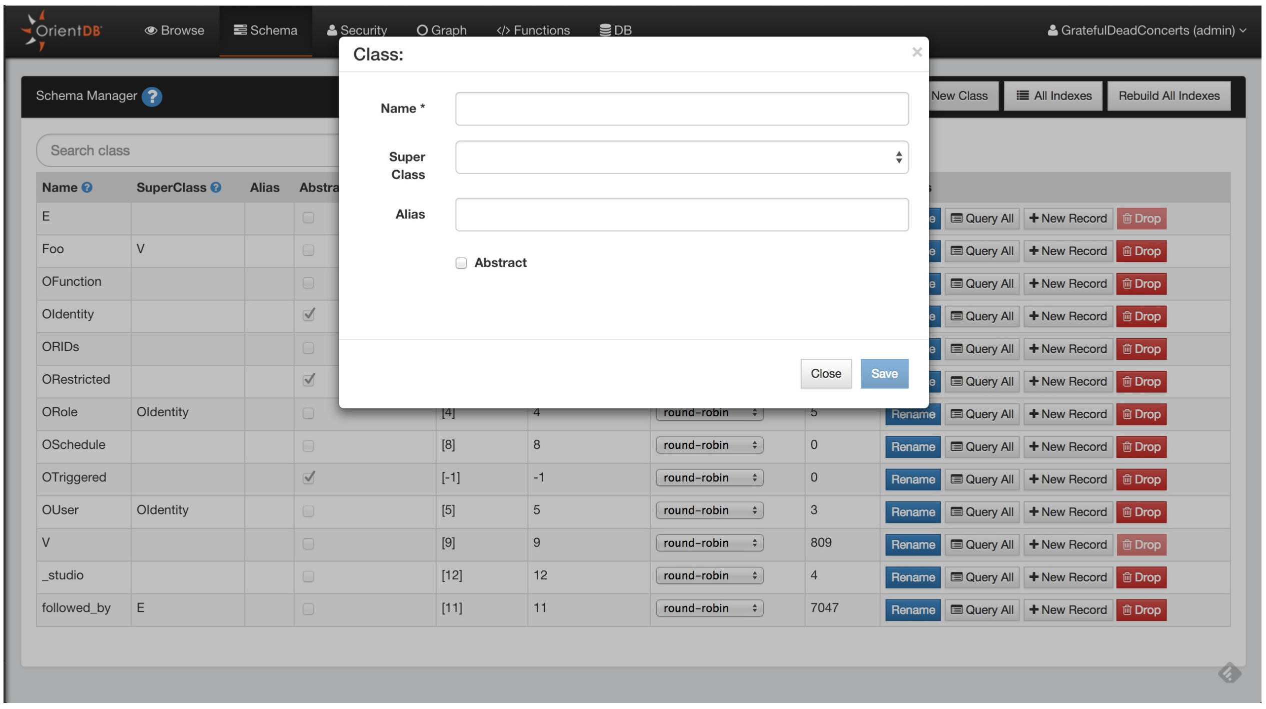
- Browse all the Classes of your database
- Create a new Class
- Rename/Drop a Class
- Change the cluster selection for a Class
- Edit a class by clicking on a class row in the table
- View all indexes created

Create a new Class

To create a new Class, just click the **New Class** button. Some information is required to create the new class.

- Name
- SuperClass
- Alias (Optional)
- Abstract

Here you can find more information about Classes



Class:

Name *

Super Class

Alias

Abstract

Close Save

Schema Manager

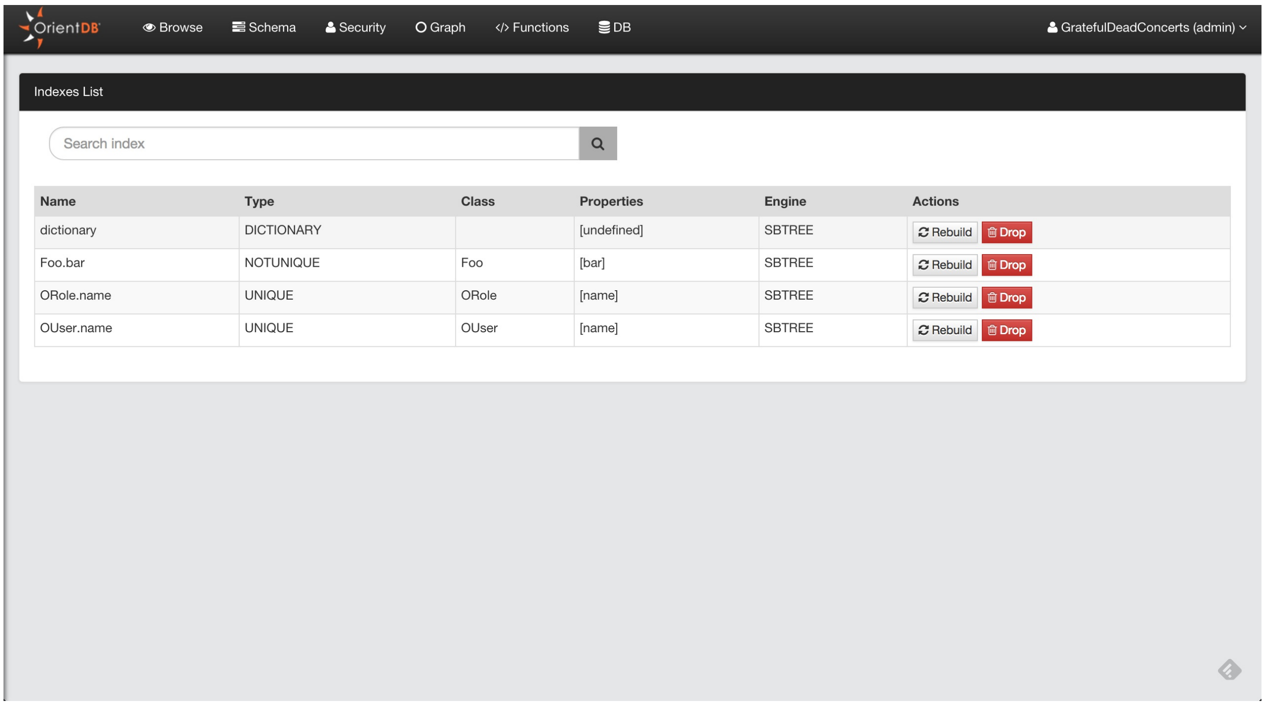
New Class | All Indexes | Rebuild All Indexes

Search class

Name	SuperClass	Alias	Abstra					
E			☐					Query All · New Record · Drop
Foo	V		☐					Query All · New Record · Drop
OFunction			☐					Query All · New Record · Drop
OIdentity			✓					Query All · New Record · Drop
ORIDs			☐					Query All · New Record · Drop
ORestricted			✓					Query All · New Record · Drop
ORole	OIdentity		☐	[4]	4	round-robin	5	Rename · Query All · New Record · Drop
OSchedule			☐	[8]	8	round-robin	0	Rename · Query All · New Record · Drop
OTriggered			✓	[-1]	-1	round-robin	0	Rename · Query All · New Record · Drop
OUser	OIdentity		☐	[5]	5	round-robin	3	Rename · Query All · New Record · Drop
V			☐	[9]	9	round-robin	809	Rename · Query All · New Record · Drop
_studio			☐	[12]	12	round-robin	4	Rename · Query All · New Record · Drop
followed_by	E		☐	[11]	11	round-robin	7047	Rename · Query All · New Record · Drop

View all indexes

When you want to have an overview of all indexes created in your database, just click the **All indexes** button in the Schema UI. This will provide quick access to some information about indexes (name, type, properties, etc) and you can drop or rebuild them from here.

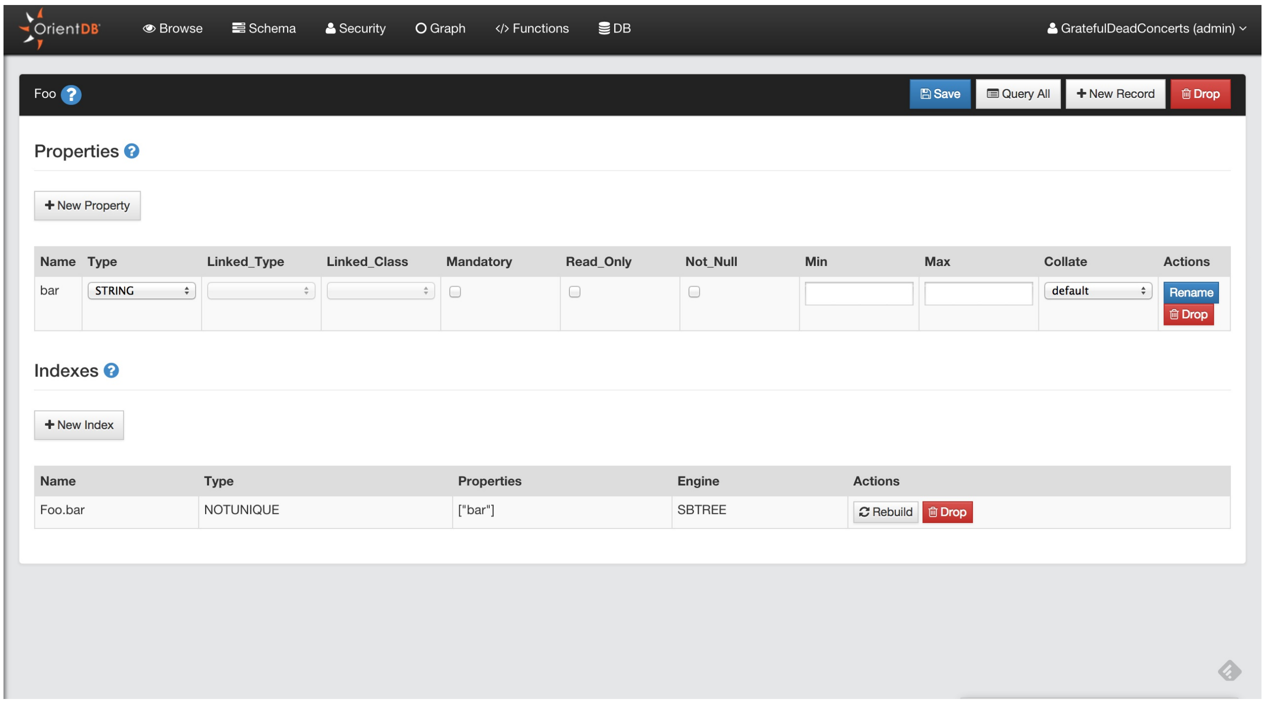


Indexes List

Search index

Name	Type	Class	Properties	Engine	Actions
dictionary	DICTIONARY		[undefined]	SBTREE	Rebuild · Drop
Foo.bar	NOTUNIQUE	Foo	[bar]	SBTREE	Rebuild · Drop
ORole.name	UNIQUE	ORole	[name]	SBTREE	Rebuild · Drop
OUser.name	UNIQUE	OUser	[name]	SBTREE	Rebuild · Drop

Class Edit

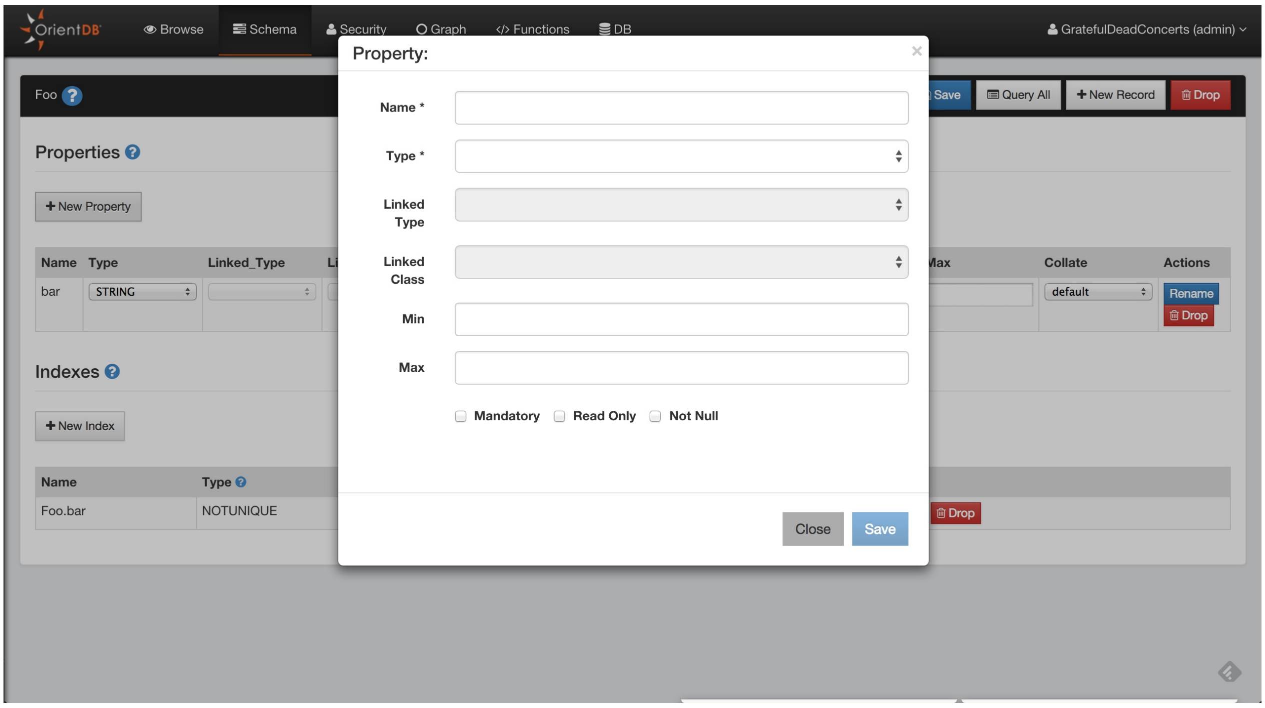


Name	Type	Linked_Type	Linked_Class	Mandatory	Read_Only	Not_Null	Min	Max	Collate	Actions
bar	STRING			☐	☐	☐			default	Rename Drop

Name	Type	Properties	Engine	Actions
Foo.bar	NOTUNIQUE	["bar"]	SBTREE	Rebuild Drop

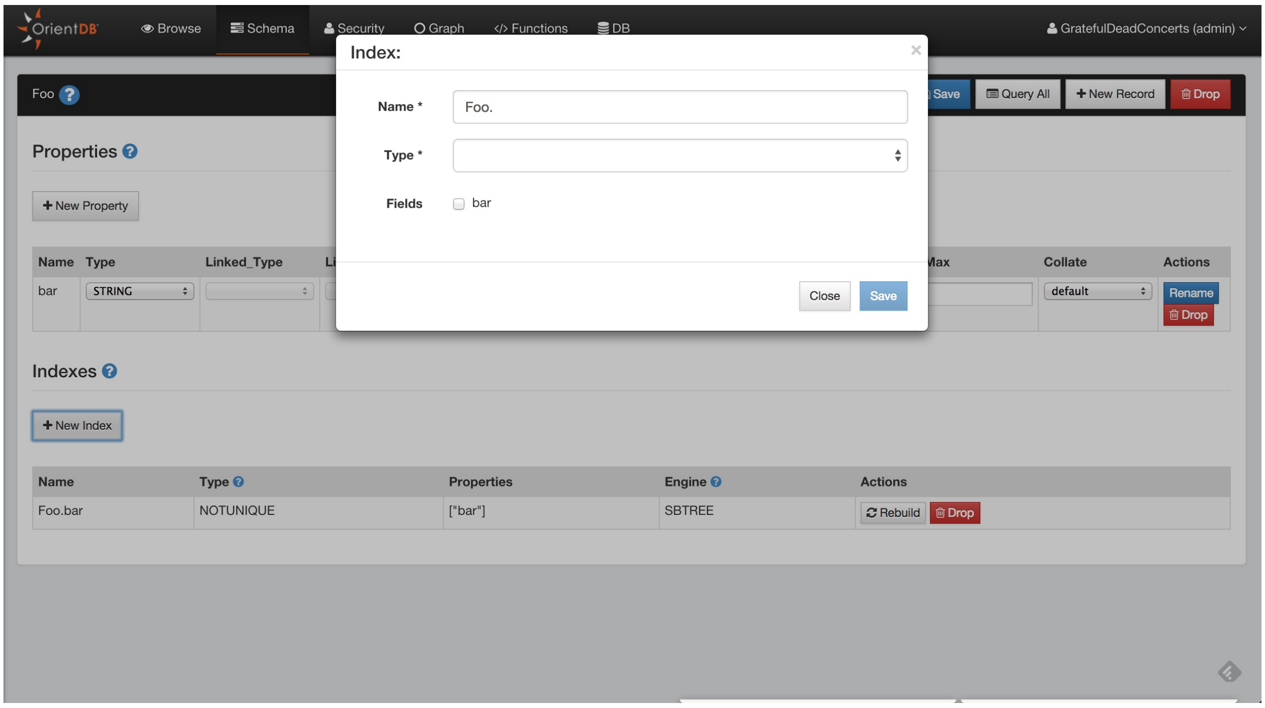
Property

Add Property



Indexes

Create new index



Index:

Name * Foo.

Type *

Fields ☐ bar

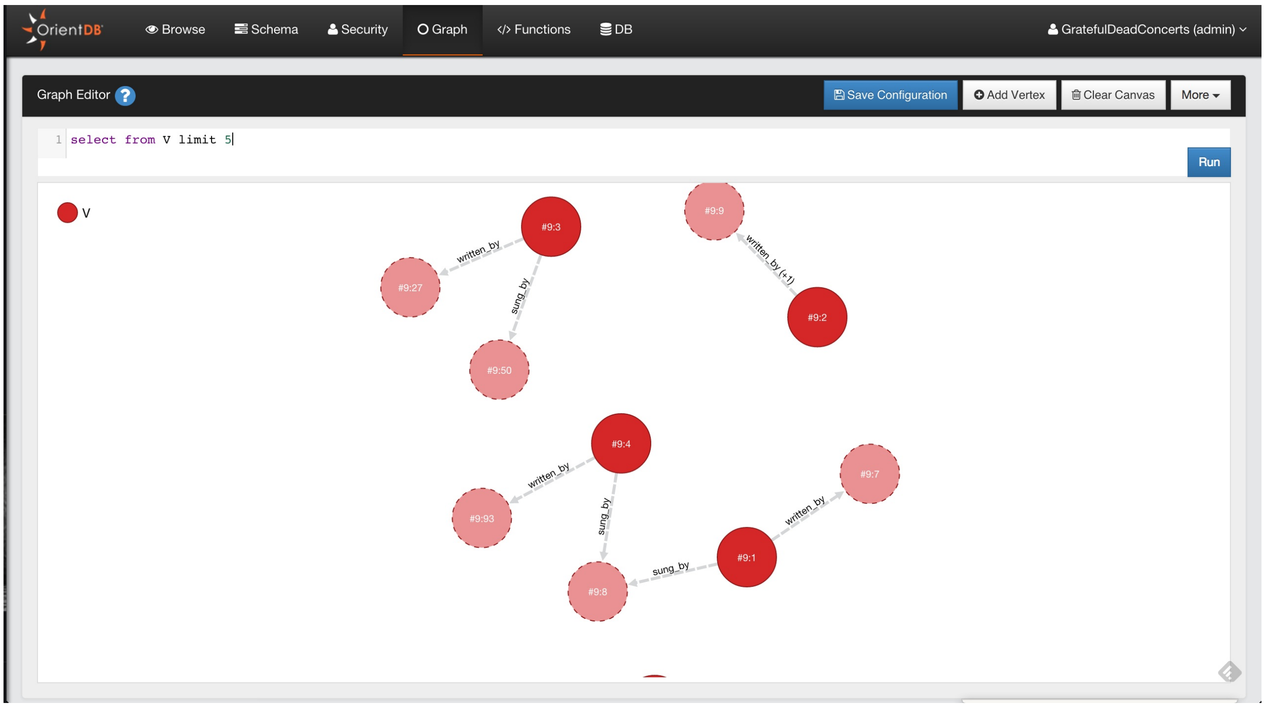
Name	Type	Linked_Type	Max	Collate
bar	STRING			default

Name	Type	Properties	Engine
Foo.bar	NOTUNIQUE	["bar"]	SBTREE

Graph Editor

Since Studio 2.0 we have a new brand graph editor. Not only you can visualize your data in a graph way but you can also interact with the graph and modify it.

To populate the graph area just type a query in the query editor or use the functionality **Send To Graph** from the Browse UI



```
select from V limit 5
```

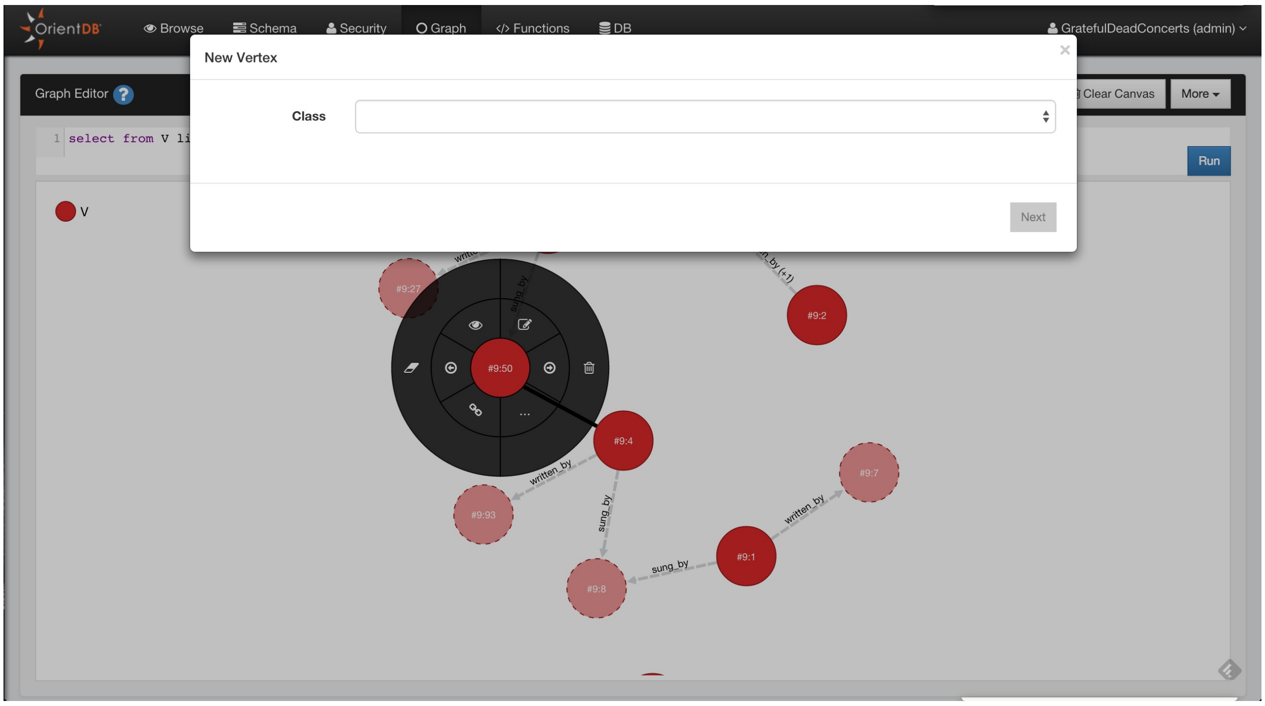
Supported operations in the Graph Editor are:

- Add Vertices
- Save the Graph Rendering Configuration
- Clear the Graph Rendering Canvas
- Delete Vertices
- Remove Vertices from Canvas
- Edit Vertices
- Inspect Vertices
- Change the Rendering Configuration of Vertices
- Navigating Relationships
- Create Edges between Vertices
- Delete Edges between Vertices
- Inspect Edges
- Edit Edges

Add Vertices

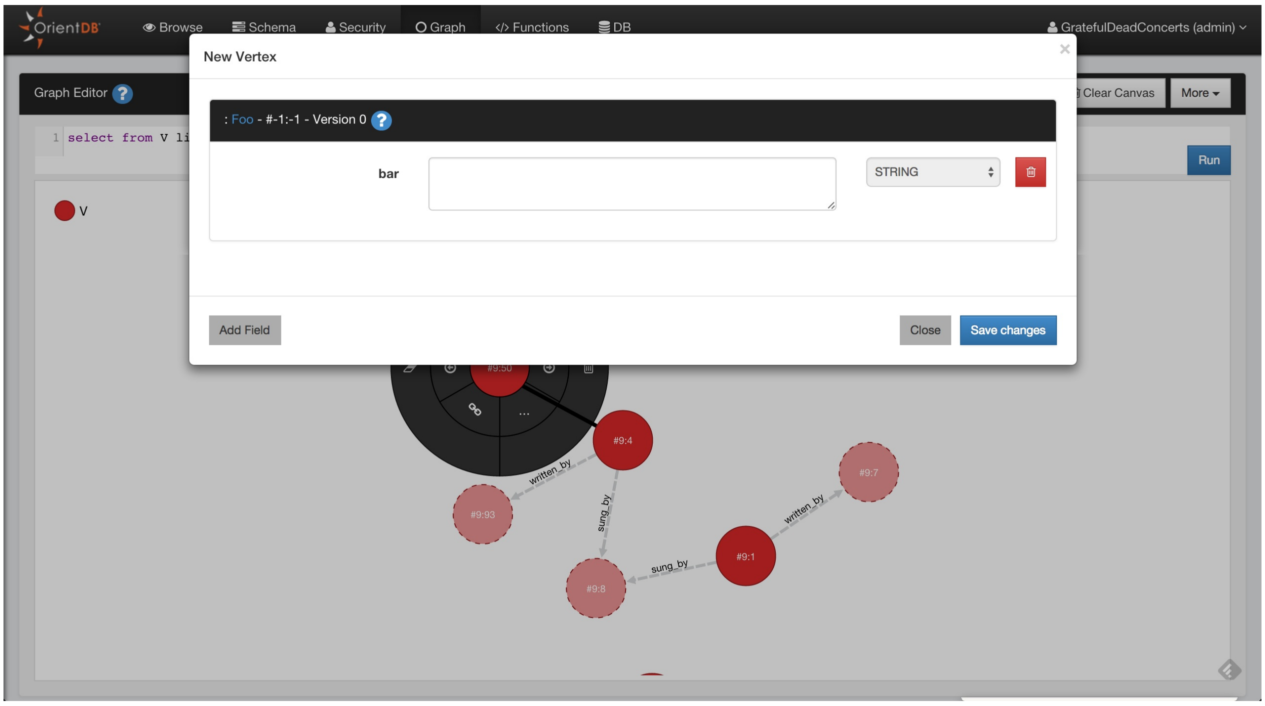
To add a new Vertex in your Graph Database and in the Graph Canvas area you have to press the button **Add Vertex**. This operation is done in two steps.

The first step you have to choose the class for the new Vertex and then click **Next**



```
select from V li
```

In the second step you have to insert the fields values of the new vertex, you can also add custom fields as OrientDB supports Schema-Less mode. To make the new vertex persistent click to **Save changes** and the vertex will be saved into the database and added to the canvas area



```
select from V li
```

Delete Vertices

Open the circular menu by clicking on the Vertex that you want to delete, open the sub-menu by passing hover the mouse to the menu entry more (...) and then click the trash icon.

Remove Vertices from Canvas

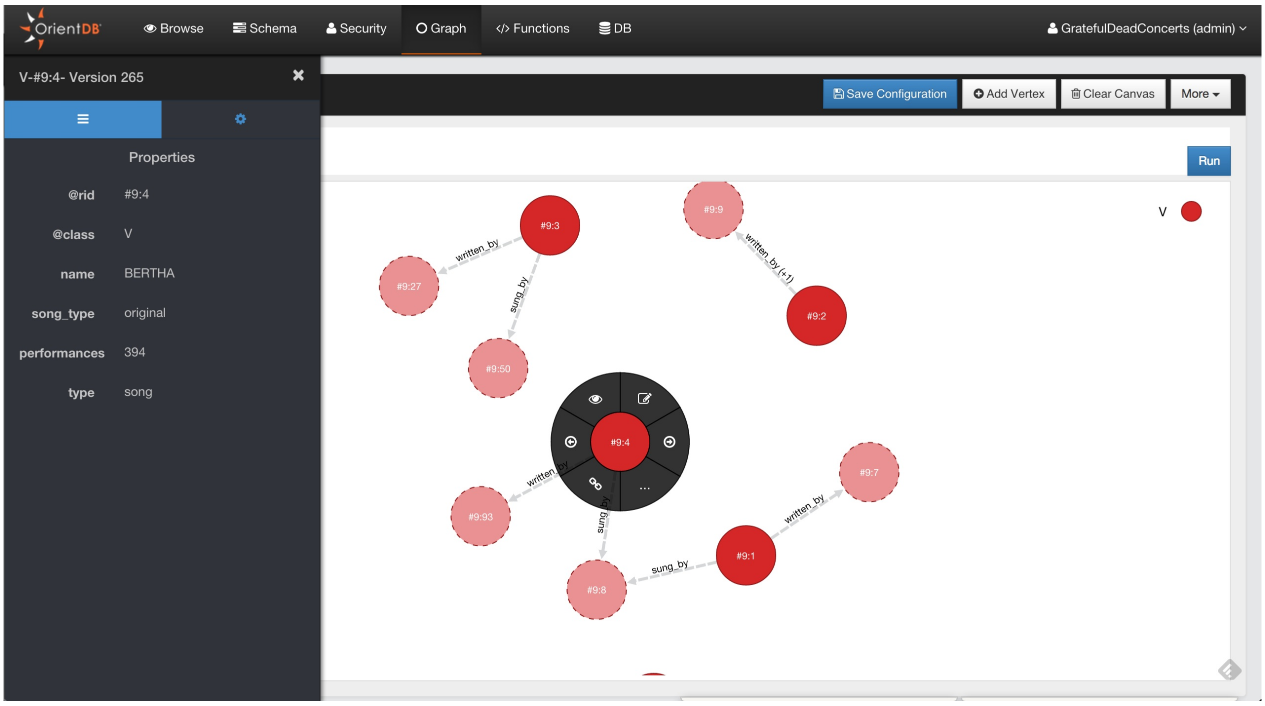
Open the circular menu , open the sub-menu by passing hover the mouse to the menu entry more (...) and then click the **eraser** icon.

Edit Vertices

Open the circular menu and then click to the **edit** icon, Studio will open a popup where you can edit the vertex properties.

Inspect Vertices

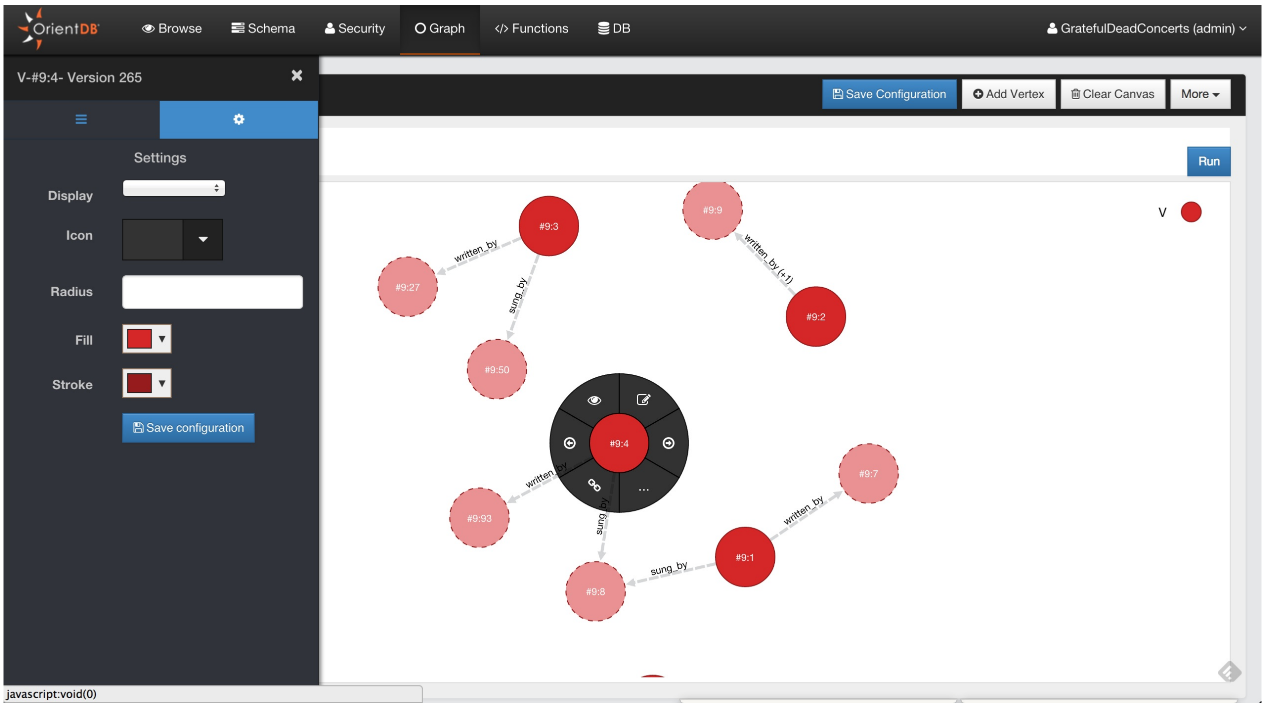
If you want to take a quick look to the Vertex property, click to the **eye** icon.



Properties

@rid	#9:4
@class	V
name	BERTHA
song_type	original
performances	394
type	song

Change the Rendering Configuration of Vertices



Navigating Relationships

Create Edges between Vertices

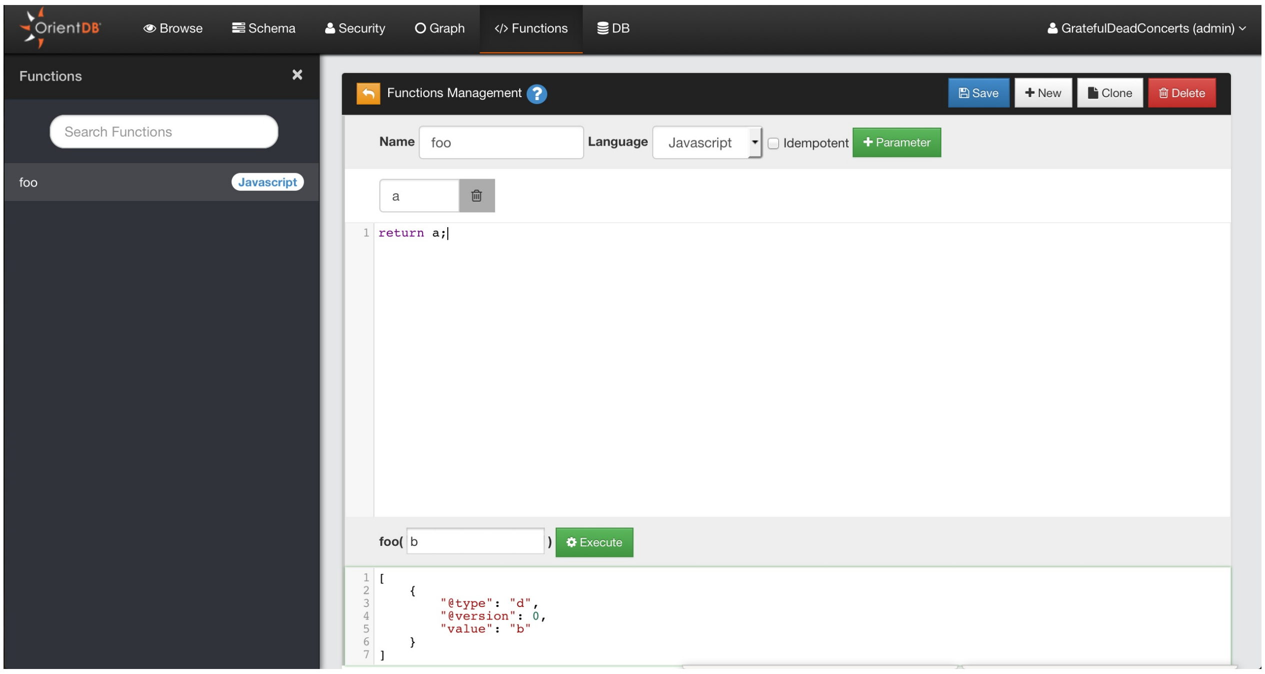
Delete Edges between Vertices

Inspect Edges

Edit Edges

Functions

OrientDB allows to extend the SQL language by providing Functions. Functions can be used also to create data-driven micro services. For more information look at Functions.



```
return a;
```

```
[
    {
        "@type": "d",
        "@version": 0,
        "value": "b"
    }
]
```

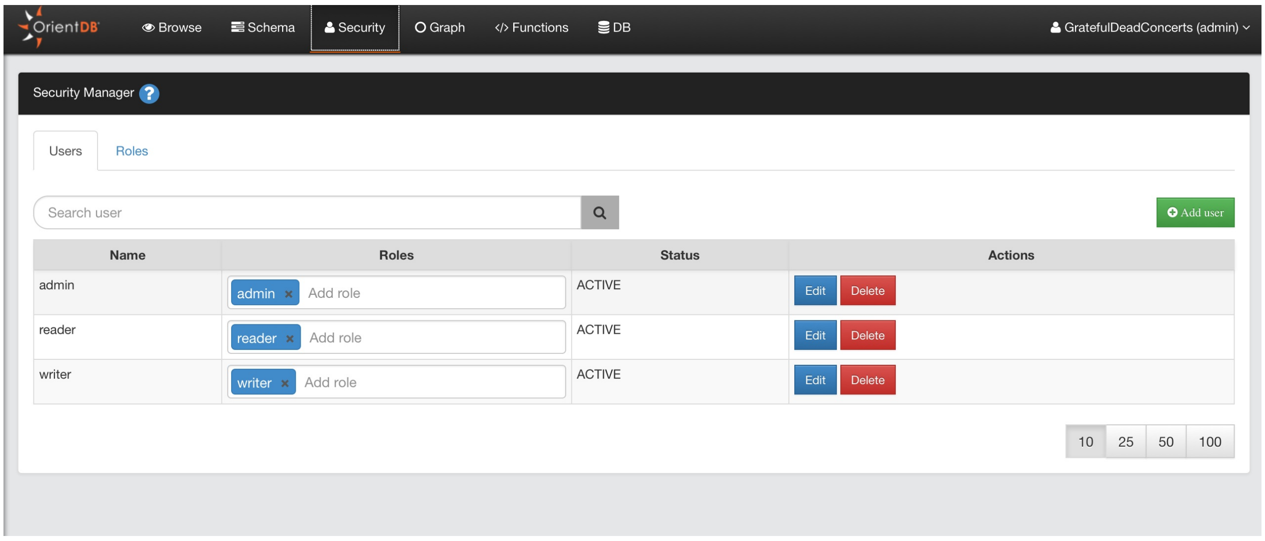
Security

Studio 2.0 includes the new Security Management where you can manage Users and Roles in a graphical way. For detailed information about Security in OrientDB, visit here

Users

Here you can manage the database users:

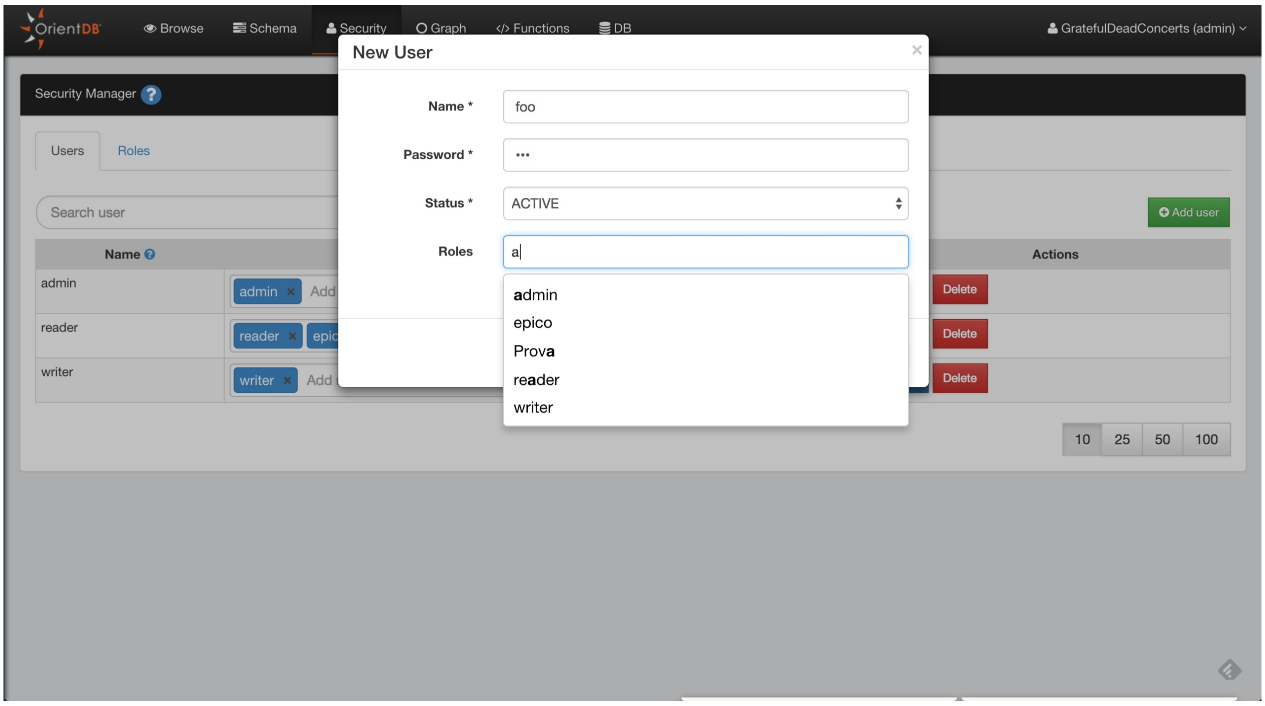
- Search Users
- Add Users
- Delete Users
- Edit User: roles can be edited in-line, for name, status and password click the **Edit** button



Name	Roles	Status	Actions
admin	admin × Add role	ACTIVE	Edit Delete
reader	reader × Add role	ACTIVE	Edit Delete
writer	writer × Add role	ACTIVE	Edit Delete

Add Users

To add a new User, click the **Add User** button, complete the information for the new user (name, password, status, roles) and then save to add the new user to the database.



New User

Name * foo

Password * •••

Status * ACTIVE

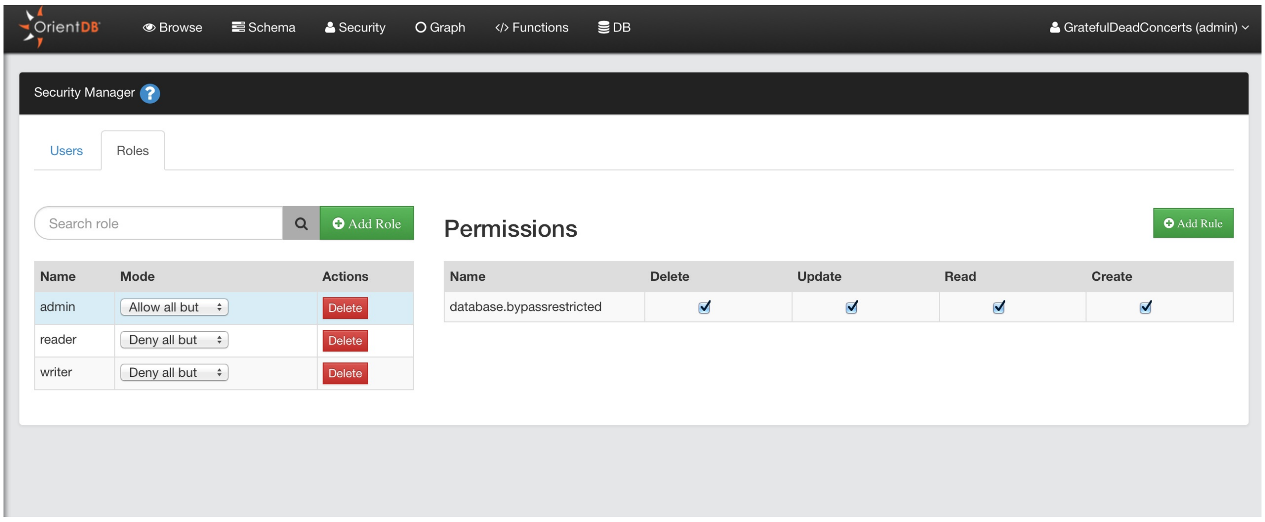
Roles a

admin
epico
Prova
reader
writer

Roles

Here you can manage the database roles:

- Search Role
- Add Role
- Delete Role
- Edit Role

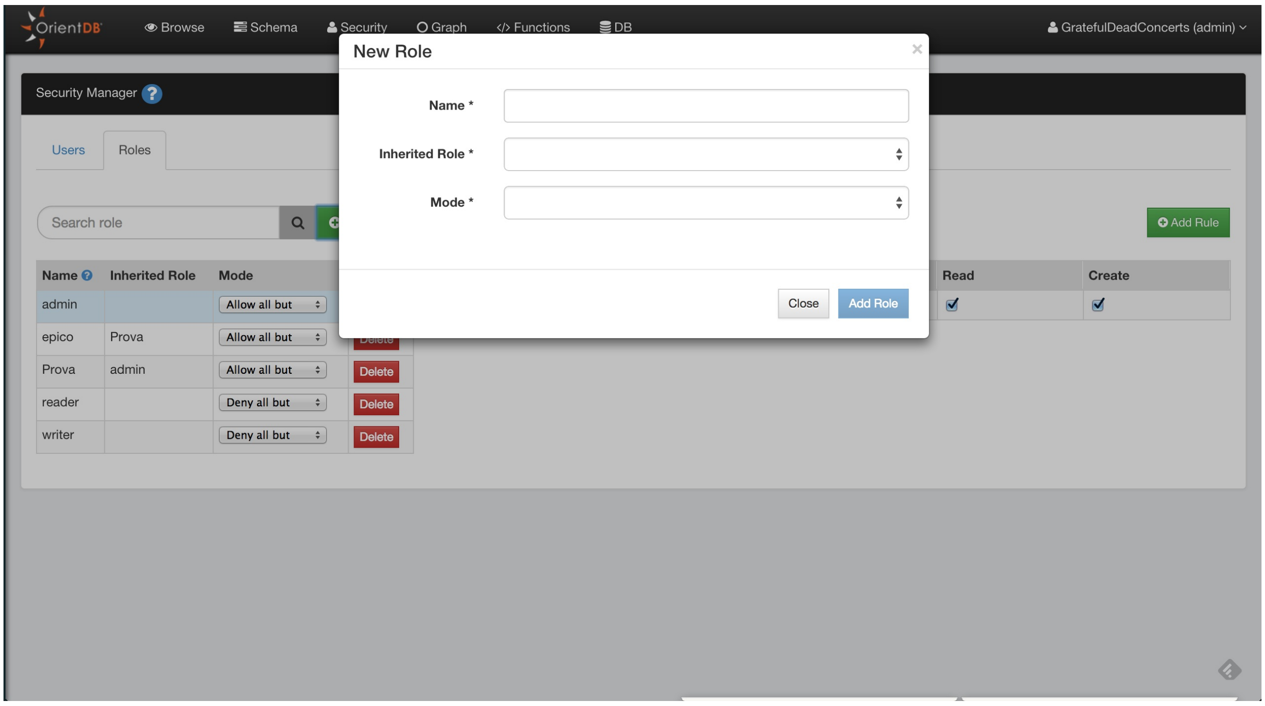


Name	Mode	Actions
admin	Allow all but	Delete
reader	Deny all but	Delete
writer	Deny all but	Delete

Name	Delete	Update	Read	Create
database.bypassrestricted	✓	✓	✓	✓

Add Role

To add a new User, click the **Add Role** button, complete the information for the new role (name, parent role, mode) and then save to add the new role to the database.



New Role

Name *

Inherited Role *

Mode *

Close Add Role

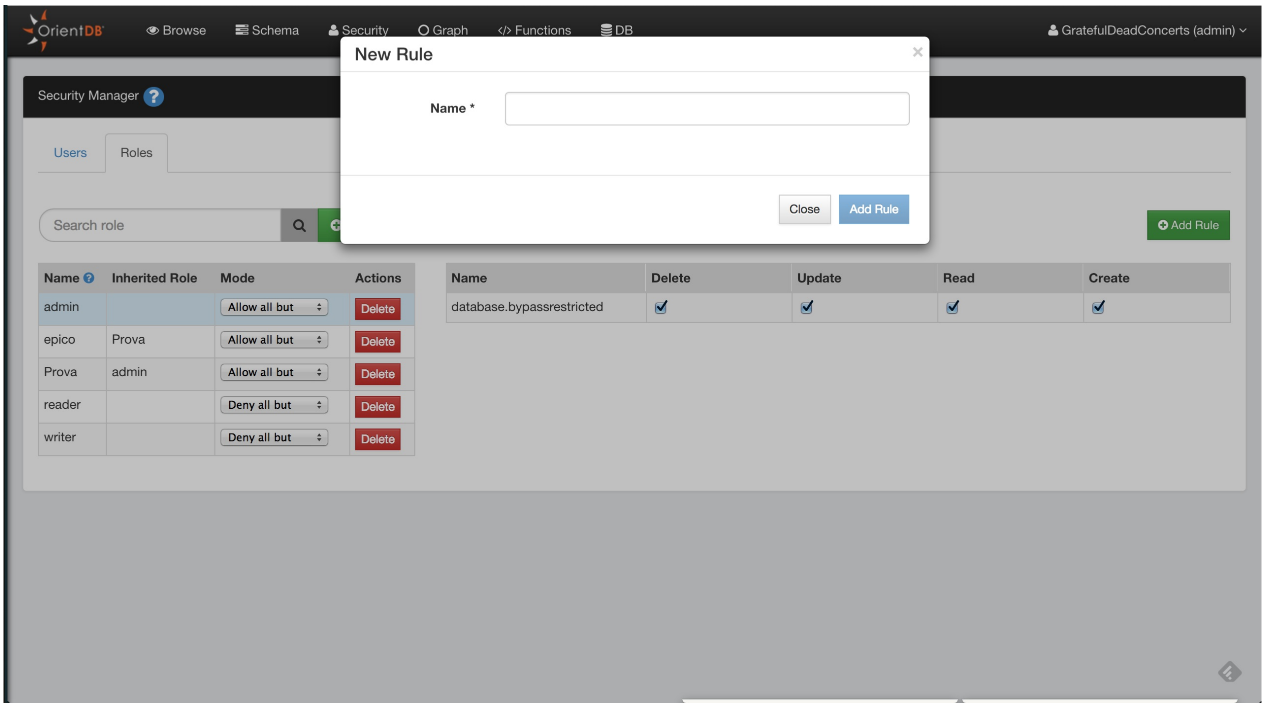
Name	Inherited Role	Mode
admin		Allow all but
epico	Prova	Allow all but
Prova	admin	Allow all but
reader		Deny all but
writer		Deny all but

Read	Create
✓	✓

Add Rule to a Role

To add a new security rule for the selected role, click the *Add Rule* button. This will ask you the string of the resource that you want to secure. For a list of available resources, visit the official documentation here

Then you can configure the CRUD permissions on the newly created resource.



Name	Inherited Role	Mode	Actions
admin		Allow all but	Delete
epico	Prova	Allow all but	Delete
Prova	admin	Allow all but	Delete
reader		Deny all but	Delete
writer		Deny all but	Delete

Name	Delete	Update	Read	Create
database.bypassrestricted	✓	✓	✓	✓

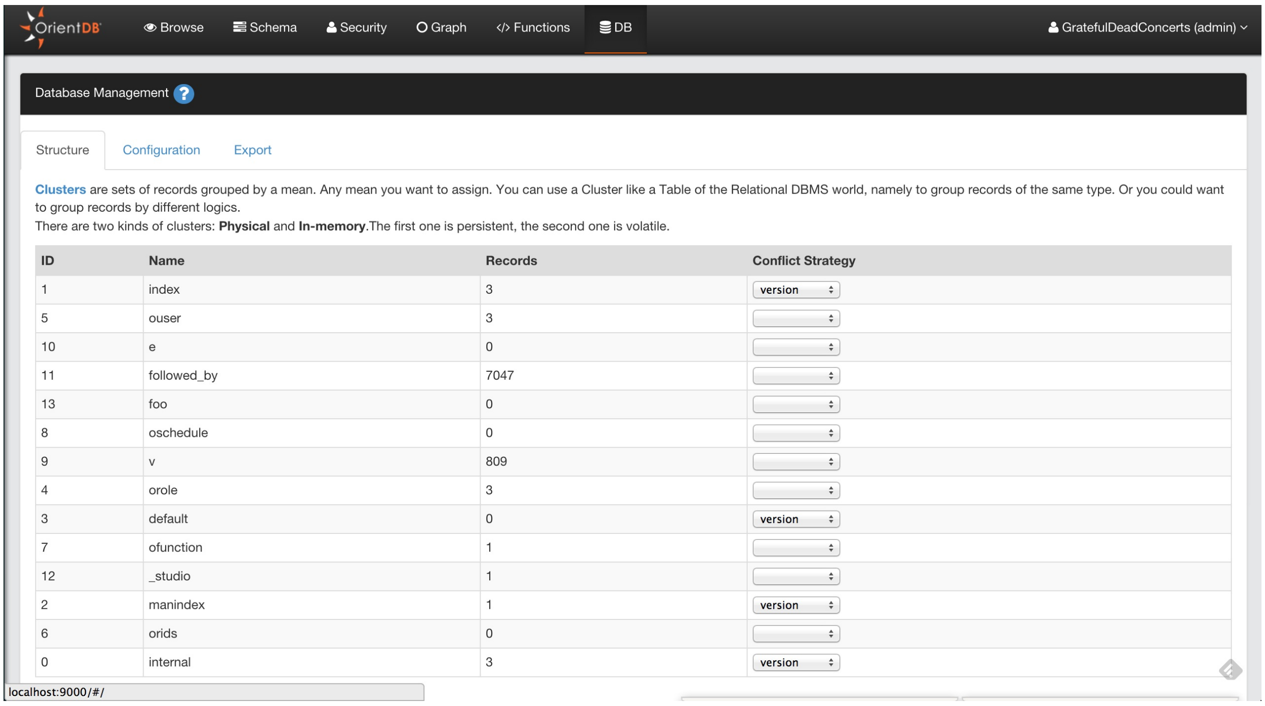
Database Management

This is the panel containing all the information about the current database.

Structure

Represents the database structure as clusters. Each cluster has the following information:

- `ID` , is the cluster ID
- `Name` , is the name of the cluster
- `Records` , are the total number of records stored in the cluster
- `Conflict Strategy` , is the conflict strategy used. I empty, the database's strategy is used as default



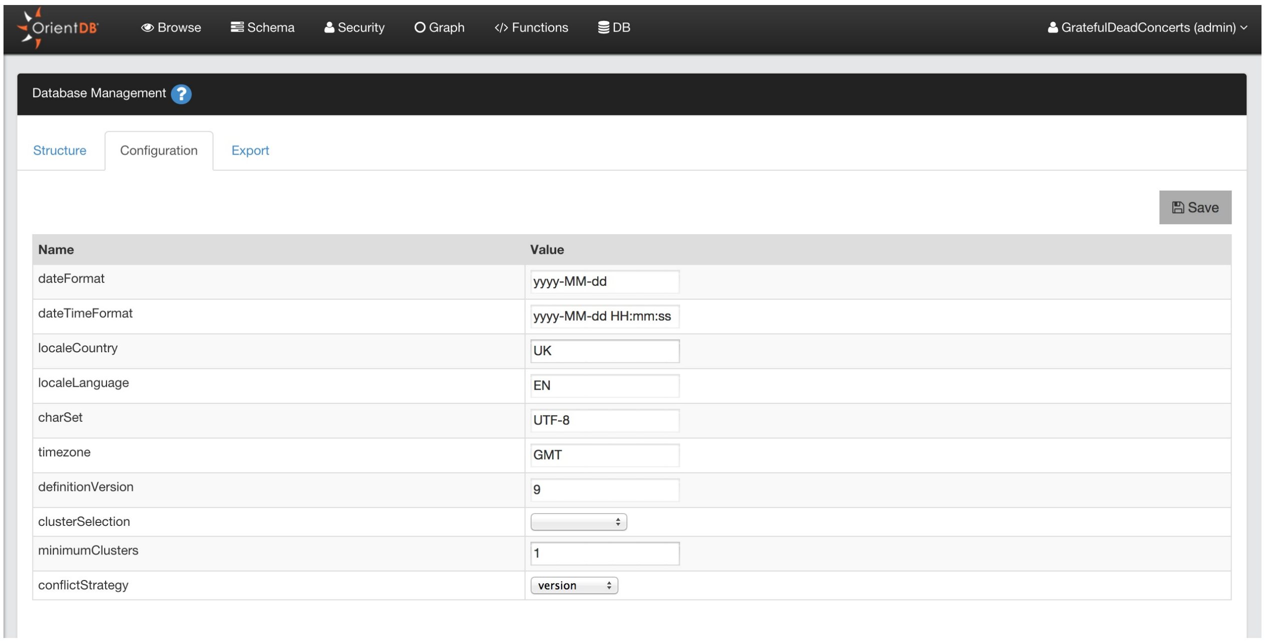
Clusters are sets of records grouped by a mean. Any mean you want to assign. You can use a Cluster like a Table of the Relational DBMS world, namely to group records of the same type. Or you could want to group records by different logics.
There are two kinds of clusters: **Physical** and **In-memory**.The first one is persistent, the second one is volatile.

ID	Name	Records	Conflict Strategy
1	index	3	version
5	ouser	3	
10	e	0	
11	followed_by	7047	
13	foo	0	
8	oschedule	0	
9	v	809	
4	orole	3	
3	default	0	version
7	ofunction	1	
12	_studio	1	
2	manindex	1	version
6	orids	0	
0	internal	3	version

Configuration

Contains the database configuration and custom properties. Here you can display and change the following settings:

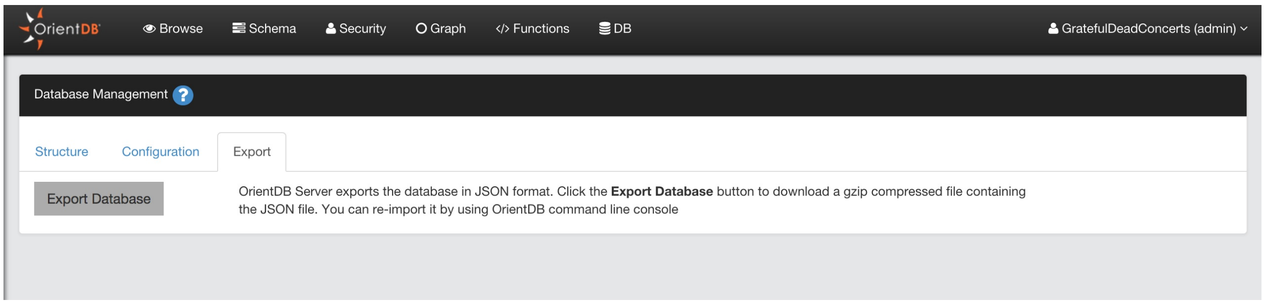
- `dateFormat` , is the date format used in the database by default. Example: yyyy-MM-dd
- `dateTimeFormat` is the datetime format used in the database by default. Example: yyyy-MM-dd HH:mm:ss
- `localeCountry` , is the country used. "NO" means no country set
- `localeLanguage` , is the language used. "no" means no language set
- `charSet` , is the charset used. Default is *UTF-8*
- `timezone` , is the timezone used. Timezone is taken on database creation
- `definitionVersion` , is the internal version used to store the metadata
- `clusterSelection` , is the strategy used on selecting the cluster on creation of new record of a class
- `minimumClusters` , minimum number of clusters to create whenat class creation
- `conflictStrategy` , is the database strategy for resolving conflicts



Name	Value
dateFormat	yyyy-MM-dd
dateTimeFormat	yyyy-MM-dd HH:mm:ss
localeCountry	UK
localeLanguage	EN
charSet	UTF-8
timezone	GMT
definitionVersion	9
clusterSelection	
minimumClusters	1
conflictStrategy	version

Export

Allows to export the current database in GZipped JSON format. To import the file into another database, use the Import Console Command.



OrientDB Server exports the database in JSON format. Click the **Export Database** button to download a gzip compressed file containing the JSON file. You can re-import it by using OrientDB command line console

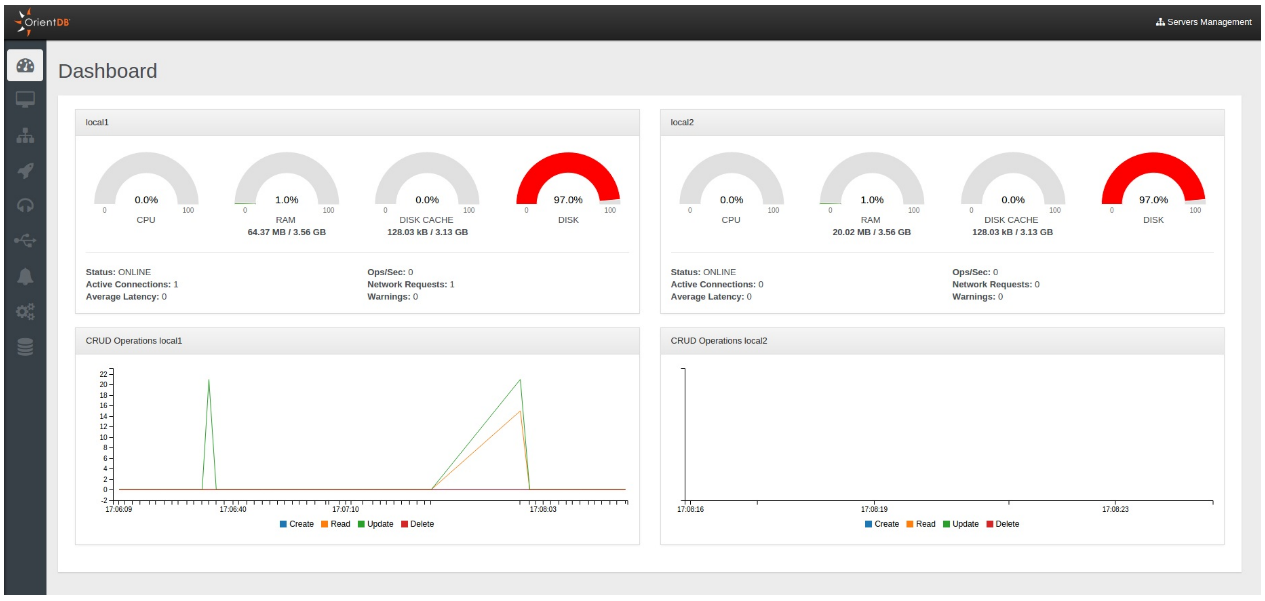
Dashboard

Studio 2.2 Enterprise Edition includes a new easy to read and single-page Dashboard with costantly updated reports. The Dashboard shows a graphical presentation of the current status and historical trends of each node joining your cluster. Performance indicators are reported in order to enable instantaneous and informed decisions which you can make at a glance.

Here you can see the Dashboard reporting the status of a cluster composed of two nodes.

For each node you can monitor several information divided in two main sections:

- System report
  - `CPU` , `RAM` , `DISK CACHE` and `DISK` used
  - `Status` of the node
  - `Operations per second`
  - `Active Connections`
  - `Network Request`
  - `Average Latency`
  - `Warnings`
- CRUD operations: includes a `Live Chart` of CRUD operations in real time.



Status: ONLINE
Active Connections: 1
Average Latency: 0
Ops/Sec: 0
Network Requests: 1
Warnings: 0

Status: ONLINE
Active Connections: 0
Average Latency: 0
Ops/Sec: 0
Network Requests: 0
Warnings: 0

Server Management

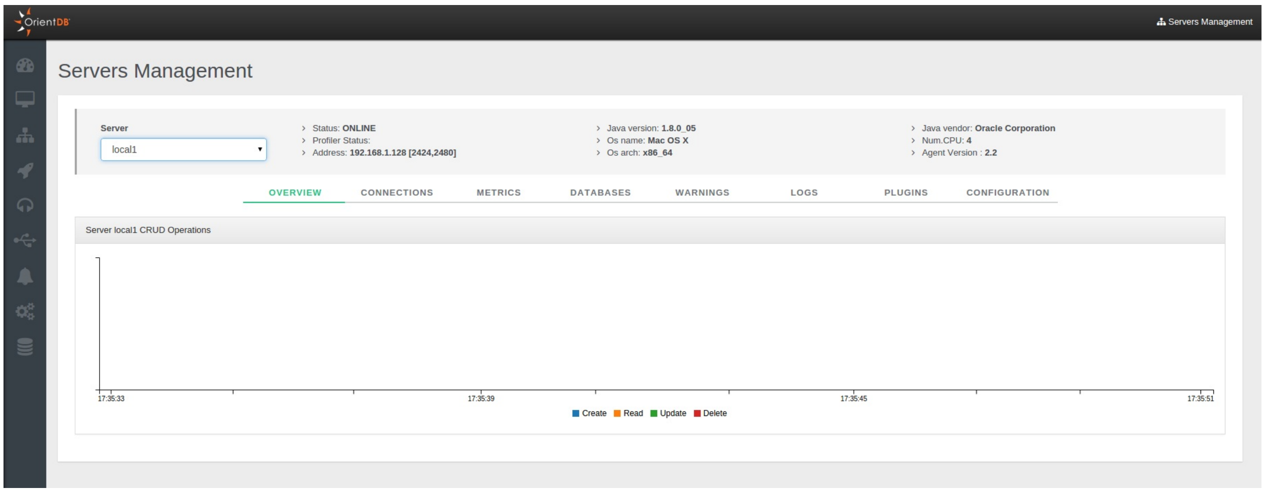
This is the section (available only for the Enterprise Edition) to work with OrientDB Server as DBA/DevOps. This control panel coming from OrientDB 2.1 Studio has been enriched with several new features for the new Enterprise Edition.

On the top of the page you can chose your server, visualize its system information and then navigate all statistics and facts related to it through the available tabs.

Overview

This panel summarizes all the most important information about the current cluster:

- `CPU` , `RAM` , `DISK CACHE` and `DISK` used
- `Status`
- `Operations per second`
- `Active Connections`
- `Warnings`
- `Live chart` with CRUD operations in real-time



Server: local1

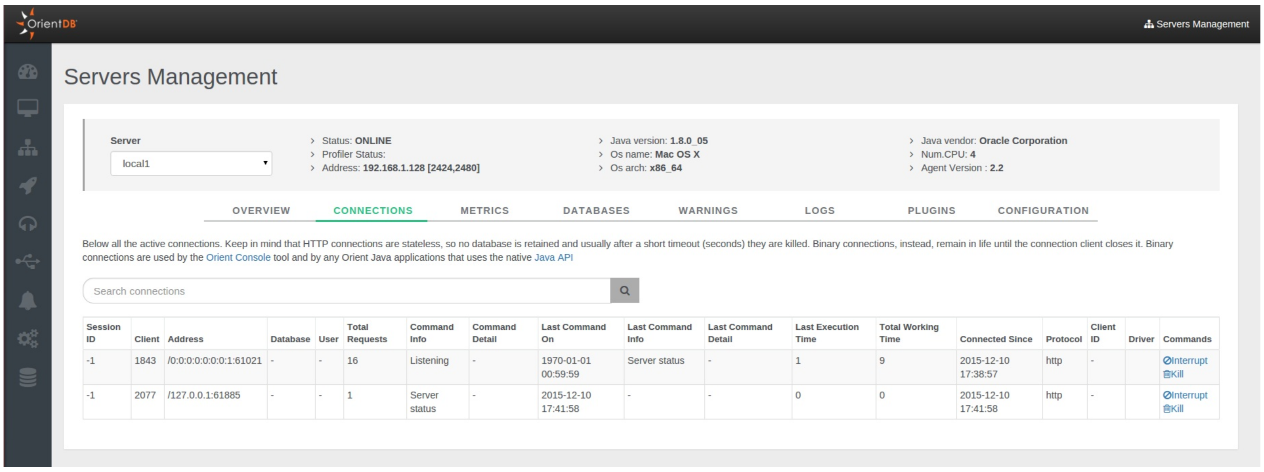
- Status: **ONLINE**
- Profiler Status:
- Address: **192.168.1.128 [2424,2480]**
- Java version: **1.8.0_05**
- Os name: **Mac OS X**
- Os arch: **x86_64**
- Java vendor: **Oracle Corporation**
- Num.CPU: **4**
- Agent Version : **2.2**

Connections

It displays all the active connections to the server. For each connection reports the following information:

- `Session ID` , as the unique session number
- `Client` , as the unique client number
- `Address` , is the connection source
- `Database` , the database name used
- `User` , the database user
- `Total Requests` , as the total number of requests executed by the connection
- `Command Info` , as the running command
- `Command Detail` , as the detail about the running command
- `Last Command On` , is the last time a request has been executed
- `Last Command Info` , is the informaton about last operation executed
- `Last Command Detail` , is the informaton about the details of last operation executed
- `Last Execution Time` , is the execution time o last request
- `Total Working Time` , is the total execution time taken by current connection so far
- `Connected Since` , is the date when the connection has been created
- `Protocol` , is the protocol among HTTP and Binary
- `Client ID` , a text representing the client connection
- `Driver` , the driver name

- `Commands` , a command button to `Interrupt` or `Kill` each session.



Servers Management

Server: local1
- Status: **ONLINE**
- Profiler Status:
- Address: **192.168.1.128 [2424,2480]**
- Java version: **1.8.0_05**
- Os name: **Mac OS X**
- Os arch: **x86_64**
- Java vendor: **Oracle Corporation**
- Num.CPU: **4**
- Agent Version : **2.2**

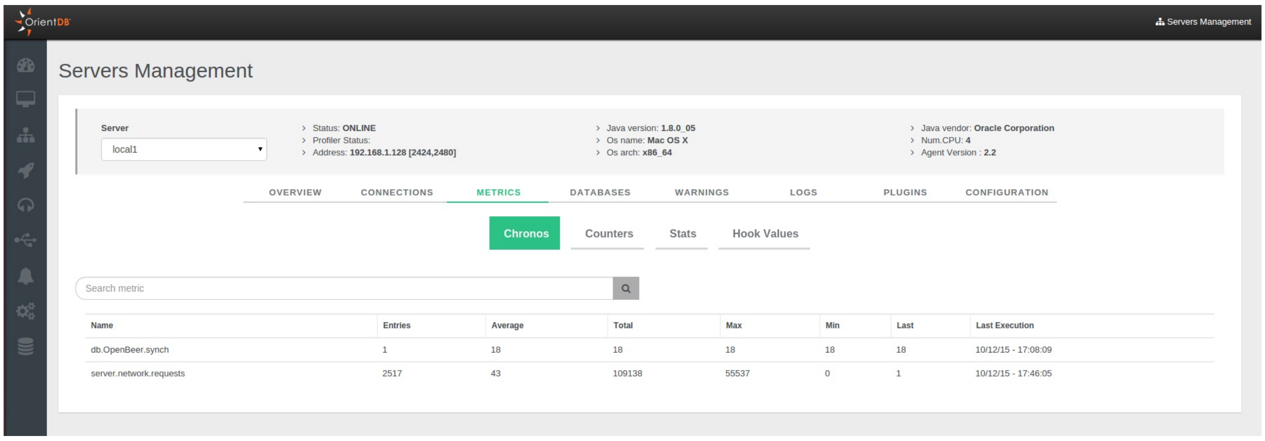
Below all the active connections. Keep in mind that HTTP connections are stateless, so no database is retained and usually after a short timeout (seconds) they are killed. Binary connections, instead, remain in life until the connection client closes it. Binary connections are used by the Orient Console tool and by any Orient Java applications that uses the native Java API

Session ID	Client	Address	Database	User	Total Requests	Command Info	Command Detail	Last Command On	Last Command Info	Last Command Detail	Last Execution Time	Total Working Time	Connected Since	Protocol	Client ID	Driver	Commands
-1	1843	/0:0:0:0:0:0:0:1:61021	-	-	16	Listening	-	1970-01-01 00:59:59	Server status	-	1	9	2015-12-10 17:38:57	http	-		Interrupt Kill
-1	2077	/127.0.0.1:61885	-	-	1	Server status	-	2015-12-10 17:41:58	-	-	0	0	2015-12-10 17:41:58	http	-		Interrupt Kill

Metrics

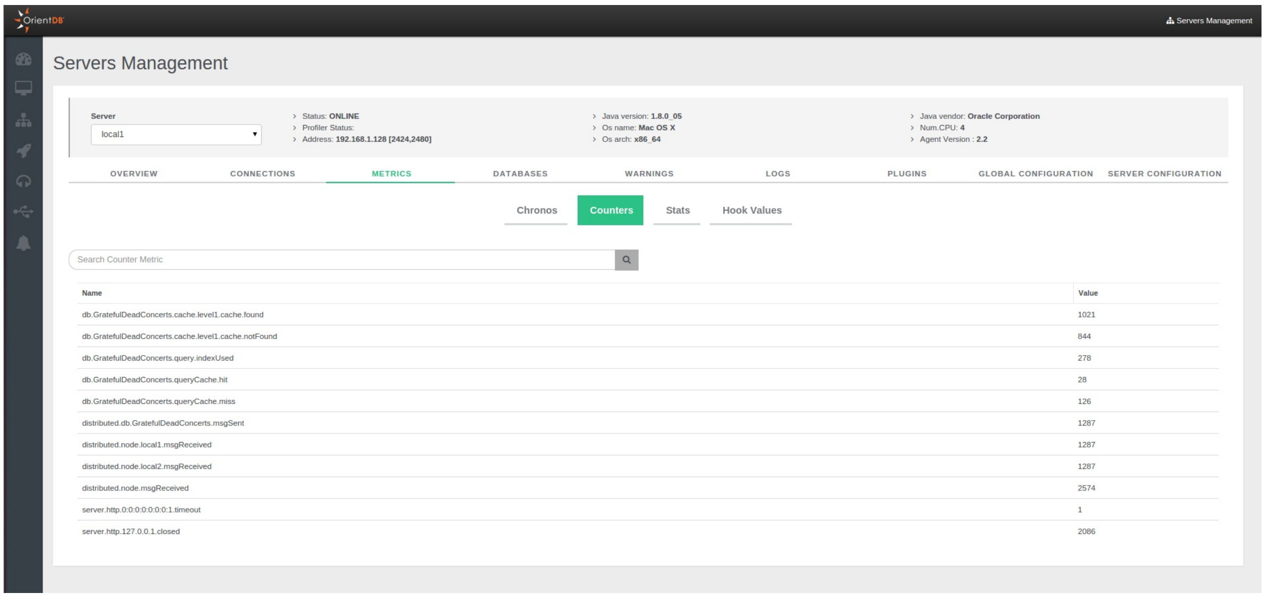
This panel shows all the metrics in 4 different tabs. To learn more about the available metrics please refer to the Profiler section.

- `Chronos`



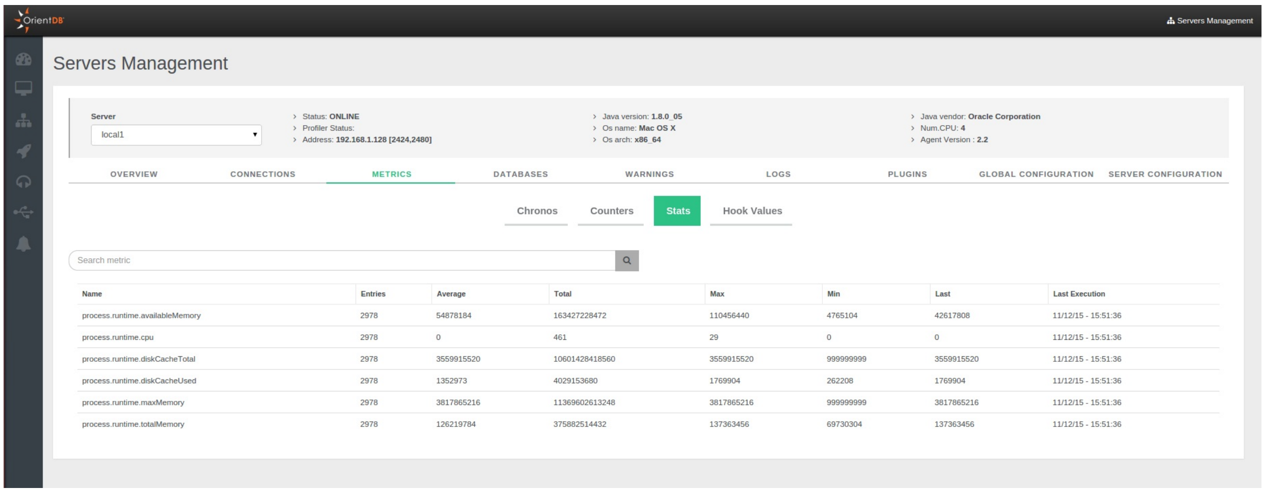
Name	Entries	Average	Total	Max	Min	Last	Last Execution
db.OpenBeer.synch	1	18	18	18	18	18	10/12/15 - 17:08:09
server.network.requests	2517	43	109138	55537	0	1	10/12/15 - 17:46:05

- `Counters`



Name	Value
db.GratefulDeadConcerts.cache.level1.cache.found	1021
db.GratefulDeadConcerts.cache.level1.cache.notFound	844
db.GratefulDeadConcerts.query.indexUsed	278
db.GratefulDeadConcerts.queryCache.hit	28
db.GratefulDeadConcerts.queryCache.miss	126
distributed.db.GratefulDeadConcerts.msgSent	1287
distributed.node.local1.msgReceived	1287
distributed.node.local2.msgReceived	1287
distributed.node.msgReceived	2574
server.http.0:0:0:0:0:0:0:1.timeout	1
server.http.127.0.0.1.closed	2086

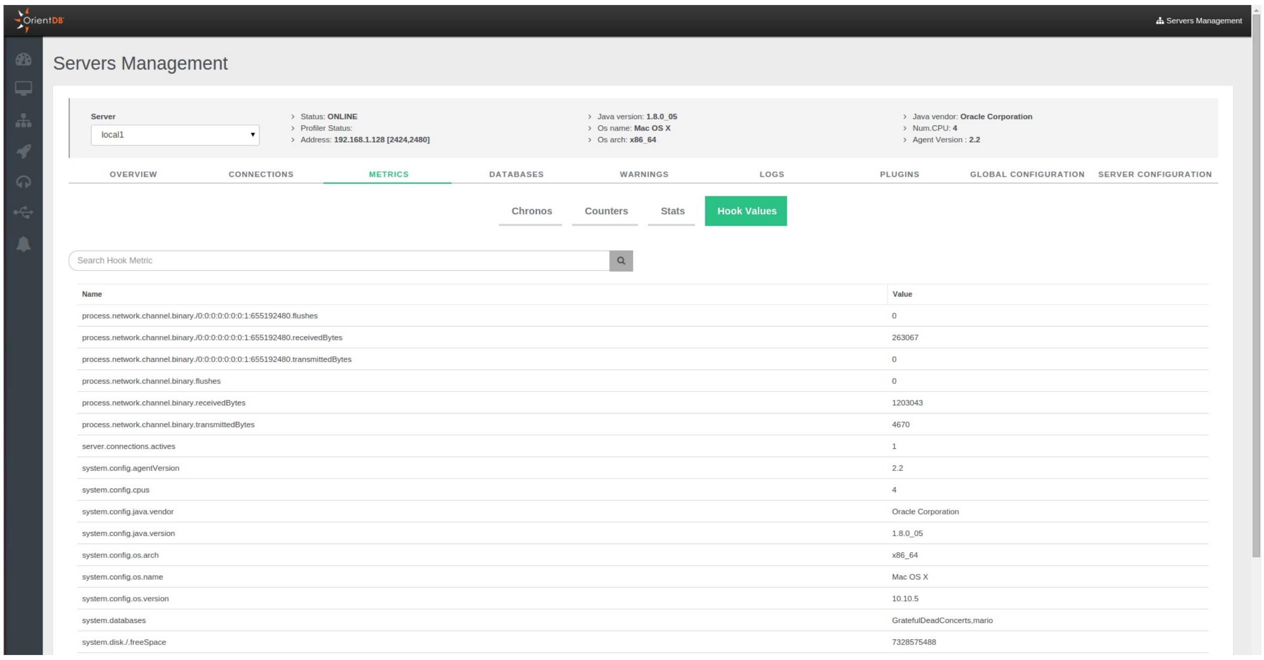
- `Stats`



- Status: **ONLINE**
- Profiler Status:
- Address: **192.168.1.128 [2424,2480]**
- Java version: **1.8.0_05**
- Os name: **Mac OS X**
- Os arch: **x86_64**
- Java vendor: **Oracle Corporation**
- Num.CPU: **4**
- Agent Version : **2.2**

Name	Entries	Average	Total	Max	Min	Last	Last Execution
process.runtime.availableMemory	2978	54878184	163427228472	110456440	4765104	42617808	11/12/15 - 15:51:36
process.runtime.cpu	2978	0	461	29	0	0	11/12/15 - 15:51:36
process.runtime.diskCacheTotal	2978	3559915520	10601428418560	3559915520	999999999	3559915520	11/12/15 - 15:51:36
process.runtime.diskCacheUsed	2978	1352973	4029153680	1769904	262208	1769904	11/12/15 - 15:51:36
process.runtime.maxMemory	2978	3817865216	11369602613248	3817865216	999999999	3817865216	11/12/15 - 15:51:36
process.runtime.totalMemory	2978	126219784	375882514432	137363456	69730304	137363456	11/12/15 - 15:51:36

- `Hook Values`

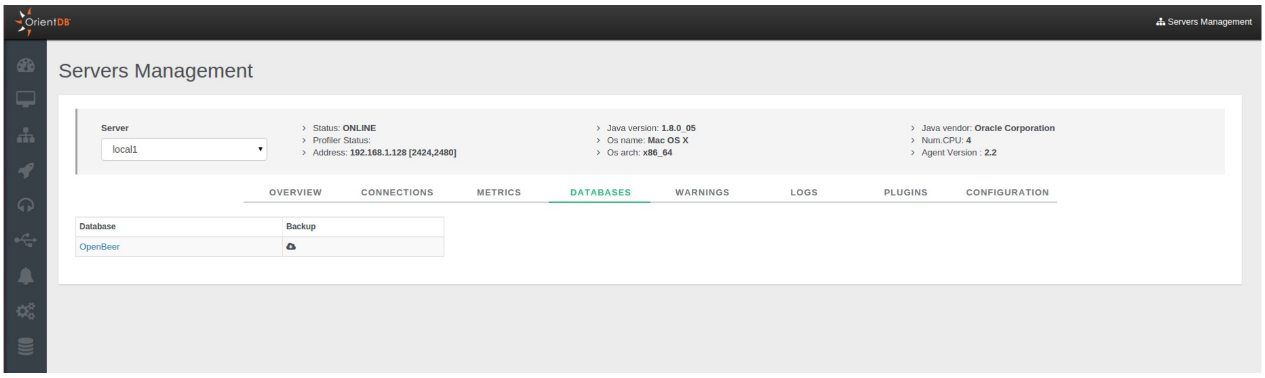


- Status: **ONLINE**
- Profiler Status:
- Address: **192.168.1.128 [2424,2480]**
- Java version: **1.8.0_05**
- Os name: **Mac OS X**
- Os arch: **x86_64**
- Java vendor: **Oracle Corporation**
- Num.CPU: **4**
- Agent Version : **2.2**

Name	Value
process.network.channel.binary./0:0:0:0:0:0:0:1:655192480.flushes	0
process.network.channel.binary./0:0:0:0:0:0:0:1:655192480.receivedBytes	263067
process.network.channel.binary./0:0:0:0:0:0:0:1:655192480.transmittedBytes	0
process.network.channel.binary.flushes	0
process.network.channel.binary.receivedBytes	1203043
process.network.channel.binary.transmittedBytes	4670
server.connections.actives	1
system.config.agentVersion	2.2
system.config.cpus	4
system.config.java.vendor	Oracle Corporation
system.config.java.version	1.8.0_05
system.config.os.arch	x86_64
system.config.os.name	Mac OS X
system.config.os.version	10.10.5
system.databases	GratefulDeadConcerts,mario
system.disk./.freeSpace	7328575488

Databases

It lists all databases created on the server. It is possible make a backup using the specific option.



- Status: **ONLINE**
- Profiler Status:
- Address: **192.168.1.128 [2424,2480]**
- Java version: **1.8.0_05**
- Os name: **Mac OS X**
- Os arch: **x86_64**
- Java vendor: **Oracle Corporation**
- Num.CPU: **4**
- Agent Version : **2.2**

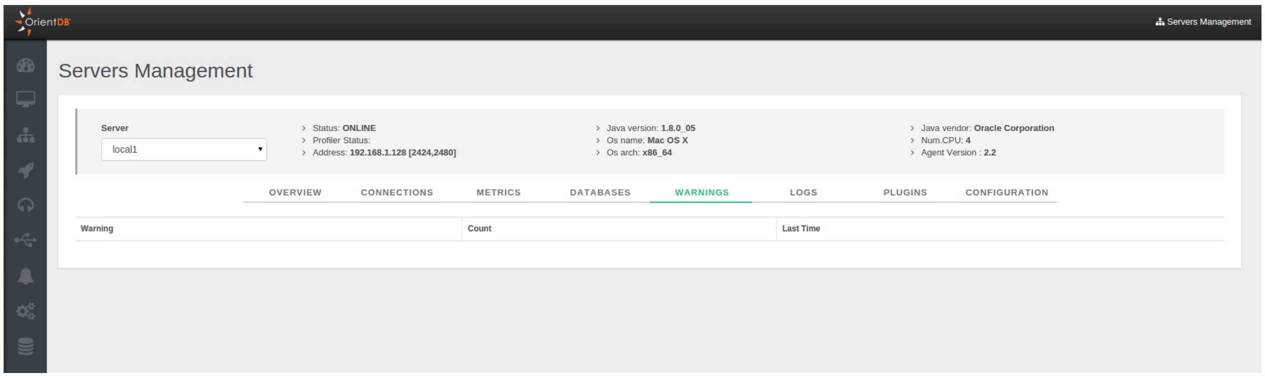
Database	Backup
OpenBeer	☁

Warnings

It list all warning messages. For each you can see:

- `Warning` , as the warning message
- `Count` , as the number of that warnings

- `Last Time`, as the timestamp of the last warning message



Servers Management

Server: local1

Status: **ONLINE**
Profiler Status:
Address: **192.168.1.128 [2424,2480]**

Java version: **1.8.0_05**
Os name: **Mac OS X**
Os arch: **x86_64**

Java vendor: **Oracle Corporation**
Num.CPU: **4**
Agent Version : **2.2**

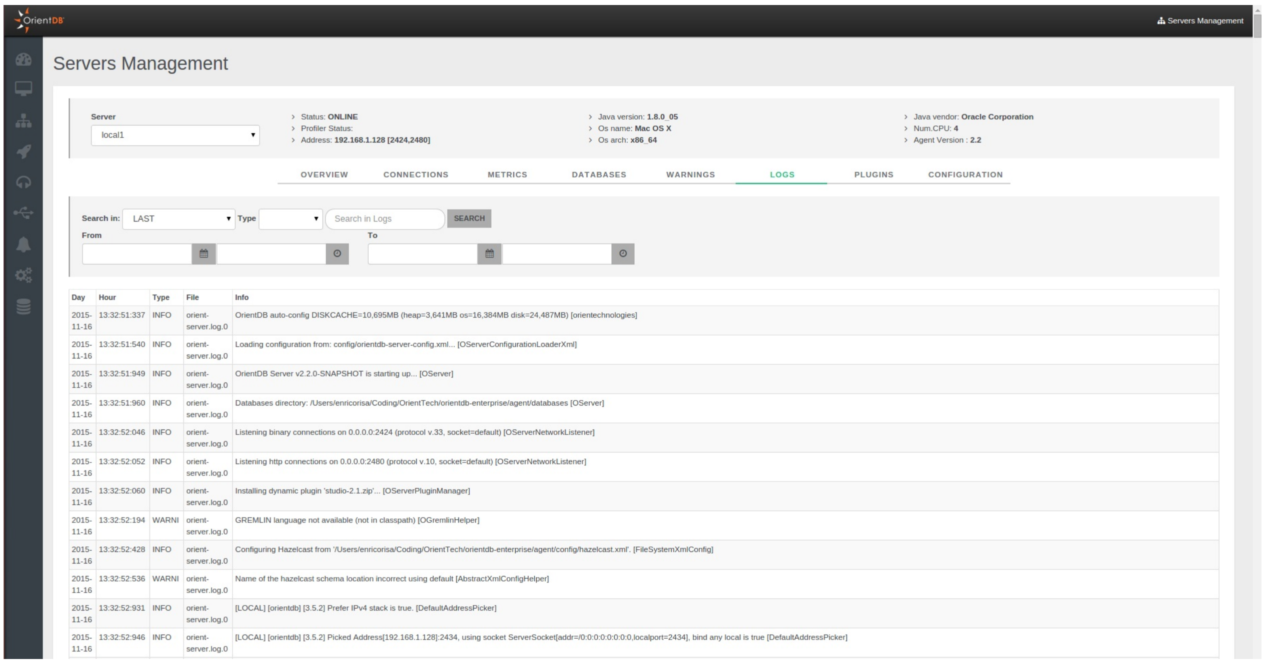
Warning	Count	Last Time

Logs

This panel shows all the logs present on the server. The information in each log row are presented divided as follows:

- `Day`
- `Hour`
- `Type`
- `File`
- `Info`

Moreover you can filter log messages through the specific panel, typing different parameters.



Servers Management

Server: local1

Status: **ONLINE**
Profiler Status:
Address: **192.168.1.128 [2424,2480]**

Java version: **1.8.0_05**
Os name: **Mac OS X**
Os arch: **x86_64**

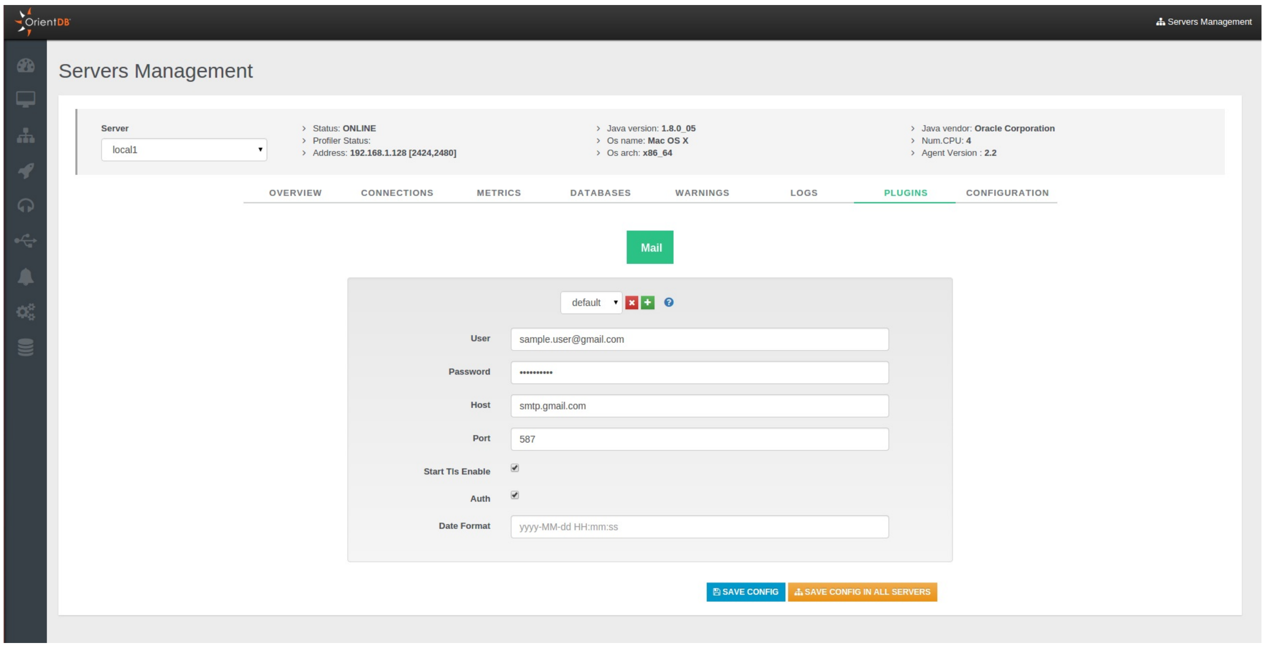
Java vendor: **Oracle Corporation**
Num.CPU: **4**
Agent Version : **2.2**

Search in: LAST

Day	Hour	Type	File	Info
2015-11-16	13:32:51:337	INFO	orient-server.log.0	OrientDB auto-config DISKCACHE=10,695MB (heap=3,641MB os=16,384MB disk=24,487MB) [orientechnologies]
2015-11-16	13:32:51:540	INFO	orient-server.log.0	Loading configuration from: config/orientdb-server-config.xml... [OServerConfigurationLoaderXml]
2015-11-16	13:32:51:949	INFO	orient-server.log.0	OrientDB Server v2.2.0-SNAPSHOT is starting up... [OServer]
2015-11-16	13:32:51:960	INFO	orient-server.log.0	Databases directory: /Users/enricorisa/Coding/OrientTech/orientdb-enterprise/agent/databases [OServer]
2015-11-16	13:32:52:046	INFO	orient-server.log.0	Listening binary connections on 0.0.0.0:2424 (protocol v.33, socket=default) [OServerNetworkListener]
2015-11-16	13:32:52:052	INFO	orient-server.log.0	Listening http connections on 0.0.0.0:2480 (protocol v.10, socket=default) [OServerNetworkListener]
2015-11-16	13:32:52:060	INFO	orient-server.log.0	Installing dynamic plugin 'studio-2.1.zip'... [OServerPluginManager]
2015-11-16	13:32:52:194	WARNI	orient-server.log.0	GREMLIN language not available (not in classpath) [OGremlinHelper]
2015-11-16	13:32:52:428	INFO	orient-server.log.0	Configuring Hazelcast from '/Users/enricorisa/Coding/OrientTech/orientdb-enterprise/agent/config/hazelcast.xml'. [FileSystemXmlConfig]
2015-11-16	13:32:52:536	WARNI	orient-server.log.0	Name of the hazelcast schema location incorrect using default [AbstractXmlConfigHelper]
2015-11-16	13:32:52:931	INFO	orient-server.log.0	[LOCAL] [orientdb] [3.5.2] Prefer IPv4 stack is true. [DefaultAddressPicker]
2015-11-16	13:32:52:946	INFO	orient-server.log.0	[LOCAL] [orientdb] [3.5.2] Picked Address[192.168.1.128]:2434, using socket ServerSocket[addr=/0:0:0:0:0:0:0:0,localport=2434], bind any local is true [DefaultAddressPicker]

Plugins

It helps you with the configuration of a new plugin, avoiding to edit the `config/orientdb-server-config.xml` configuration file.



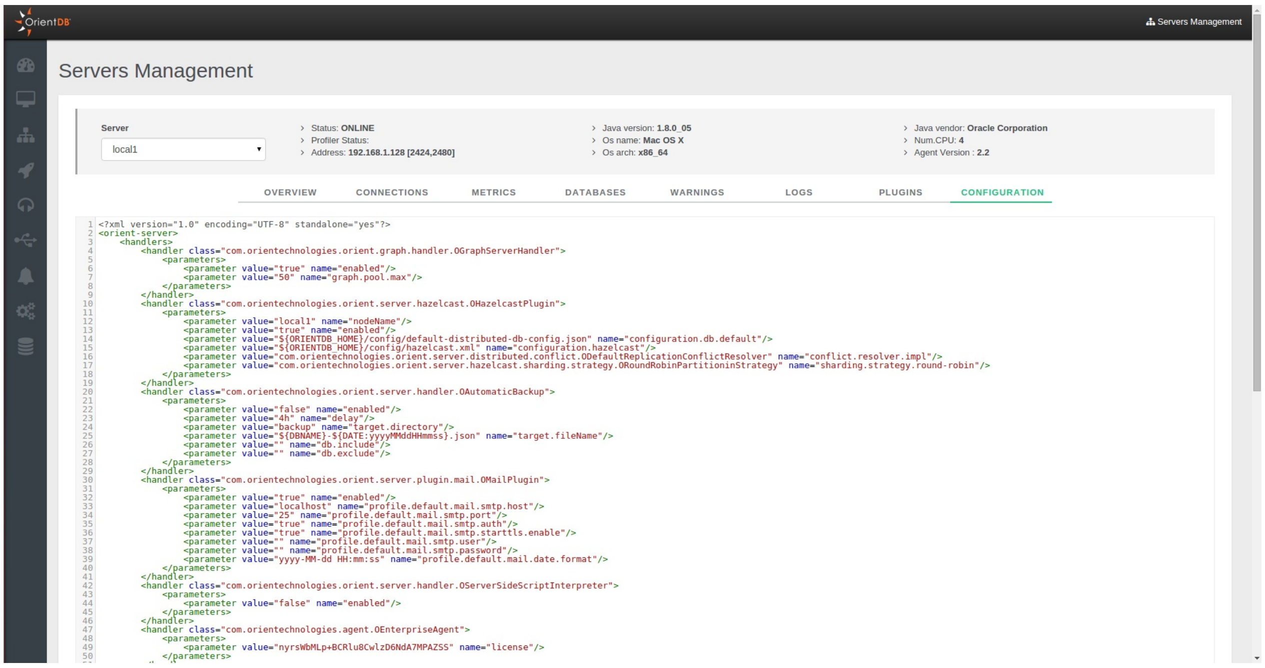
Server: local1

- Status: **ONLINE**
- Profiler Status:
- Address: **192.168.1.128 [2424,2480]**
- Java version: **1.8.0_05**
- Os name: **Mac OS X**
- Os arch: **x86_64**
- Java vendor: **Oracle Corporation**
- Num.CPU: **4**
- Agent Version : **2.2**

	default
User	sample.user@gmail.com
Password	••••••••••
Host	smtp.gmail.com
Port	587
Start Tls Enable	✓
Auth	✓
Date Format	yyyy-MM-dd HH:mm:ss

Configuration

You can consult in read-only mode the configuration of the server contained in the `config/orientdb-server-config.xml` file.



Server: local1

- Status: **ONLINE**
- Profiler Status:
- Address: **192.168.1.128 [2424,2480]**
- Java version: **1.8.0_05**
- Os name: **Mac OS X**
- Os arch: **x86_64**
- Java vendor: **Oracle Corporation**
- Num.CPU: **4**
- Agent Version : **2.2**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<orient-server>
    <handlers>
        <handler class="com.orientechnologies.orient.graph.handler.OGraphServerHandler">
            <parameters>
                <parameter value="true" name="enabled"/>
                <parameter value="50" name="graph.pool.max"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.orient.server.hazelcast.OHazelcastPlugin">
            <parameters>
                <parameter value="local1" name="nodeName"/>
                <parameter value="true" name="enabled"/>
                <parameter value="${ORIENTDB_HOME}/config/default-distributed-db-config.json" name="configuration.db.default"/>
                <parameter value="${ORIENTDB_HOME}/config/hazelcast.xml" name="configuration.hazelcast"/>
                <parameter value="com.orientechnologies.orient.server.distributed.conflict.ODefaultReplicationConflictResolver" name="conflict.resolver.impl"/>
                <parameter value="com.orientechnologies.orient.server.hazelcast.sharding.strategy.ORoundRobinPartitioninStrategy" name="sharding.strategy.round-robin"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.orient.server.handler.OAutomaticBackup">
            <parameters>
                <parameter value="false" name="enabled"/>
                <parameter value="4h" name="delay"/>
                <parameter value="backup" name="target.directory"/>
                <parameter value="${DBNAME}-${DATE:yyyyMMddHHmmss}.json" name="target.fileName"/>
                <parameter value="" name="db.include"/>
                <parameter value="" name="db.exclude"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.orient.server.plugin.mail.OMailPlugin">
            <parameters>
                <parameter value="true" name="enabled"/>
                <parameter value="localhost" name="profile.default.mail.smtp.host"/>
                <parameter value="25" name="profile.default.mail.smtp.port"/>
                <parameter value="true" name="profile.default.mail.smtp.auth"/>
                <parameter value="true" name="profile.default.mail.smtp.starttls.enable"/>
                <parameter value="" name="profile.default.mail.smtp.user"/>
                <parameter value="" name="profile.default.mail.smtp.password"/>
                <parameter value="yyyy-MM-dd HH:mm:ss" name="profile.default.mail.date.format"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.orient.server.handler.OServerSideScriptInterpreter">
            <parameters>
                <parameter value="false" name="enabled"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.agent.OEnterpriseAgent">
            <parameters>
                <parameter value="nyrsWbMLp+BCRlu8CwlzD6NdA7MPAZSS" name="license"/>
            </parameters>
```

Cluster Management

This is the section (available only for the Enterprise Edition) to work with OrientDB Cluster as DBA/DevOps.

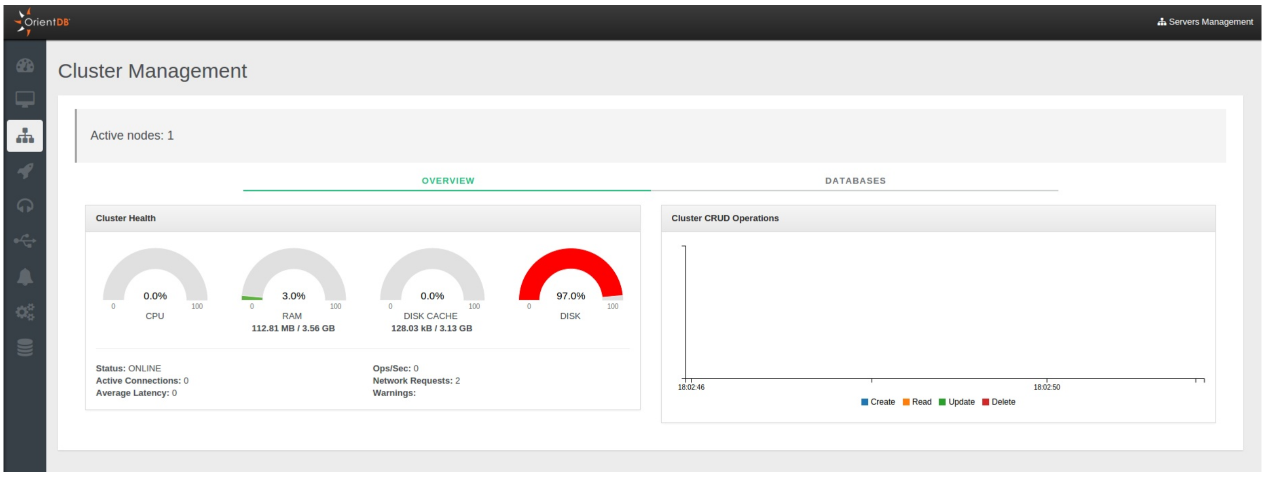
NOTE: *This feature is available only in the OrientDB Enterprise Edition. If you are interested in a commercial license look at OrientDB Subscription Packages*.

On the top of the page are reported the number of active nodes joining your cluster.

Overview

This page summarizes all the most important information about all servers connected to the cluster:

- `CPU` , `RAM` , `DISK CACHE` and `DISK` used
- `Status`
- `Operations per second`
- `Active Connections`
- `Network Requests`
- `Average Latency`
- `Warnings`
- `Live chart` with CRUD operations in real-time



Active nodes: 1

112.81 MB / 3.56 GB
128.03 kB / 3.13 GB

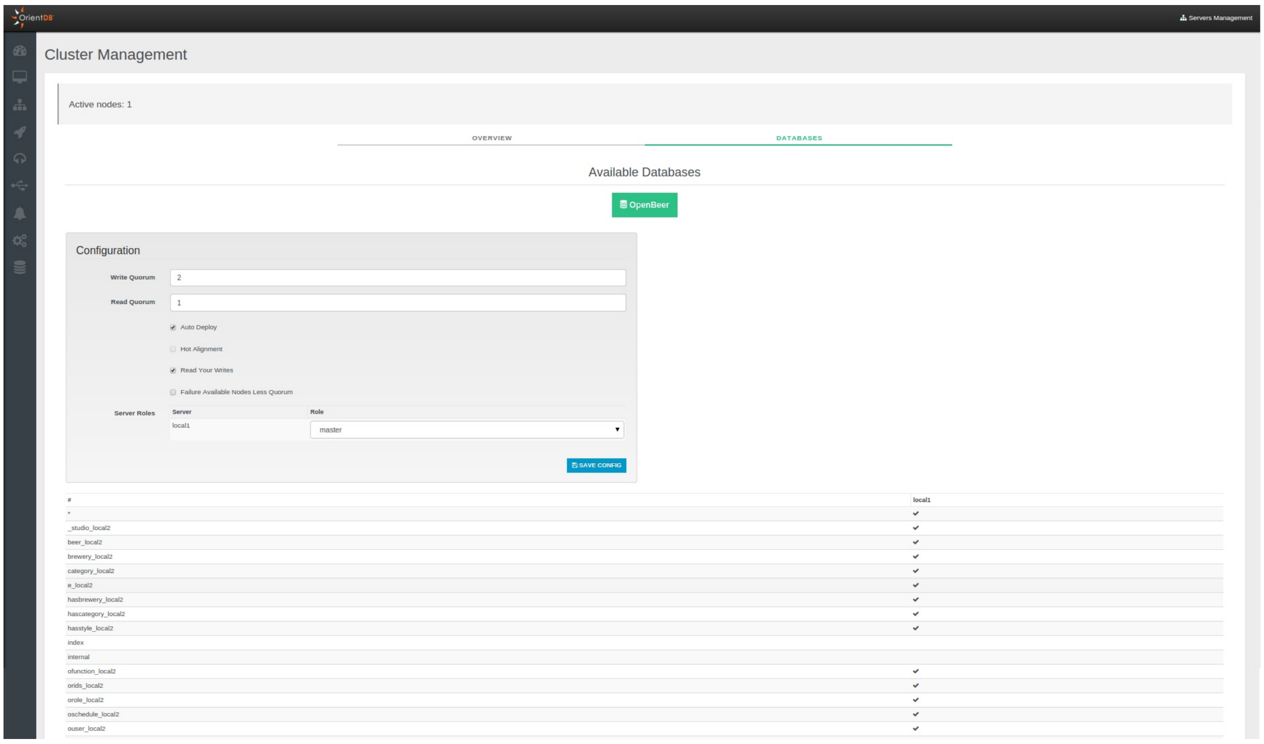
Status: ONLINE
Active Connections: 0
Average Latency: 0
Ops/Sec: 0
Network Requests: 2
Warnings:

Databases

In this panel you can see all databases present on each server joining your cluster. Through the box above you can change in real time the current cluster configuration, without touching the `config/default-distributed-db-config.json` file content. You can set the following parameters:

- `Write Quorum`
- `Read Quorum`
- `Auto Deploy`
- `Hot Alignment`
- `Read your Writes`
- `Failure Available Nodes Less Quorum`
- `Server Roles` , roles may be "Master" or "Replica"

To learn more about these configuration parameters please visit the Distributed Configuration section.



Cluster Management

Active nodes: 1

Available Databases

Configuration

Write Quorum: 2

Read Quorum: 1

[✓] Auto Deploy

[ ] Hot Alignment

[✓] Read Your Writes

[ ] Failure Available Nodes Less Quorum

Server Roles

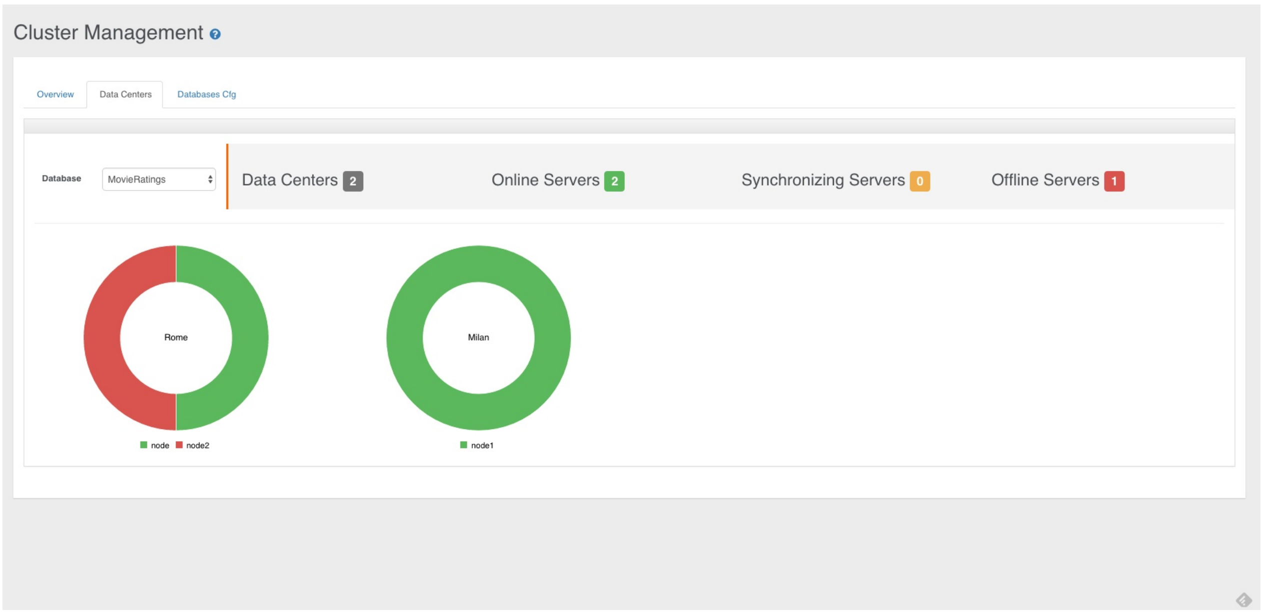
Server	Role
local1	master

#	local1
*	✓
_studio_local2	✓
beer_local2	✓
brewery_local2	✓
category_local2	✓
e_local2	✓
hasbrewery_local2	✓
hascategory_local2	✓
hasstyle_local2	✓
index	
internal	
ofunction_local2	✓
orids_local2	✓
orole_local2	✓
oschedule_local2	✓
ouser_local2	✓

Data Centers

This is the section (available only for the Enterprise Edition) to work with OrientDB Server as DBA/DevOps.

Overview



Query Profiler

Starting from version 2.2, Studio Enterprise Edition includes a functionality called *Profiling*. To understand how Profiling works, please refer to the Profiler page.

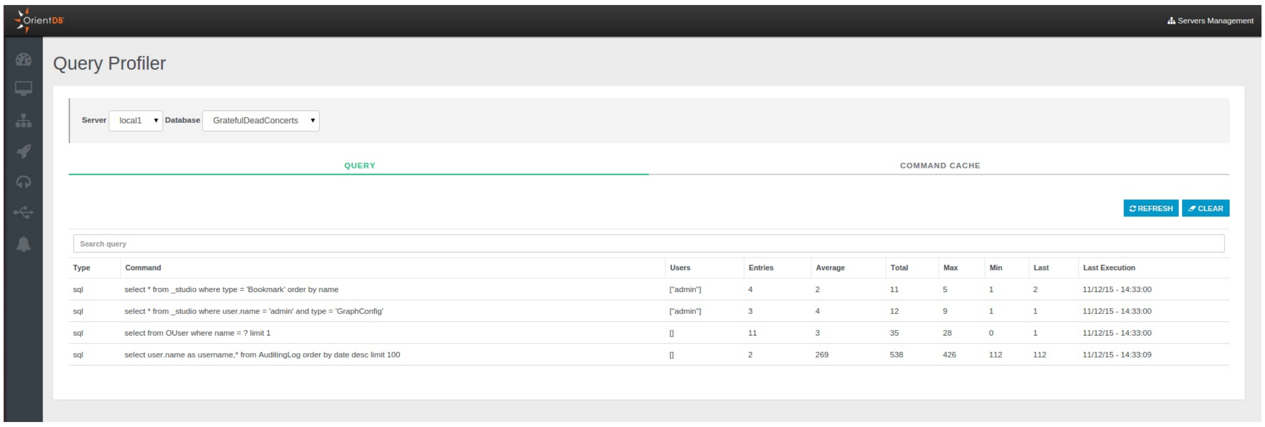
In the above section you can choose the server in order to investigate queries executed on it and manage the local cache.

NOTE: *This feature is available only in the OrientDB Enterprise Edition. If you are interested in a commercial license look at OrientDB Subscription Packages.*

Query

This panel shows all the queries executed on a specific server grouped by the command content. For each query the following information are reported:

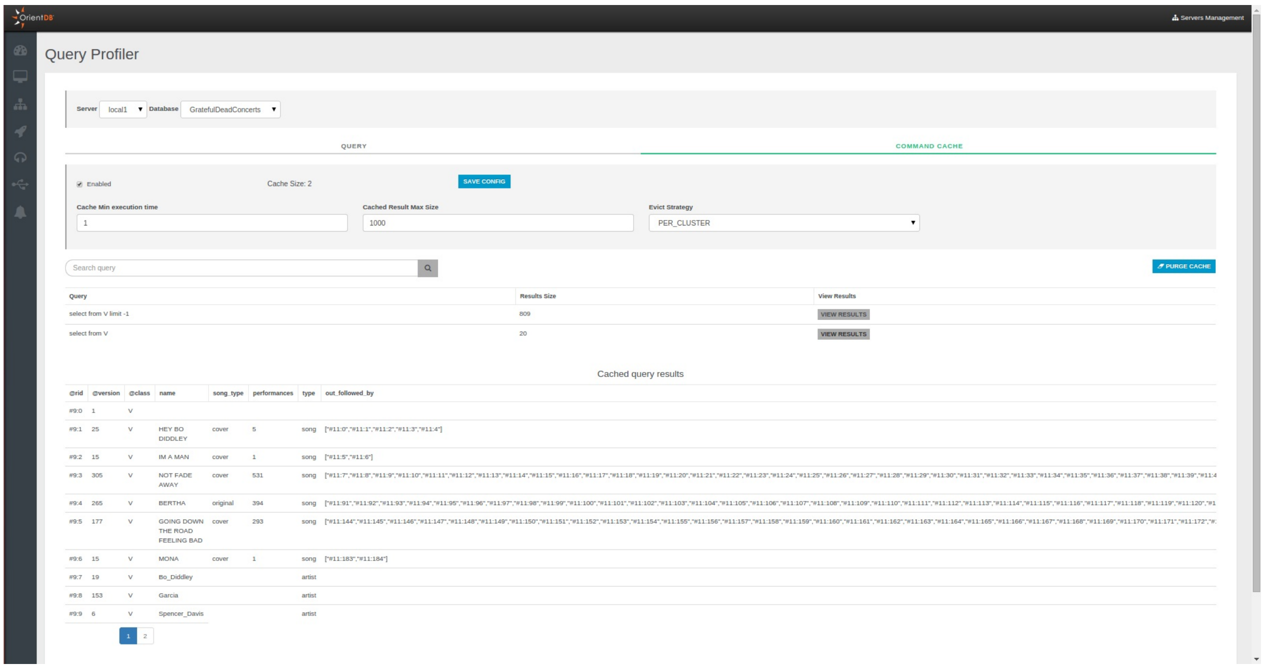
- `Type` , as the query type
- `Command` , as the content of the query
- `Users` , as the users who executed the query
- `Entries` , as the number of times the query it was executed
- `Average` , as the average required time by the queries
- `Total` , as the total required time by all the queries
- `Max` , as the maximum required time
- `Min` , as the minimum required time
- `Last` , as the time required by the last query
- `Last execution` , as the timestamp of the last query execution



Type	Command	Users	Entries	Average	Total	Max	Min	Last	Last Execution
sql	select * from _studio where type = 'Bookmark' order by name	["admin"]	4	2	11	5	1	2	11/12/15 - 14:33:00
sql	select * from _studio where user.name = 'admin' and type = 'GraphConfig'	["admin"]	3	4	12	9	1	1	11/12/15 - 14:33:00
sql	select from OUser where name = ? limit 1	[]	11	3	35	28	0	1	11/12/15 - 14:33:00
sql	select user.name as username,* from AuditingLog order by date desc limit 100	[]	2	269	538	426	112	112	11/12/15 - 14:33:09

Command Cache

Through this panel you can manage the cache of the specific server and consult the cached results of queries by using the `VIEW RESULTS` button. You can even filter the queries by the "Query" field and purge the whole cache by using the `PURGE CACHE` button.



Server local1 Database GratefulDeadConcerts

Enabled ✓ Cache Size: 2

Cache Min execution time	Cached Result Max Size	Evict Strategy
1	1000	PER_CLUSTER

Query	Results Size	View Results
select from V limit -1	809	VIEW RESULTS
select from V	20	VIEW RESULTS

Cached query results

@rid	@version	@class	name	song_type	performances	type	out_followed_by
#9:0	1	V					
#9:1	25	V	HEY BO DIDDLEY	cover	5	song	["#11:0","#11:1","#11:2","#11:3","#11:4"]
#9:2	15	V	IM A MAN	cover	1	song	["#11:5","#11:6"]
#9:3	305	V	NOT FADE AWAY	cover	531	song	["#11:7","#11:8","#11:9","#11:10","#11:11","#11:12","#11:13","#11:14","#11:15","#11:16","#11:17","#11:18","#11:19","#11:20","#11:21","#11:22","#11:23","#11:24","#11:25","#11:26","#11:27","#11:28","#11:29","#11:30","#11:31","#11:32","#11:33","#11:34","#11:35","#11:36","#11:37","#11:38","#11:39","#11:4
#9:4	265	V	BERTHA	original	394	song	["#11:91","#11:92","#11:93","#11:94","#11:95","#11:96","#11:97","#11:98","#11:99","#11:100","#11:101","#11:102","#11:103","#11:104","#11:105","#11:106","#11:107","#11:108","#11:109","#11:110","#11:111","#11:112","#11:113","#11:114","#11:115","#11:116","#11:117","#11:118","#11:119","#11:120","#1
#9:5	177	V	GOING DOWN THE ROAD FEELING BAD	cover	293	song	["#11:144","#11:145","#11:146","#11:147","#11:148","#11:149","#11:150","#11:151","#11:152","#11:153","#11:154","#11:155","#11:156","#11:157","#11:158","#11:159","#11:160","#11:161","#11:162","#11:163","#11:164","#11:165","#11:166","#11:167","#11:168","#11:169","#11:170","#11:171","#11:172","#:
#9:6	15	V	MONA	cover	1	song	["#11:183","#11:184"]
#9:7	19	V	Bo_Diddley			artist	
#9:8	153	V	Garcia			artist	
#9:9	6	V	Spencer_Davis			artist	

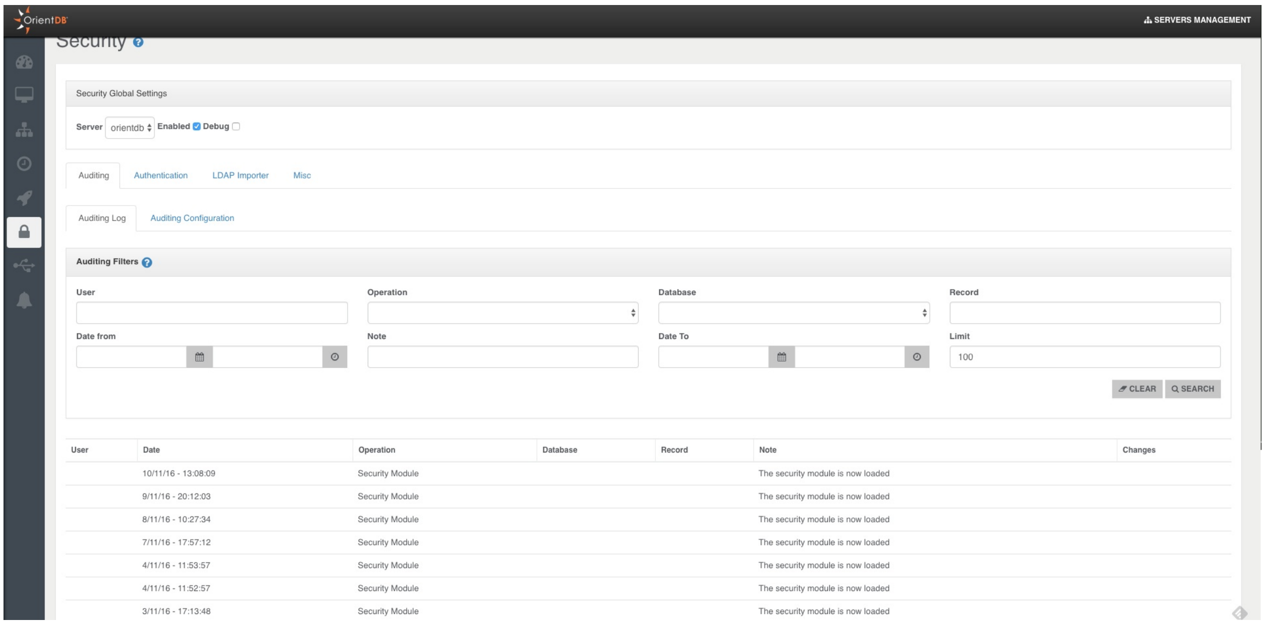
Auditing

Studio 2.2 Enterprise Edition includes a functionality called Auditing. To understand how Auditing works, please read the Auditing page on the OrientDB Manual.

NOTE: *This feature is available only in the OrientDB Enterprise Edition. If you are interested in a commercial license look at OrientDB Subscription Packages*.

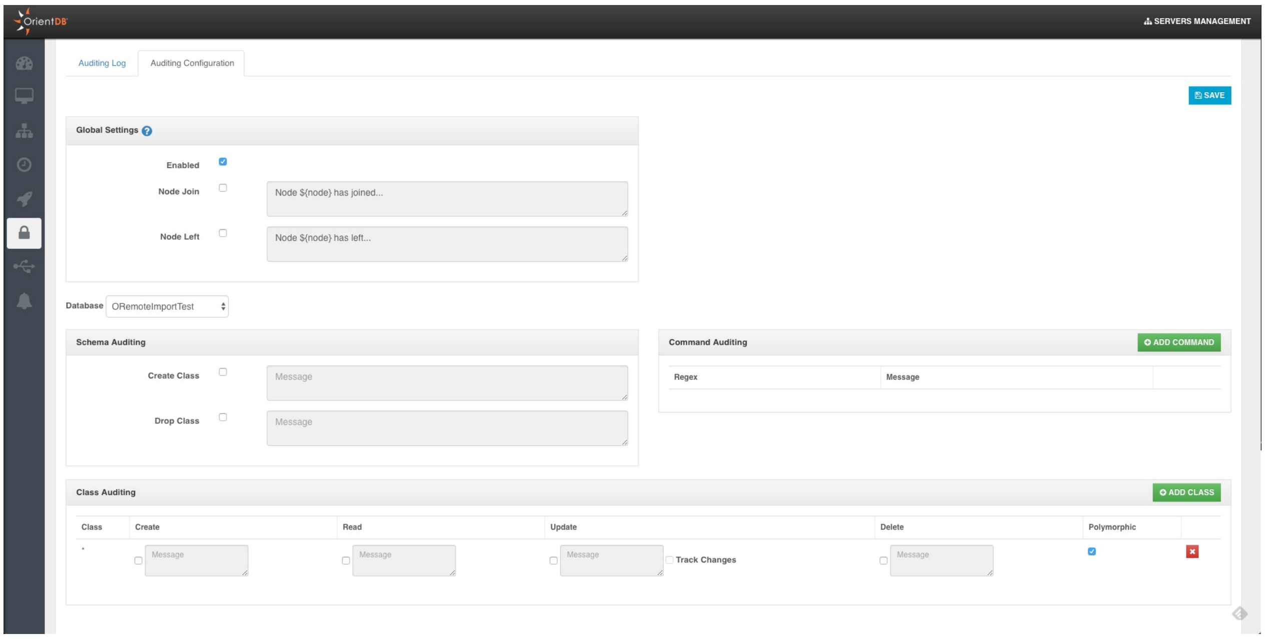
By default all the auditing logs are saved as documents of class `OAuditingLog` in the internal database `OSystem`. If your account has enough privileges, you can directly query the auditing log. Example on retrieving last 20 logs: `select from OAuditingLog order by @rid desc limit 20`.

However, Studio provides a panel to filter the Auditing Log messages on a specific server without using SQL.



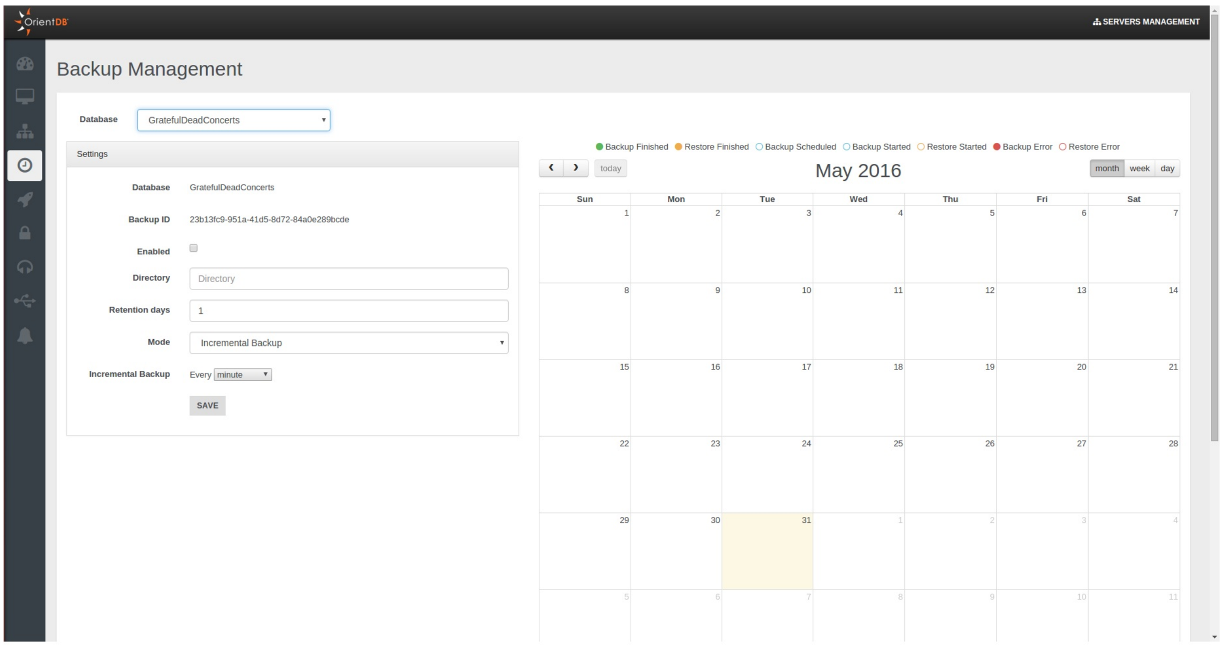
User	Date	Operation	Database	Record	Note	Changes
	10/11/16 - 13:08:09	Security Module			The security module is now loaded	
	9/11/16 - 20:12:03	Security Module			The security module is now loaded	
	8/11/16 - 10:27:34	Security Module			The security module is now loaded	
	7/11/16 - 17:57:12	Security Module			The security module is now loaded	
	4/11/16 - 11:53:57	Security Module			The security module is now loaded	
	4/11/16 - 11:52:57	Security Module			The security module is now loaded	
	3/11/16 - 17:13:48	Security Module			The security module is now loaded	

Studio Auditing panel helps you also on Auditing configuration of servers, avoiding to edit the `auditing-config.json` file under the database folder.



Backup Management (Enterprise Only)

Studio 2.2 Enterprise Edition includes a **Backup Manager** that allows you to schedule and perform your backups and easily execute and manage restores you may need. You can enjoy this new functionality by reaching the **Backup Management** panel in the Server Management area, this is what you will find:


OrientDB
SERVERS MANAGEMENT
Backup Management
● Backup Finished ● Restore Finished ○ Backup Scheduled ○ Backup Started ○ Restore Started ● Backup Error ○ Restore Error
today
May 2016
month
week
day
Sun
Mon
Tue
Wed
Thu
Fri
Sat

Database: GratefulDeadConcerts

Settings

Database	GratefulDeadConcerts
Backup ID	23b13fc9-951a-41d5-8d72-84a0e289bcde
Enabled	☐
Directory	Directory
Retention days	1
Mode	Incremental Backup
Incremental Backup	Every minute

SAVE

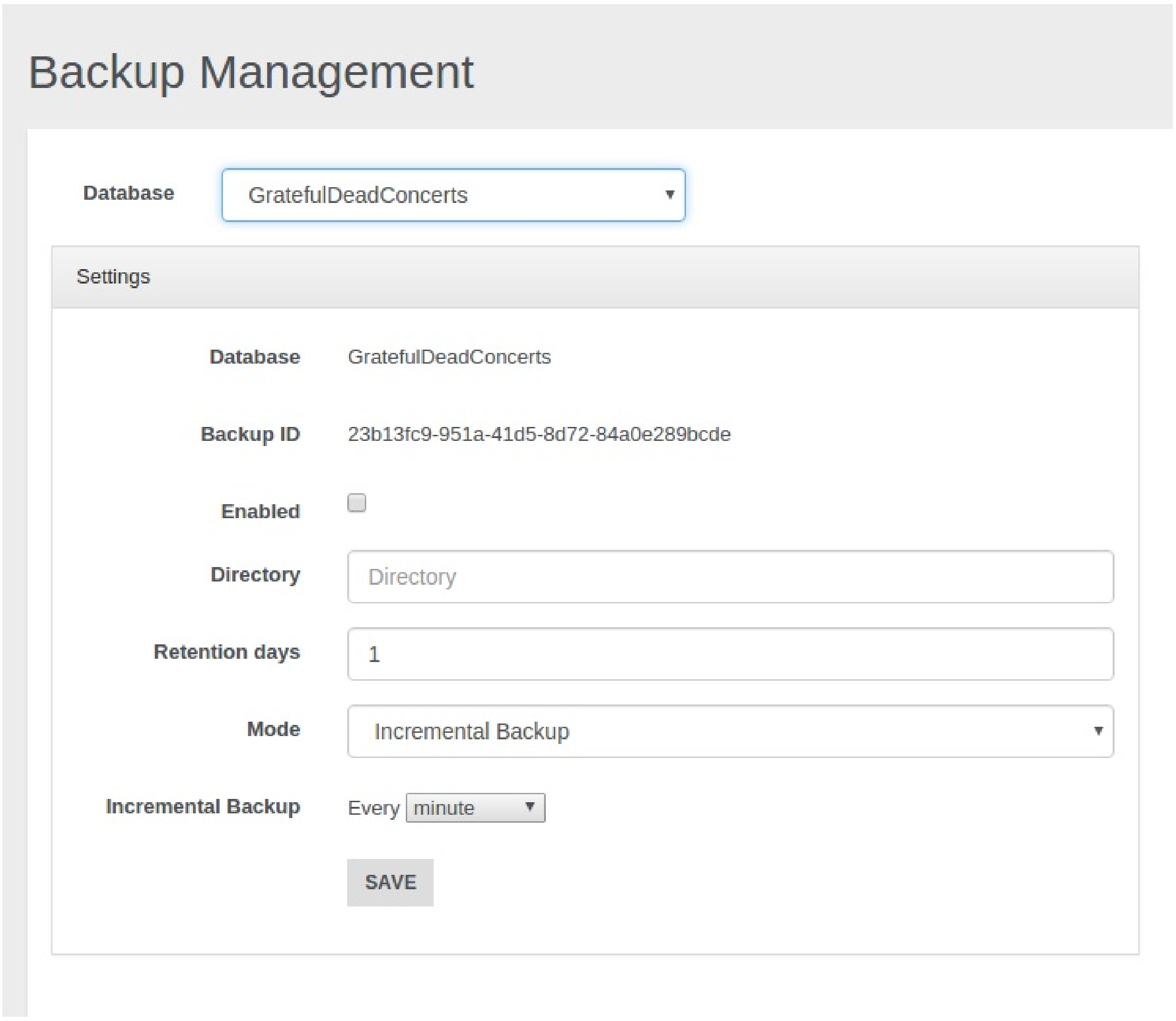
How you can see the panel is divided into two sections, on the left side you can schedule your backups, on the right side there is the calendar where you can check:

- the executed tasks (backup or restore)
- the scheduled backups
- eventual errors raised during the execution of a task

Now let's start seeing how you can schedule your backups.

Backup scheduling

On the left you can find all the settings for your backup scheduling.


Backup Management

Database: GratefulDeadConcerts

Settings

Database	GratefulDeadConcerts
Backup ID	23b13fc9-951a-41d5-8d72-84a0e289bcde
Enabled	☐
Directory	Directory
Retention days	1
Mode	Incremental Backup
Incremental Backup	Every minute

As first thing choose the **database** that you want backup. In the example above we have chosen the GratefulDeadConcerts database. Then you must specify the **output directory** where you want to save your backups and the **retention days** of your backups. Now you must select the **backup mode** you want to use:

- **Full backup**
- **Incremental Backup**
- **Full + Incremental Backup**

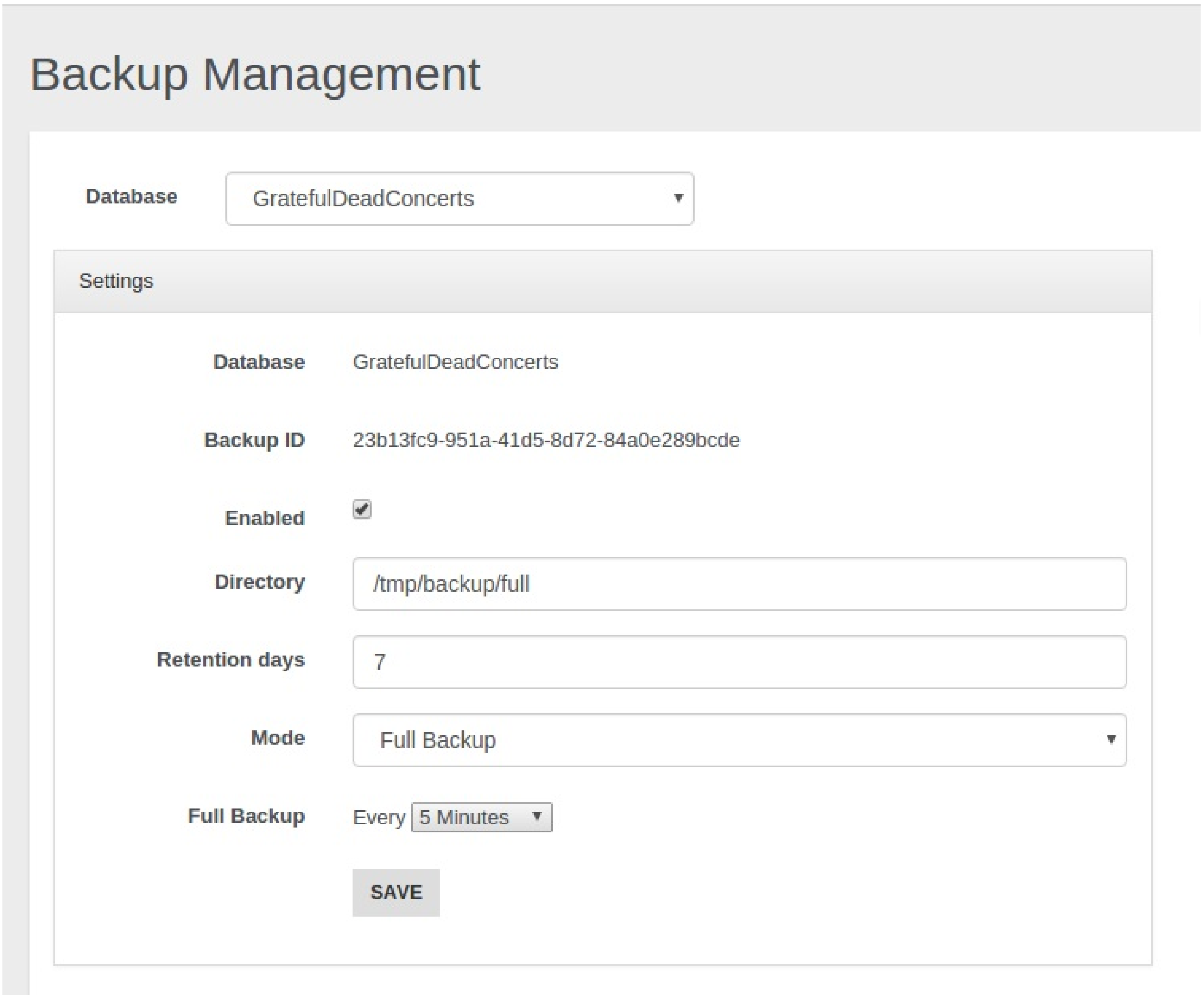
These modes will be analysed afterwards.

Once you have chosen the desired backup mode, you have to choose the **backup period** that indicates the time you want to wait between each backup and the next one. Eventually you must flag the **Enabled** checkbox and click on the **Save** button in order to start the scheduling of the backups according to your settings.

Below we will examine briefly the three different backup strategies.

Full backup

Through this mode when each period passes a **full backup** will be performed in the path you specified in the settings just discussed. If you want know more about the full backup you can refer to the Full Backup page.



Backup Management

Database: GratefulDeadConcerts

Settings

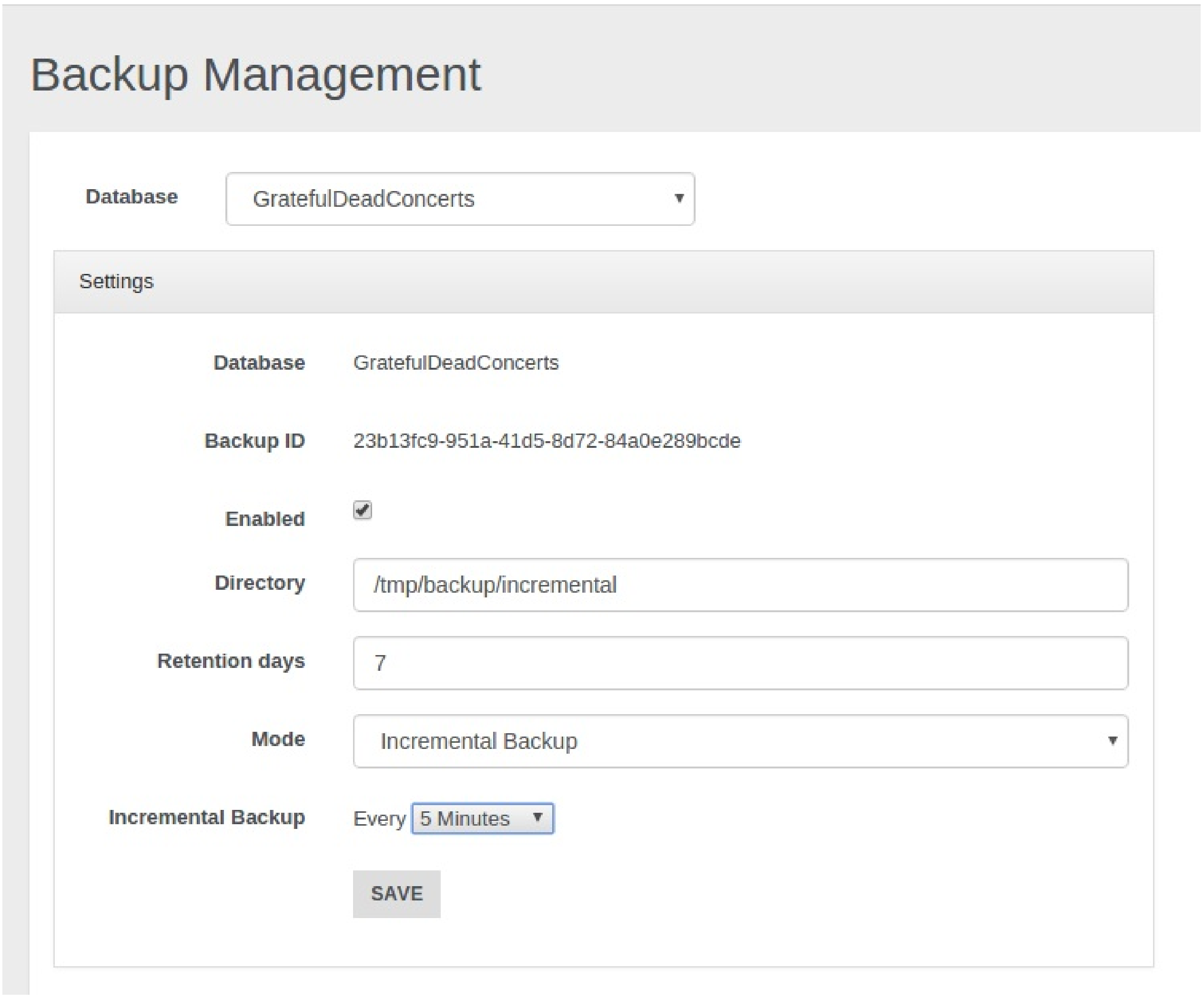
Database	GratefulDeadConcerts
Backup ID	23b13fc9-951a-41d5-8d72-84a0e289bcde
Enabled	✓
Directory	/tmp/backup/full
Retention days	7
Mode	Full Backup
Full Backup	Every 5 Minutes

With the settings shown above a full backup will be performed every 5 minutes. Thus in our example after 5 minutes we will have the first backup, after 10 minutes the second one and so on.

```
/tmp/backup/full
          |
          |____________GratefulDeadConcerts-1465213003035
          |                       |____________GratefulDeadConcerts_2016-06-06-13-36-43_0_full.ibu
          |
          |____________GratefulDeadConcerts-1465213020008
          |                       |____________GratefulDeadConcerts_2016-06-06-13-37-00_0_full.ibu
          |
          |____________GratefulDeadConcerts-1465213080003
          |                       |____________GratefulDeadConcerts_2016-06-06-13-38-00_0_full.ibu
          ...
```

Incremental Backup

If you prefer to execute an **incremental backup** you can select this mode. As declared in the Incremental Backup page the incremental backup generates smaller backup files by storing only the **delta** between two versions of the database. Let's suppose we want execute a backup every 5 minutes: a **first full backup** will be performed, then it will be followed by a new **incremental backup**, containing only the delta, **every 5 minutes**.


Backup Management

Database	GratefulDeadConcerts

Settings

Database	GratefulDeadConcerts
Backup ID	23b13fc9-951a-41d5-8d72-84a0e289bcde
Enabled	✓
Directory	/tmp/backup/incremental
Retention days	7
Mode	Incremental Backup
Incremental Backup	Every 5 Minutes

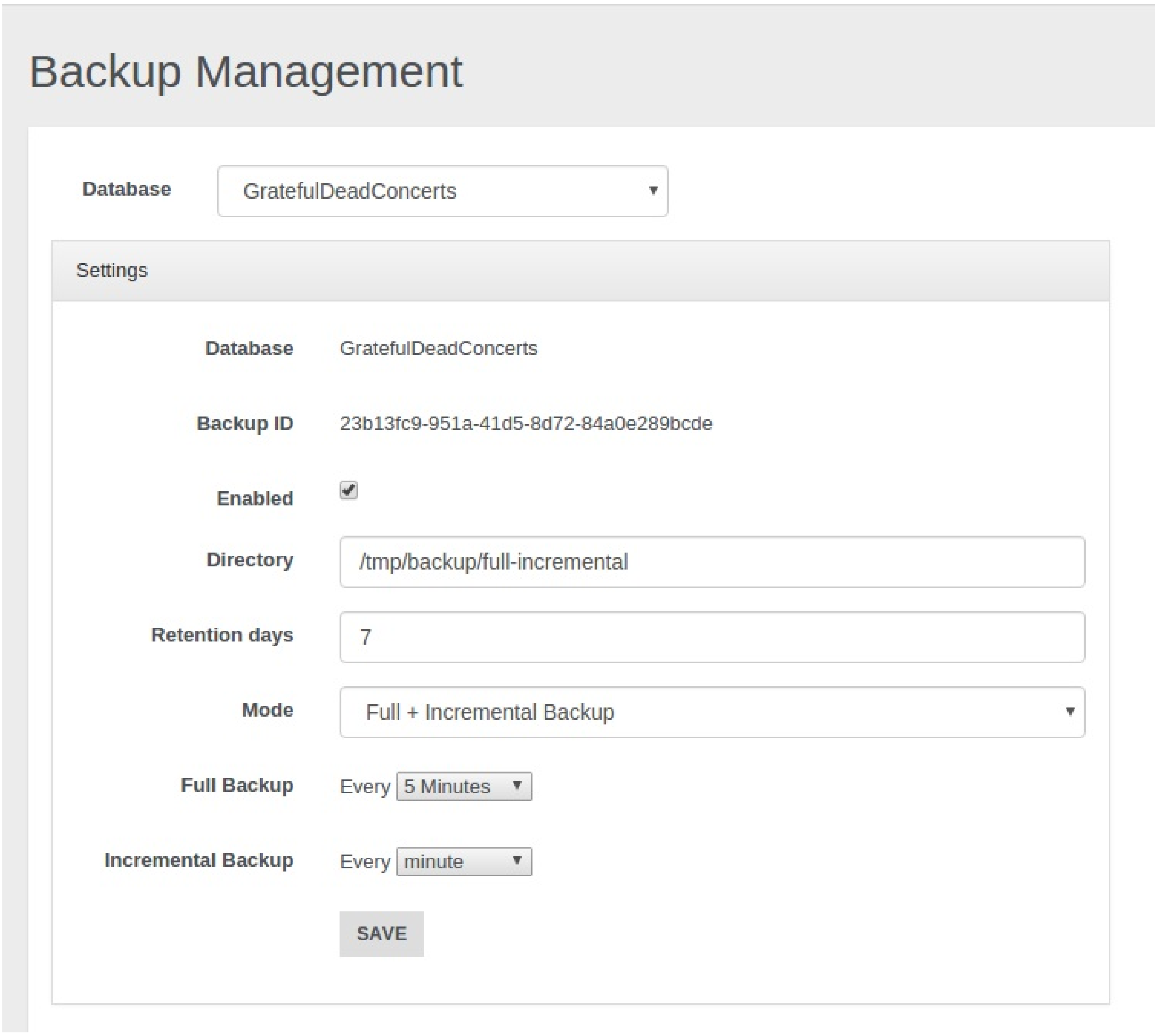
```
/tmp/backup/incremental
          |
          |___________GratefulDeadConcerts-incremental
                                |___________GratefulDeadConcerts_2016-06-06-13-27-00_0_full.ibu
                                |___________GratefulDeadConcerts_2016-06-06-13-28-00_1.ibu
                                |___________GratefulDeadConcerts_2016-06-06-13-29-00_2.ibu
                                |___________GratefulDeadConcerts_2016-06-06-13-30-00_3.ibu
                                |___________GratefulDeadConcerts_2016-06-06-13-31-00_4.ibu
                                ...
```

Full + Incremental Backup

This mode follows an **hybrid approach** between the first two strategies, combining them according to your criteria. The first significant thing you can notice is that **you must specify two different backup-periods**:

- **Full Backup period**: it specifies how much time will be waited between two sequential full backups.
- **Incremental Backup period**: it specifies how much time will be waited between two sequential incremental backups.

Let's analyse in which way the two modes are combined. Suppose we decided to execute the full backup every 5 minutes and the incremental backup every minute as shown in the example below.



Backup Management

Database	GratefulDeadConcerts

Settings

Database	GratefulDeadConcerts
Backup ID	23b13fc9-951a-41d5-8d72-84a0e289bcde
Enabled	✓
Directory	/tmp/backup/full-incremental
Retention days	7
Mode	Full + Incremental Backup
Full Backup	Every 5 Minutes
Incremental Backup	Every minute

SAVE

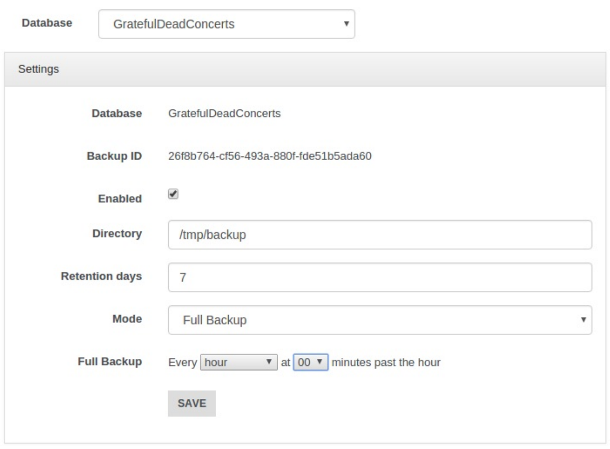
Thus we will obtain that **every 5 minutes** a new directory with a **full backup** will be added in the specified path, then **in the following 4 minutes** only **incremental backups** will be performed. As we set 1 minute for the incremental backup, we will have 4 incremental backups after the first full backup. After 5 minutes a new full backup in another folder will be performed, and the following incrementals will be executed according to the delta relative to this second full backup and they will put in this second folder. That's all, after another 5 minutes we will have a third directory with an initial full backup that will be followed by 4 incremental backups, and so on.

```
/tmp/backup/full-incremental
          |
          |____________GratefulDeadConcerts-1465213200182
          |                      |____________GratefulDeadConcerts_2016-06-06-13-40-00_0_full.ibu
          |                      |____________GratefulDeadConcerts_2016-06-06-13-41-00_1.ibu
          |                      |____________GratefulDeadConcerts_2016-06-06-13-42-00_2.ibu
          |                      |____________GratefulDeadConcerts_2016-06-06-13-44-00_3.ibu
          |                      |____________GratefulDeadConcerts_2016-06-06-13-44-00_4.ibu
          |
          |____________GratefulDeadConcerts-1465213440019
          |                      |____________GratefulDeadConcerts_2016-06-06-13-45-00_0_full.ibu
          |                      |____________GratefulDeadConcerts_2016-06-06-13-46-00_1.ibu
          |                      |____________GratefulDeadConcerts_2016-06-06-13-47-00_2.ibu
          |                      |____________GratefulDeadConcerts_2016-06-06-13-48-00_3.ibu
          |                      |____________GratefulDeadConcerts_2016-06-06-13-49-00_4.ibu
          |
          |____________GratefulDeadConcerts-1467210084991
          |                      |____________GratefulDeadConcerts_2016-06-06-13-46-00_0_full.ibu
          |                      ...
          ...
```

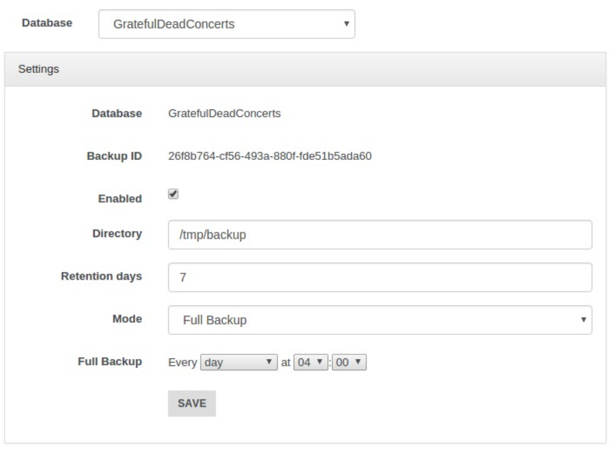
In this way we can have a "checkpoint" for each different directory to use in order to restore the database to a specific moment. You can decide if delete or maintain old backups and for each of them you can exploit the incremental backup features at the same time. To achieve this goal and use this feature properly mind that **full backup period must be major than incremental backup period**, different settings may cause illogical behaviours.

Granularity

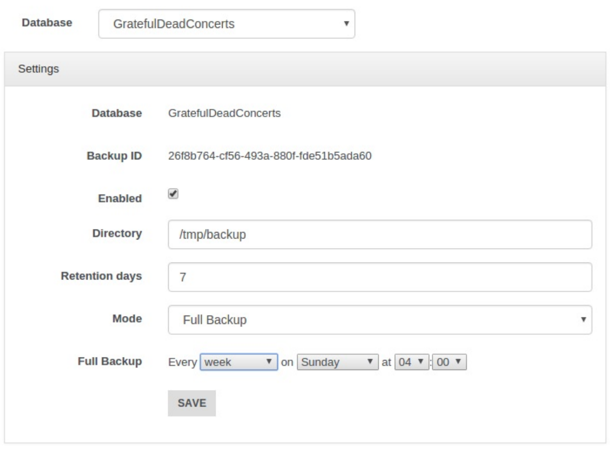
You can have different granularities to schedule your backups. Besides **minutes** granularity you can choose **hour**, **day**, **week**, **month**, and **year** granularity.



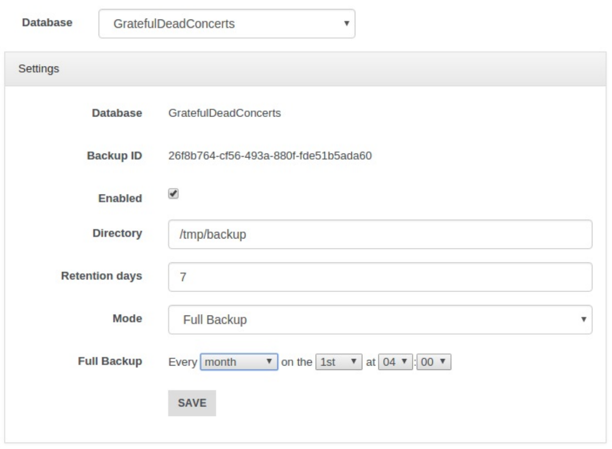
Database	GratefulDeadConcerts
Settings	
Database	GratefulDeadConcerts
Backup ID	26f8b764-cf56-493a-880f-fde51b5ada60
Enabled	✓
Directory	/tmp/backup
Retention days	7
Mode	Full Backup
Full Backup	Every hour at 00 minutes past the hour
	SAVE



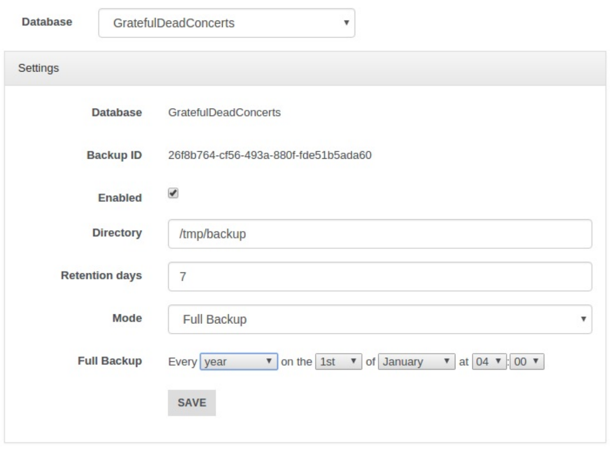
Database	GratefulDeadConcerts
Settings	
Database	GratefulDeadConcerts
Backup ID	26f8b764-cf56-493a-880f-fde51b5ada60
Enabled	✓
Directory	/tmp/backup
Retention days	7
Mode	Full Backup
Full Backup	Every day at 04 : 00
	SAVE



Database	GratefulDeadConcerts
Settings	
Database	GratefulDeadConcerts
Backup ID	26f8b764-cf56-493a-880f-fde51b5ada60
Enabled	✓
Directory	/tmp/backup
Retention days	7
Mode	Full Backup
Full Backup	Every week on Sunday at 04 : 00
	SAVE



Database	GratefulDeadConcerts
Settings	
Database	GratefulDeadConcerts
Backup ID	26f8b764-cf56-493a-880f-fde51b5ada60
Enabled	✓
Directory	/tmp/backup
Retention days	7
Mode	Full Backup
Full Backup	Every month on the 1st at 04 : 00
	SAVE



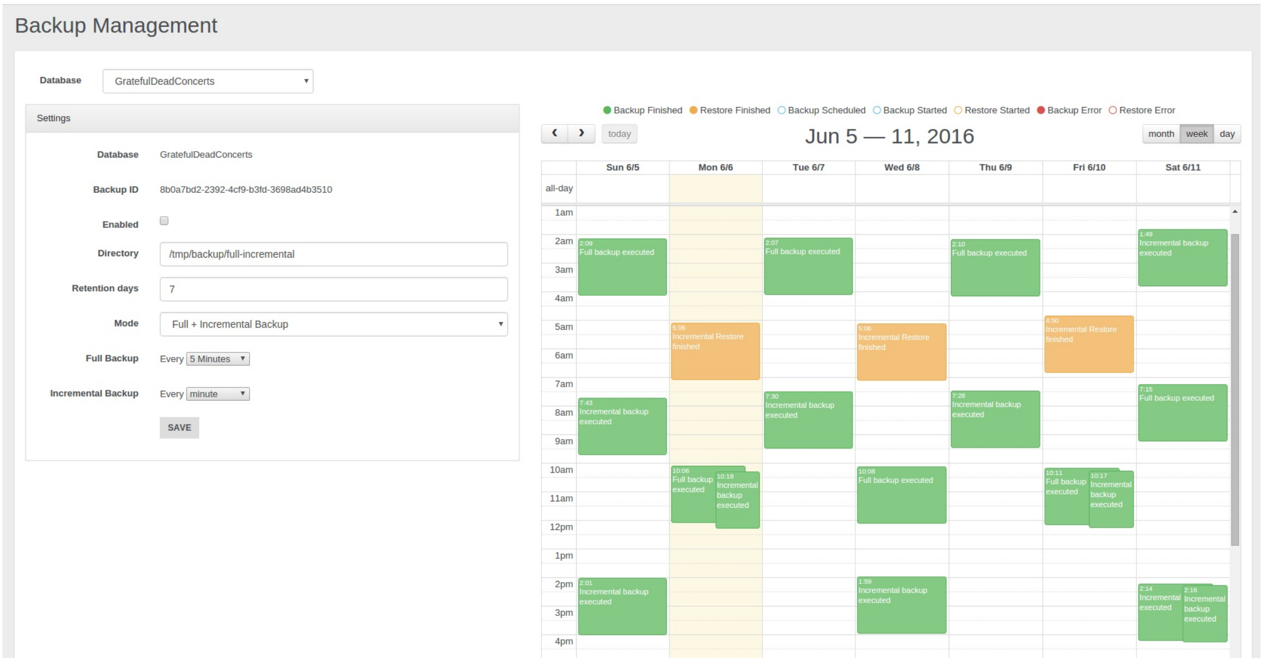
Database	GratefulDeadConcerts
Settings	
Database	GratefulDeadConcerts
Backup ID	26f8b764-cf56-493a-880f-fde51b5ada60
Enabled	✓
Directory	/tmp/backup
Retention days	7
Mode	Full Backup
Full Backup	Every year on the 1st of January at 04 : 00
	SAVE

Restore

In the calendar you can visualize and filter all the tasks (with the eventual related errors) through the voices below:

- Backup Finished

- Restore Finished
- Backup Scheduled
- Backup Started
- Restore Started
- Backup Error
- Restore Error


Backup Management

Database: GratefulDeadConcerts

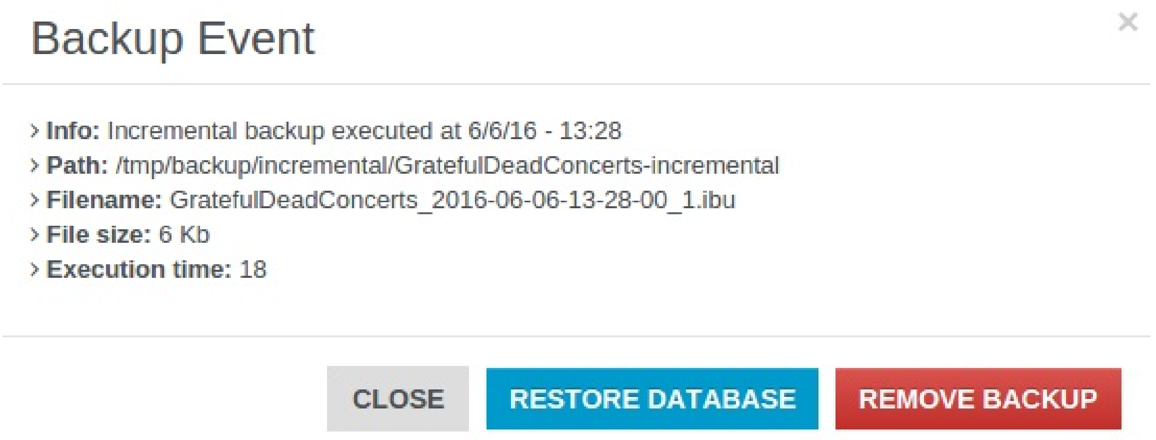
Settings

Database	GratefulDeadConcerts
Backup ID	8b0a7bd2-2392-4cf9-b3fd-3698ad4b3510
Enabled	☐
Directory	/tmp/backup/full-incremental
Retention days	7
Mode	Full + Incremental Backup
Full Backup	Every 5 Minutes
Incremental Backup	Every minute

SAVE
● Backup Finished ● Restore Finished ○ Backup Scheduled ○ Backup Started ○ Restore Started ● Backup Error ○ Restore Error
today
Jun 5 — 11, 2016
month
week
day
Sun 6/5
Mon 6/6
Tue 6/7
Wed 6/8
Thu 6/9
Fri 6/10
Sat 6/11
all-day
1am
2am
3am
4am
5am
6am
7am
8am
9am
10am
11am
12pm
1pm
2pm
3pm
4pm
2:09 Full backup executed
7:43 Incremental backup executed
2:01 Incremental backup executed
5:06 Incremental Restore finished
10:06 Full backup executed
10:18 Incremental backup executed
2:07 Full backup executed
7:30 Incremental backup executed
5:06 Incremental Restore finished
10:08 Full backup executed
1:59 Incremental backup executed
2:10 Full backup executed
7:28 Incremental backup executed
4:50 Incremental Restore finished
10:11 Full backup executed
10:17 Incremental backup executed
1:49 Incremental backup executed
7:15 Full backup executed
2:14 Incremental executed
2:16 Incremental backup executed

Notice you can choose three different scopes: **month**, **week** and **day**.

Clicking on a backup you can examine additional info like **execution time** and **timestamp**, **directory path**, **file name** and **file size**. Moreover you can **remove** the backup or carry out a restore starting from it.
Use this way to delete your backups because removing them manually may generates **unexpected behaviours**.


Backup Event

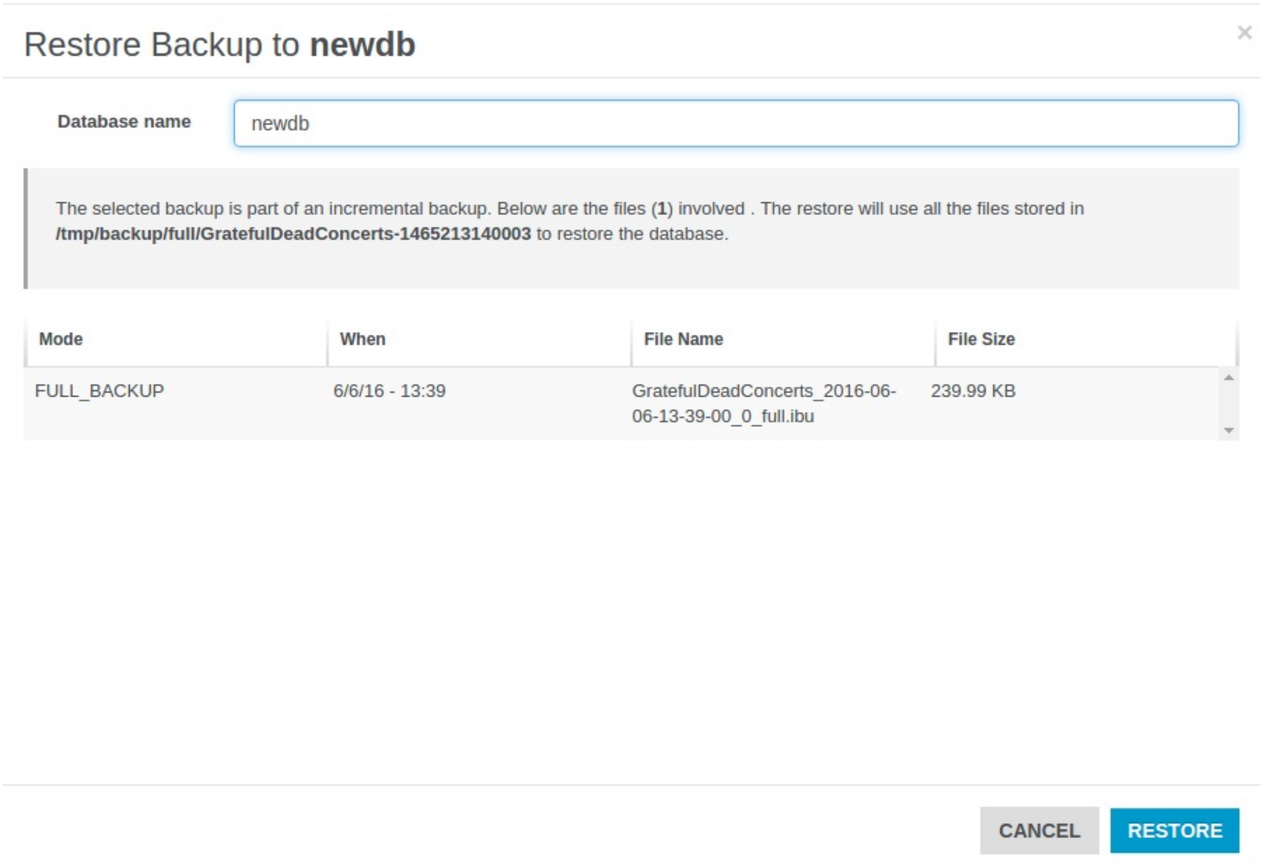
› **Info:** Incremental backup executed at 6/6/16 - 13:28
› **Path:** /tmp/backup/incremental/GratefulDeadConcerts-incremental
› **Filename:** GratefulDeadConcerts_2016-06-06-13-28-00_1.ibu
› **File size:** 6 Kb
› **Execution time:** 18

Let's make a restore by clicking on the button "Restore Database". A new window will be opened. Here you must **select the database where you want restore the backup**: notice you must declare just a name and a new empty database will be automatically created by the restore procedure, **don't use**:

- **an existent not-empty database**
- **fresh manually-built database**

Below are reported all the files involved in the restore procedure: the number of files used to restore your database depends on the **backup mode** you chose for the selected backup task.

If the backup belongs to a **Full Backup** schedule, just a file will be involved for each restore procedure.



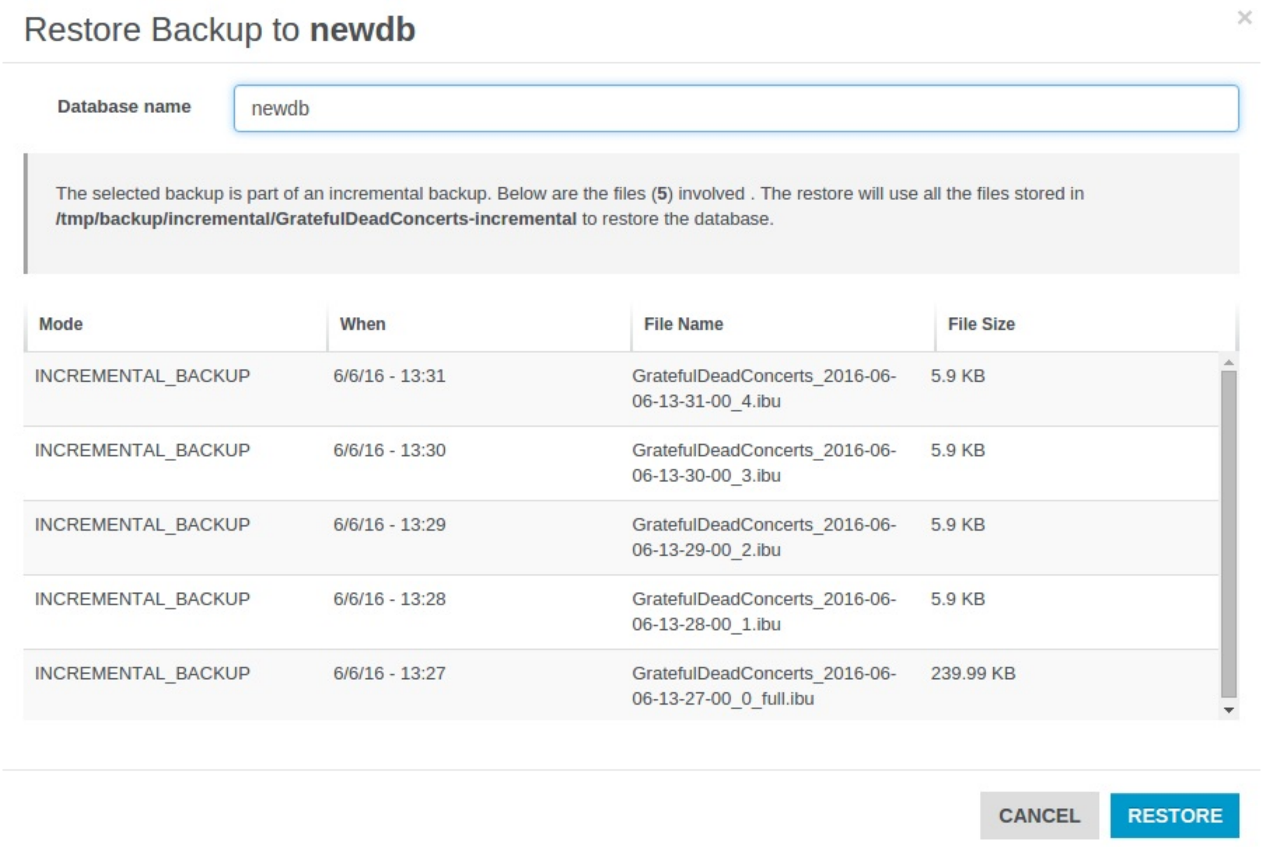
Restore Backup to **newdb**

Database name: newdb

The selected backup is part of an incremental backup. Below are the files (**1**) involved . The restore will use all the files stored in **/tmp/backup/full/GratefulDeadConcerts-1465213140003** to restore the database.

Mode	When	File Name	File Size
FULL_BACKUP	6/6/16 - 13:39	GratefulDeadConcerts_2016-06-06-13-39-00_0_full.ibu	239.99 KB

If the backup belongs to an **Incremental Backup** schedule, doesn't matter which file is selected, all the files in the directory will be processed during the restore.



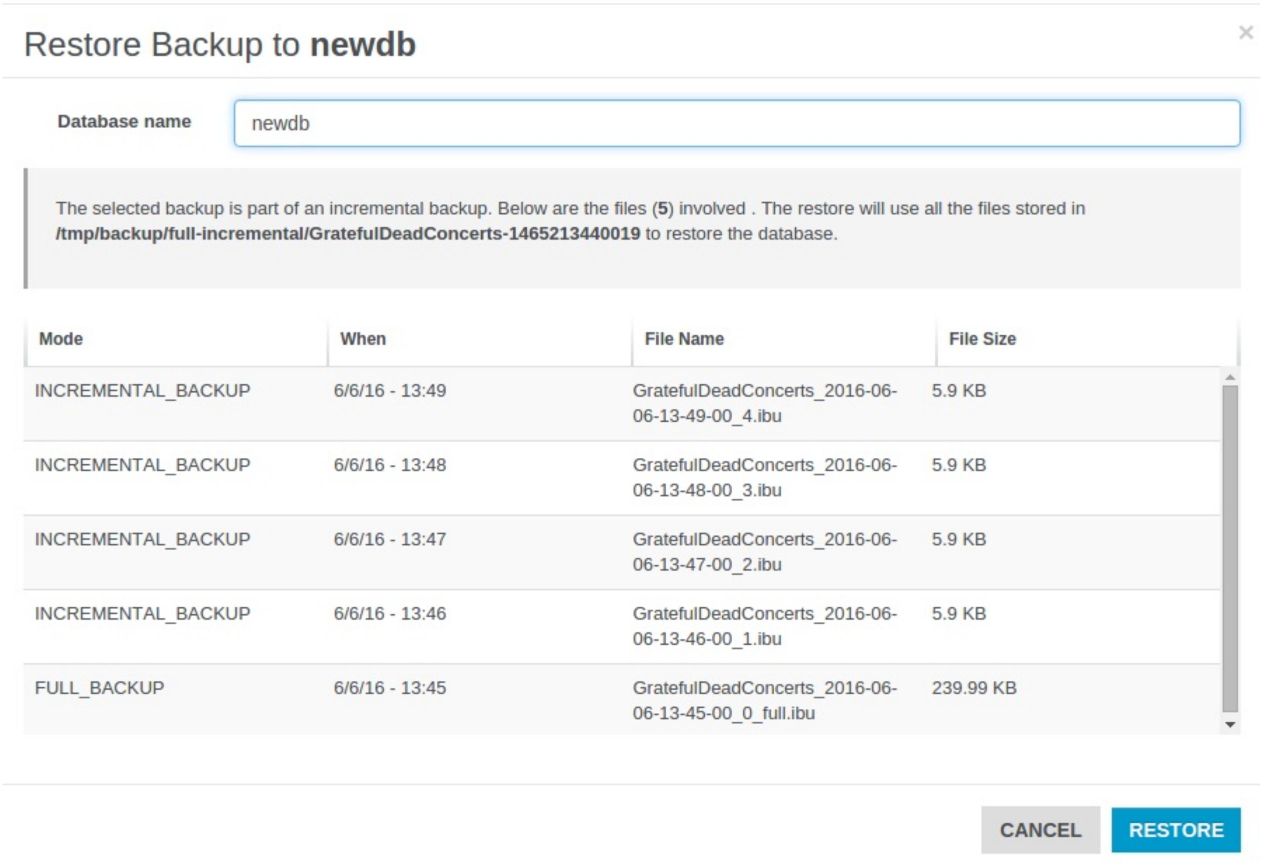
Restore Backup to **newdb**

Database name: newdb

The selected backup is part of an incremental backup. Below are the files (**5**) involved . The restore will use all the files stored in **/tmp/backup/incremental/GratefulDeadConcerts-incremental** to restore the database.

Mode	When	File Name	File Size
INCREMENTAL_BACKUP	6/6/16 - 13:31	GratefulDeadConcerts_2016-06-06-13-31-00_4.ibu	5.9 KB
INCREMENTAL_BACKUP	6/6/16 - 13:30	GratefulDeadConcerts_2016-06-06-13-30-00_3.ibu	5.9 KB
INCREMENTAL_BACKUP	6/6/16 - 13:29	GratefulDeadConcerts_2016-06-06-13-29-00_2.ibu	5.9 KB
INCREMENTAL_BACKUP	6/6/16 - 13:28	GratefulDeadConcerts_2016-06-06-13-28-00_1.ibu	5.9 KB
INCREMENTAL_BACKUP	6/6/16 - 13:27	GratefulDeadConcerts_2016-06-06-13-27-00_0_full.ibu	239.99 KB

If you chose a backup belonging to a **Full + Incremental Backup** schedule, then will be evaluated all the files contained in the folder which contains the backup file you selected from the calendar.



Restore Backup to **newdb**

Database name: newdb

The selected backup is part of an incremental backup. Below are the files (**5**) involved . The restore will use all the files stored in **/tmp/backup/full-incremental/GratefulDeadConcerts-1465213440019** to restore the database.

Mode	When	File Name	File Size
INCREMENTAL_BACKUP	6/6/16 - 13:49	GratefulDeadConcerts_2016-06-06-13-49-00_4.ibu	5.9 KB
INCREMENTAL_BACKUP	6/6/16 - 13:48	GratefulDeadConcerts_2016-06-06-13-48-00_3.ibu	5.9 KB
INCREMENTAL_BACKUP	6/6/16 - 13:47	GratefulDeadConcerts_2016-06-06-13-47-00_2.ibu	5.9 KB
INCREMENTAL_BACKUP	6/6/16 - 13:46	GratefulDeadConcerts_2016-06-06-13-46-00_1.ibu	5.9 KB
FULL_BACKUP	6/6/16 - 13:45	GratefulDeadConcerts_2016-06-06-13-45-00_0_full.ibu	239.99 KB

Teleporter

In Studio 2.2 you can configure the execution of the new Teleporter plugin, which allows you to import your relational database into OrientDB in few simple steps. If you are interested in a detailed description of the tool, of its inner workings and features you can view the Teleporter Documentation.

NOTE: This feature is available both for the OrientDB Enterprise Edition and the OrientDB Community Edition. **But beware**: in **Community Edition** you can migrate your source relational database but **you cannot enjoy the synchronize feature**, only available in the **Enterprise Edition**.

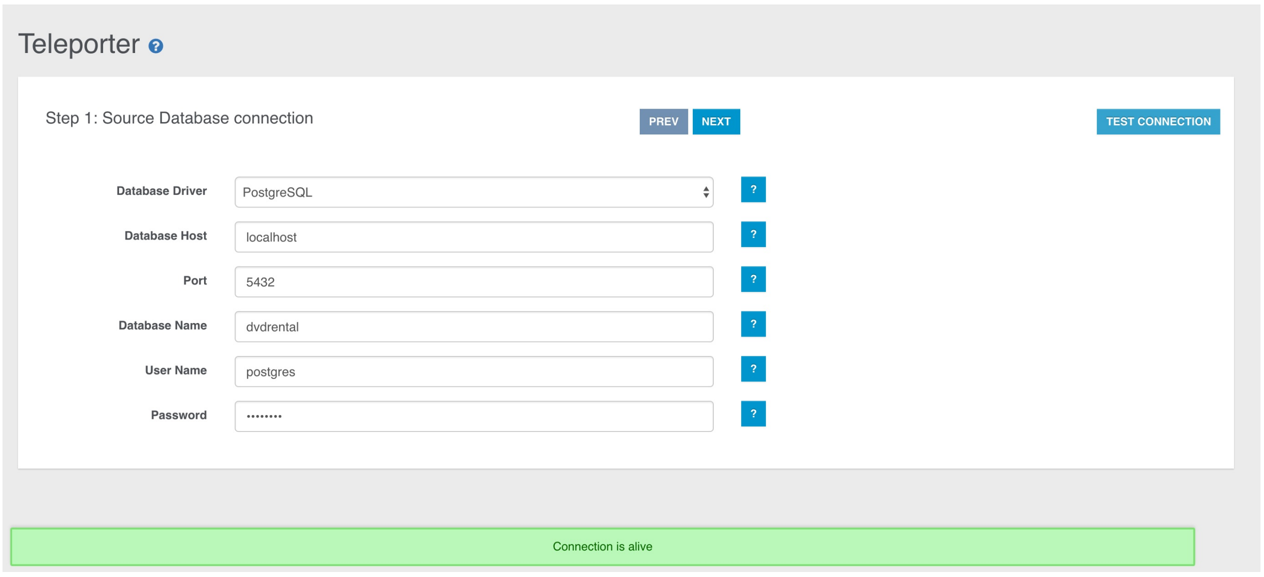
This visual tool consists in a wizard composed of 4 steps, where just **Step 1** and **Step 2** are strictly necessary in order to perform your migration. Let's have a look at each configuration step.

Step 1

In the first step you have to type the following required parameters:

- `Database Driver` , as the driver name of the DBMS from which you want to execute the import. You have to choose among:
  - Oracle
  - SQLServer
  - Mysql
  - PostgreSQL
  - HyperSQL
- `Database Host` , as the host where your DBMS instance is running on
- `Port` , as the port where your DBMS is listening on
- `Database Name` , as the name of the source database
- `User Name` , as the username to access the source database (it may be blank)
- `Password` , as the password to access the source database (it may be blank)

After you typed all the required parameters for the migration you can test the connection.



Step 1: Source Database connection

Database Driver	PostgreSQL
Database Host	localhost
Port	5432
Database Name	dvdrental
User Name	postgres
Password	••••••••

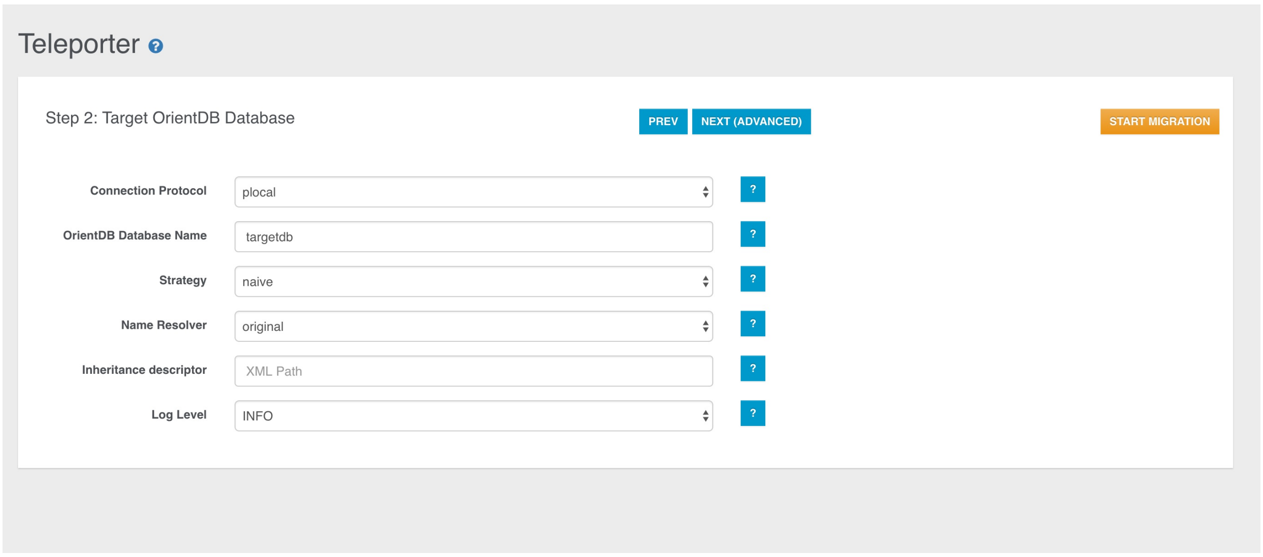
Connection is alive

Step 2

In the second step you have to specify all the parameters about the OrientDB target database:

- `Connection protocol` , as the protocol adopted to write in OrientDB. You have to choose among:
  - plocal
  - memory
- `OrientDB Database Name` , as the name of the target database in OrientDB
- `Strategy` , as the strategy adopted during the migration (More about strategies)
- `Name Resolver` , as the basic name resolver to adopt during names' resolution

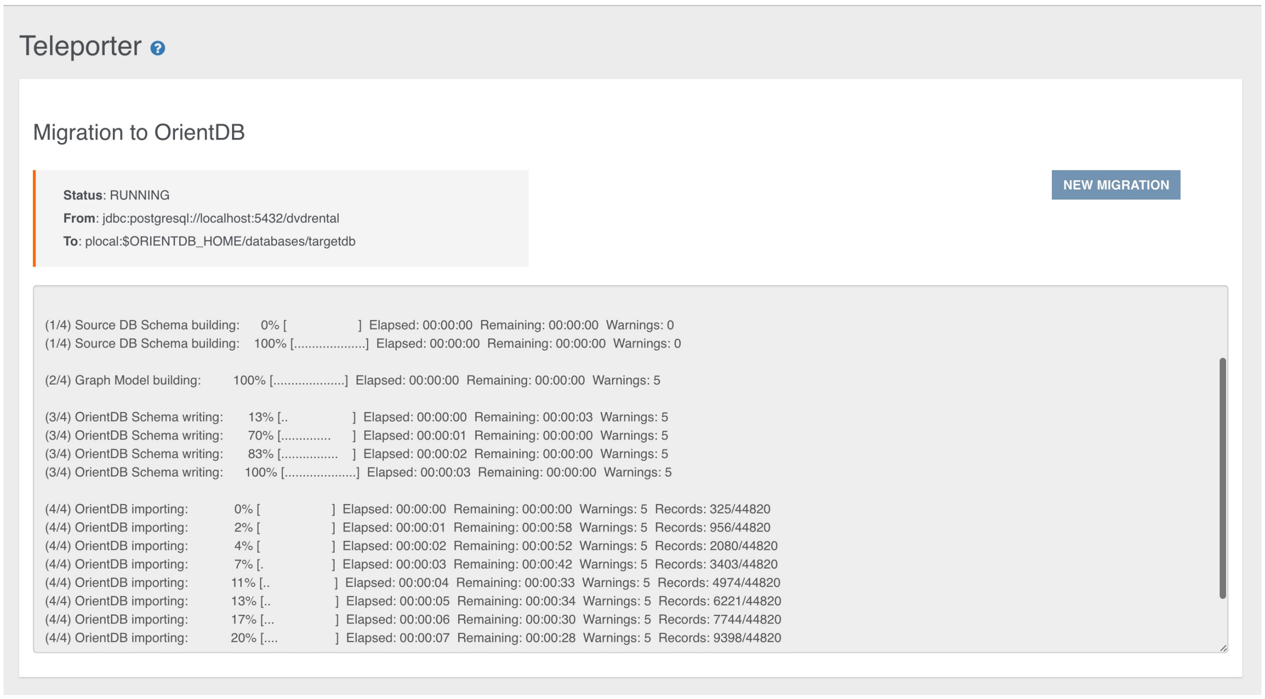
- `Inheritance descriptor` , as the XML file's path. In this descriptor you can report all the info describing inheritance relationships present between the tables in the source database
- `Log Level` , as the log level adopted by Teleporter during the migration. You can choose among:
  - NO
  - DEBUG
  - INFO
  - WARNING
  - ERROR


Teleporter

Step 2: Target OrientDB Database

Connection Protocol	plocal
OrientDB Database Name	targetdb
Strategy	naive
Name Resolver	original
Inheritance descriptor	XML Path
Log Level	INFO

Once we have collected all the minimal info needed for the migration, you can run your configured job through the `START MIGRATION` button, then the job progress monitor will be displayed:

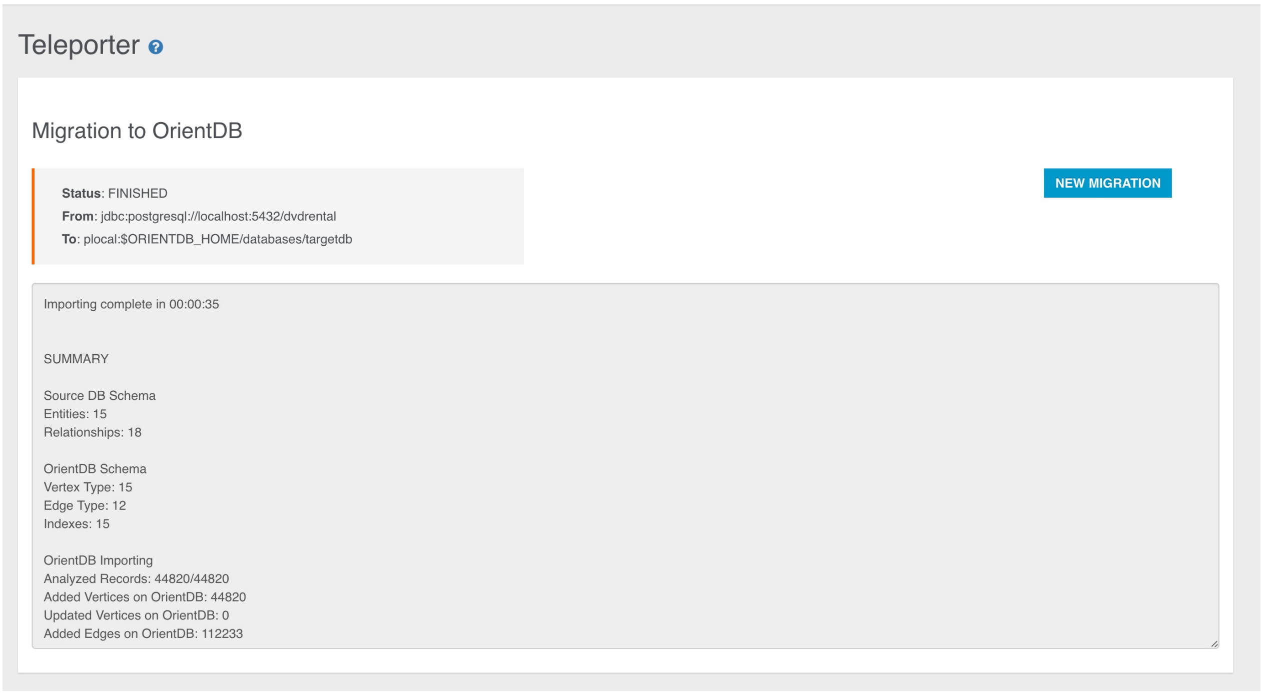

Teleporter

Migration to OrientDB

Status: RUNNING
From: jdbc:postgresql://localhost:5432/dvdrental
To: plocal:$ORIENTDB_HOME/databases/targetdb

```
(1/4) Source DB Schema building:     0% [                    ]  Elapsed: 00:00:00  Remaining: 00:00:00  Warnings: 0
(1/4) Source DB Schema building:   100% [....................]  Elapsed: 00:00:00  Remaining: 00:00:00  Warnings: 0

(2/4) Graph Model building:        100% [....................]  Elapsed: 00:00:00  Remaining: 00:00:00  Warnings: 5

(3/4) OrientDB Schema writing:      13% [..                  ]  Elapsed: 00:00:00  Remaining: 00:00:03  Warnings: 5
(3/4) OrientDB Schema writing:      70% [..............      ]  Elapsed: 00:00:01  Remaining: 00:00:00  Warnings: 5
(3/4) OrientDB Schema writing:      83% [................    ]  Elapsed: 00:00:02  Remaining: 00:00:00  Warnings: 5
(3/4) OrientDB Schema writing:     100% [....................]  Elapsed: 00:00:03  Remaining: 00:00:00  Warnings: 5

(4/4) OrientDB importing:            0% [                    ]  Elapsed: 00:00:00  Remaining: 00:00:00  Warnings: 5  Records: 325/44820
(4/4) OrientDB importing:            2% [                    ]  Elapsed: 00:00:01  Remaining: 00:00:58  Warnings: 5  Records: 956/44820
(4/4) OrientDB importing:            4% [                    ]  Elapsed: 00:00:02  Remaining: 00:00:52  Warnings: 5  Records: 2080/44820
(4/4) OrientDB importing:            7% [.                   ]  Elapsed: 00:00:03  Remaining: 00:00:42  Warnings: 5  Records: 3403/44820
(4/4) OrientDB importing:           11% [..                  ]  Elapsed: 00:00:04  Remaining: 00:00:33  Warnings: 5  Records: 4974/44820
(4/4) OrientDB importing:           13% [..                  ]  Elapsed: 00:00:05  Remaining: 00:00:34  Warnings: 5  Records: 6221/44820
(4/4) OrientDB importing:           17% [...                 ]  Elapsed: 00:00:06  Remaining: 00:00:30  Warnings: 5  Records: 7744/44820
(4/4) OrientDB importing:           20% [....                ]  Elapsed: 00:00:07  Remaining: 00:00:28  Warnings: 5  Records: 9398/44820
```

At the end of the migration, statistics and warnings about the process are reported as shown below:



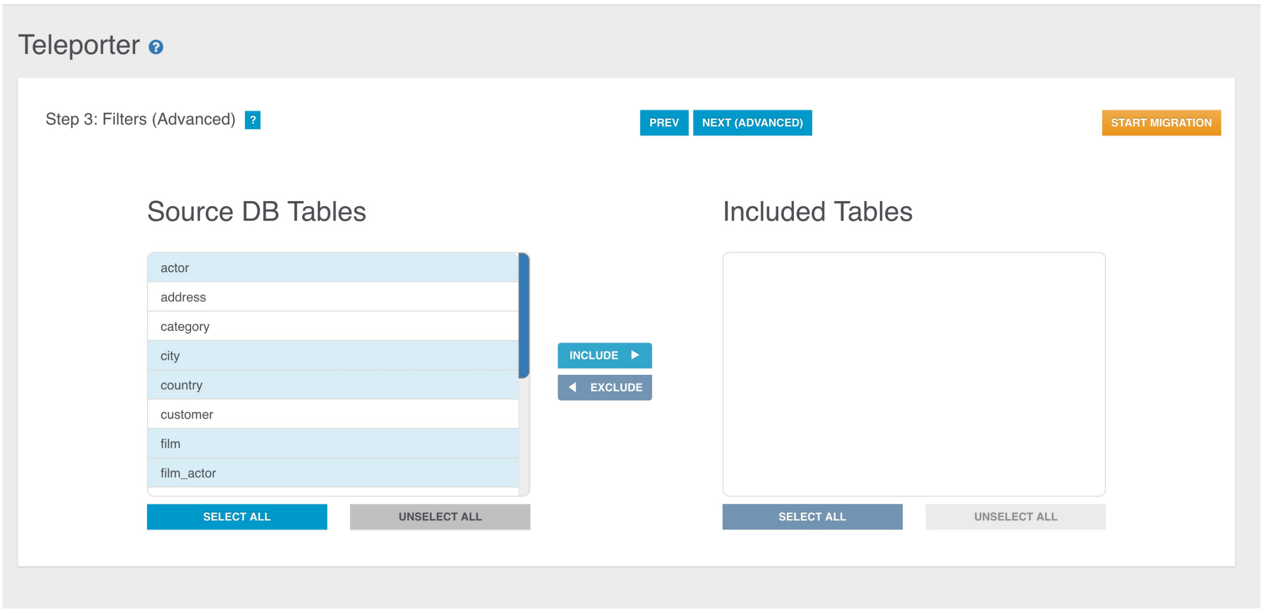
Status: FINISHED
From: jdbc:postgresql://localhost:5432/dvdrental
To: plocal:$ORIENTDB_HOME/databases/targetdb

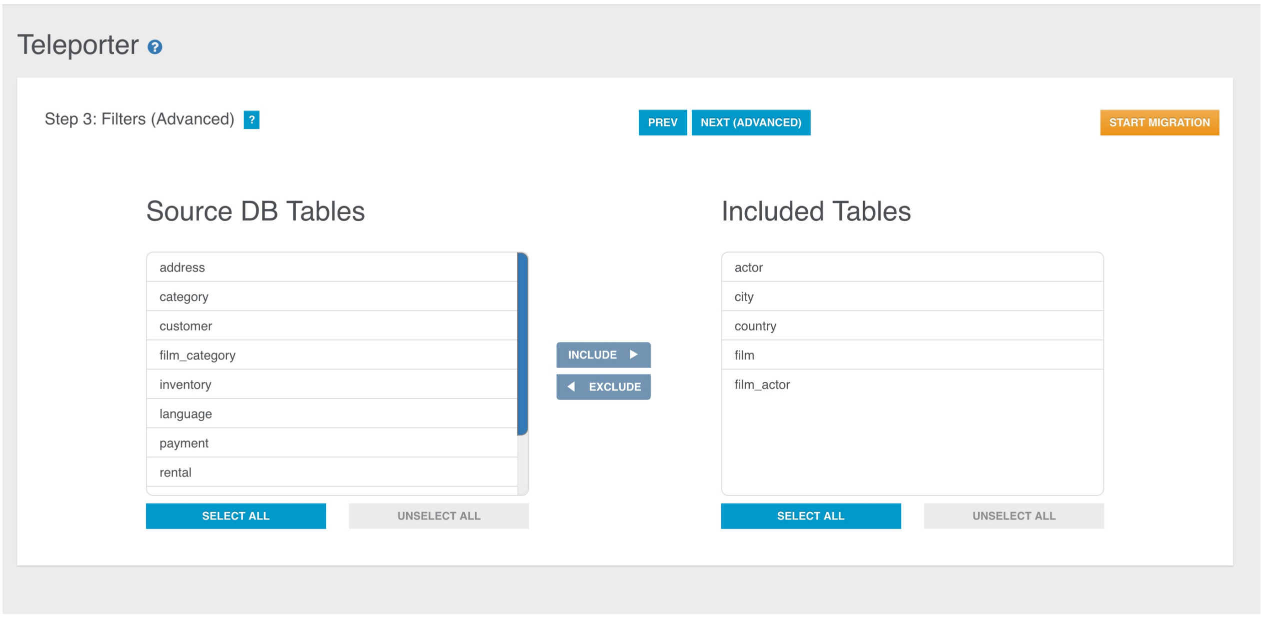
```
Importing complete in 00:00:35


SUMMARY

Source DB Schema
Entities: 15
Relationships: 18

OrientDB Schema
Vertex Type: 15
Edge Type: 12
Indexes: 15

OrientDB Importing
Analyzed Records: 44820/44820
Added Vertices on OrientDB: 44820
Updated Vertices on OrientDB: 0
Added Edges on OrientDB: 112233
```

Otherwise you can go on in your migrationg customisation jumping to the next step.

Step 3

Here you can exploit Teleporter's filtering features: in the panel on the left all the tables present in the source database are reported. If you want migrate just a subset of these tables, you just have to select and move them in the right panel through the specific buttons (you can also drag-and-drop the selected items).



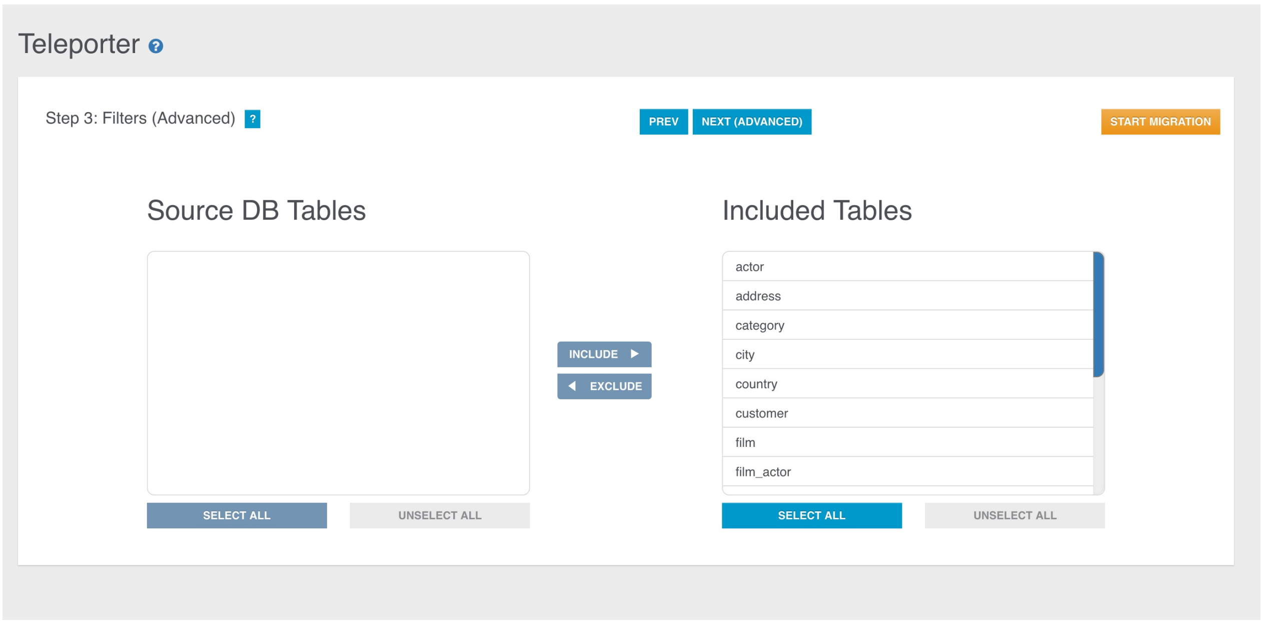


Source DB Tables: address, category, customer, film_category, inventory, language, payment, rental

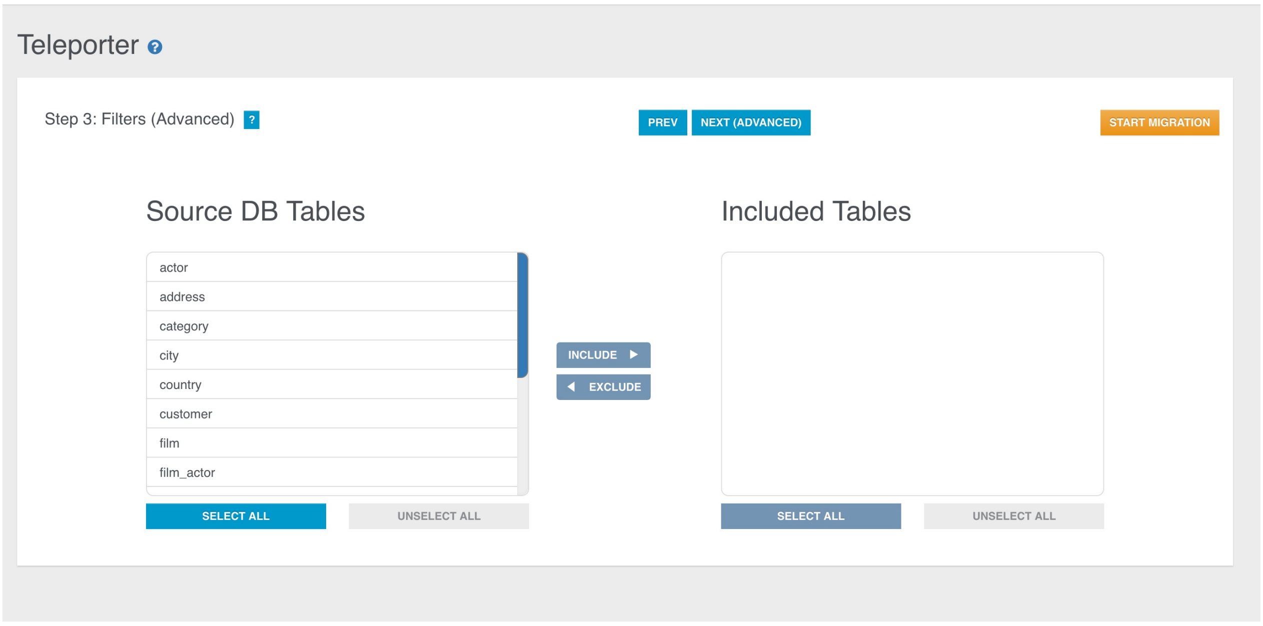
Included Tables: actor, city, country, film, film_actor

You can perform the same operations also in the opposite direction, that is excluding some tables during the migration just moving them from the right panel to the left one.

If the right panel is empty, no filters will be applied. Instead, if the right panel is not empty, just the selected tables in the right panel will be imported while all the others will be filtered out. Thus, for example, these two configurations are equivalent:



Included Tables: actor, address, category, city, country, customer, film, film_actor

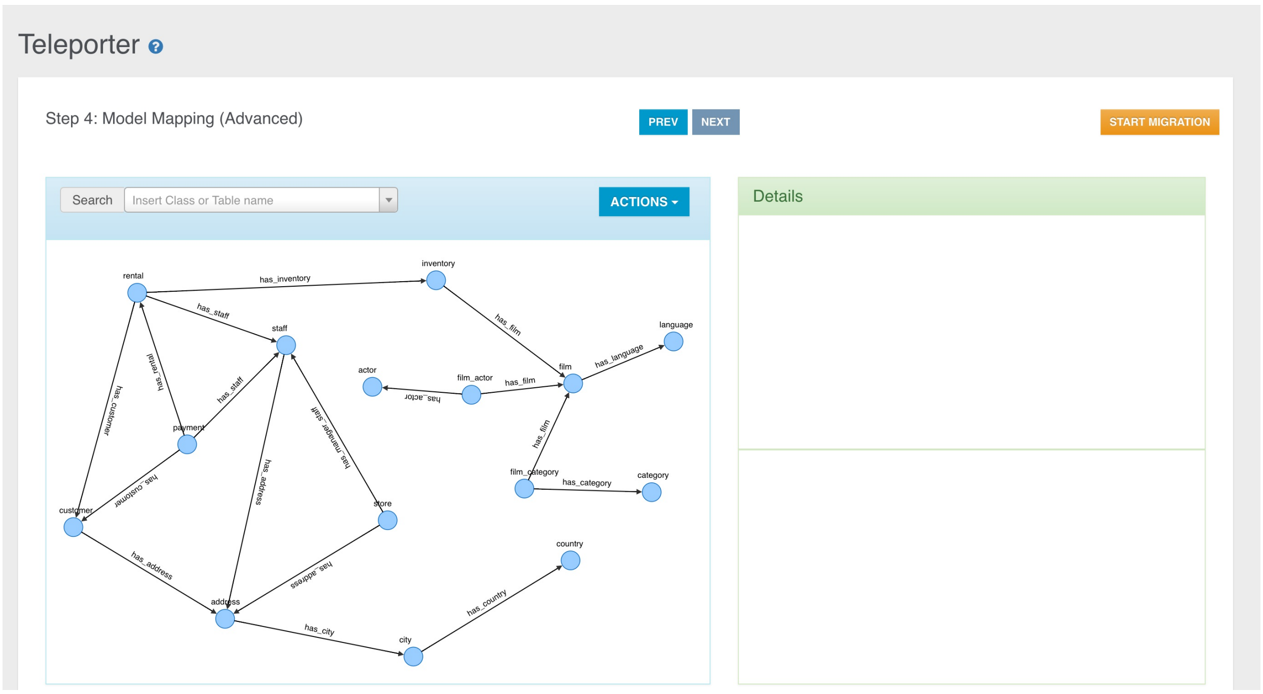


Here too you can start your migration or go to the 4th and last configuration step.

Step 4

In the last step Teleporter will provide you a Graph Model coming from the translation of the ER-Model inferred from the source database schema. The correspondent Graph Model is built according to basic mapping rules and your choices as well (filters applied, chosen strategy, name resolver adopted etc.). This step has two aims:

- it gives you an idea of how your source database will appear once imported in OrientDB
- it allows you to edit the graph model



You can see two panels, the Graph Model Panel on the left, containing the Graph Model built from Teleporter, and the Details Panel on the right, reporting all the details about the current selected element in the left panel.

The Details Panel is divided into two sections:

1. in the top area you can enjoy a graph perspective of the element selected in the Graph Panel: you can inspect info about the OrientDB schema, like class name and properties.
2. in the bottom area you have a source-schema perspective, where you got the source-schema items the information above comes from.

This step is conceived to make very easy the graph model editing and to change the mapping with the source database schema. In fact you can modify the basic mapping

- Renaming classes (both for Vertex and Edge classes)
- Excluding/re-including a property mapped with a column in the correspondent source table
- Adding new properties
- Dropping existent properties
- Editing properties
- Adding new Edge classes and/or instances
- Inspecting original schema data, both for tables and relationships

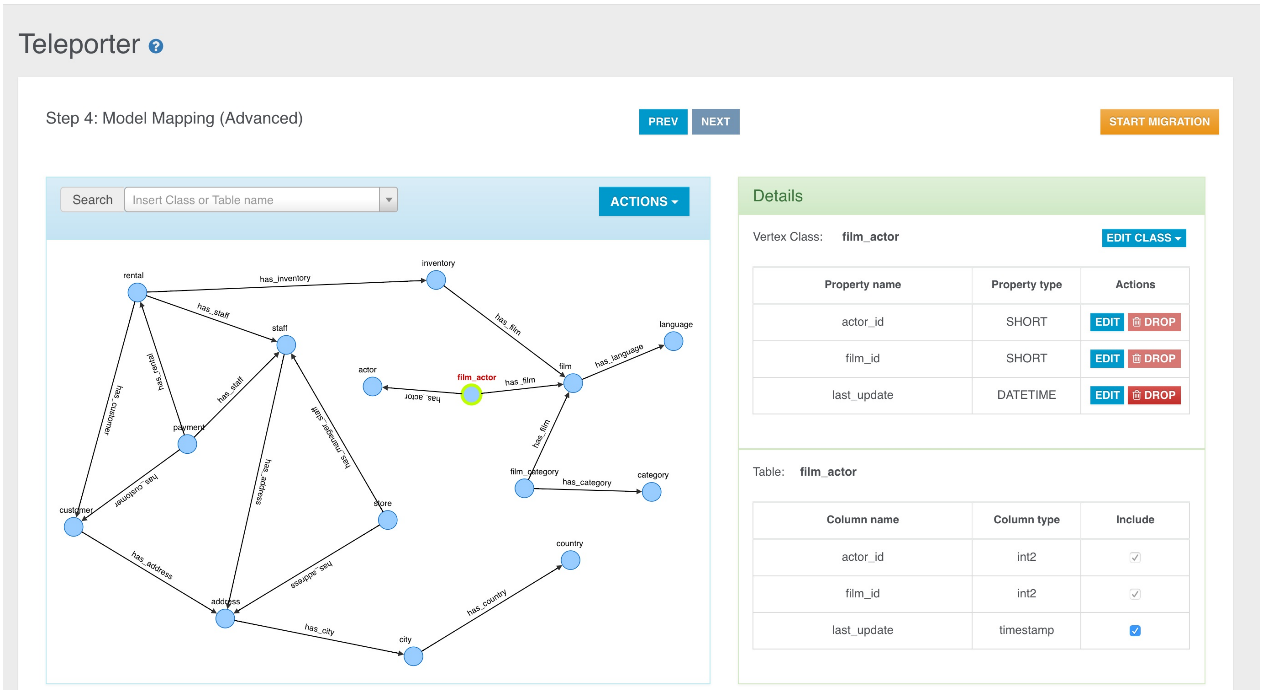
Let's have a deeper look at each of these operations.

Inspecting Classes and source correspondent elements

Via the Details Panel you can inspect information about:

1. Vertex class

If you select a Vertex Class, you can inspect the correspondence between each column in the source table and the correspondent property in the translated Vertex class. Columns and properties are strongly bound: you can exclude, include or rename a property, but the bindings with the correspondent column will remain.



Vertex Class: **film_actor**

Property name	Property type	Actions
actor_id	SHORT	EDIT DROP
film_id	SHORT	EDIT DROP
last_update	DATETIME	EDIT DROP

Table: **film_actor**

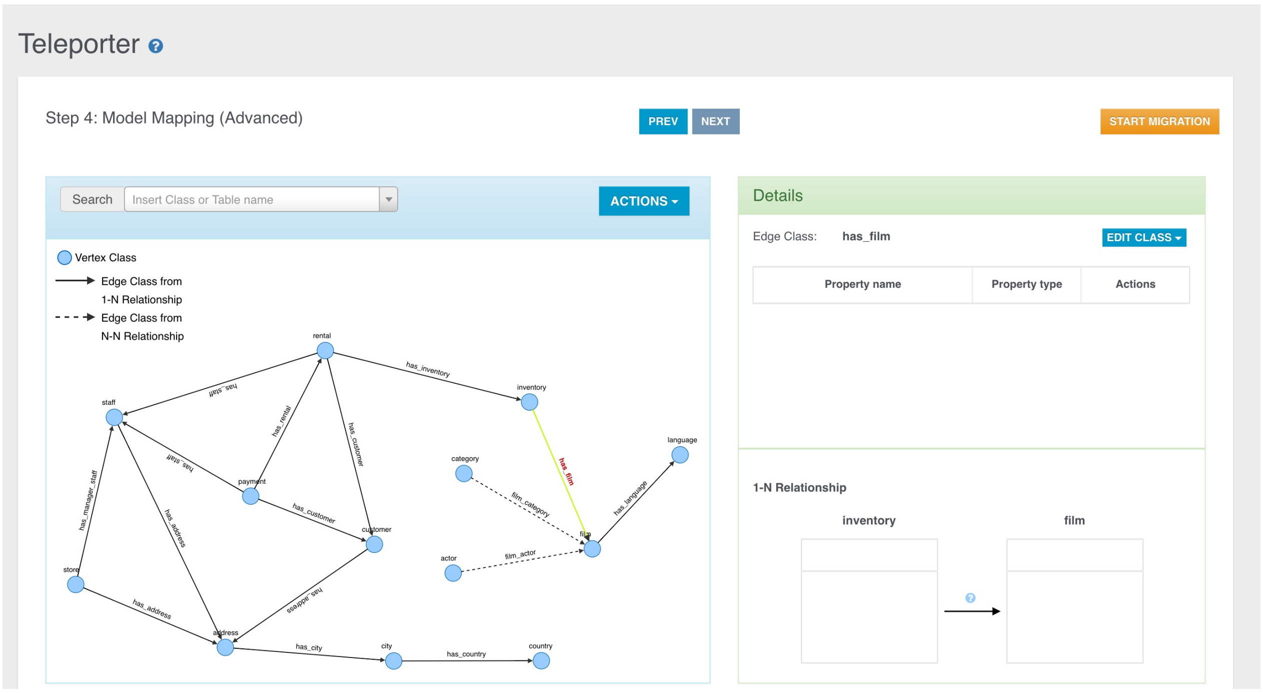
Column name	Column type	Include
actor_id	int2	✓
film_id	int2	✓
last_update	timestamp	✓

1. Edge Class

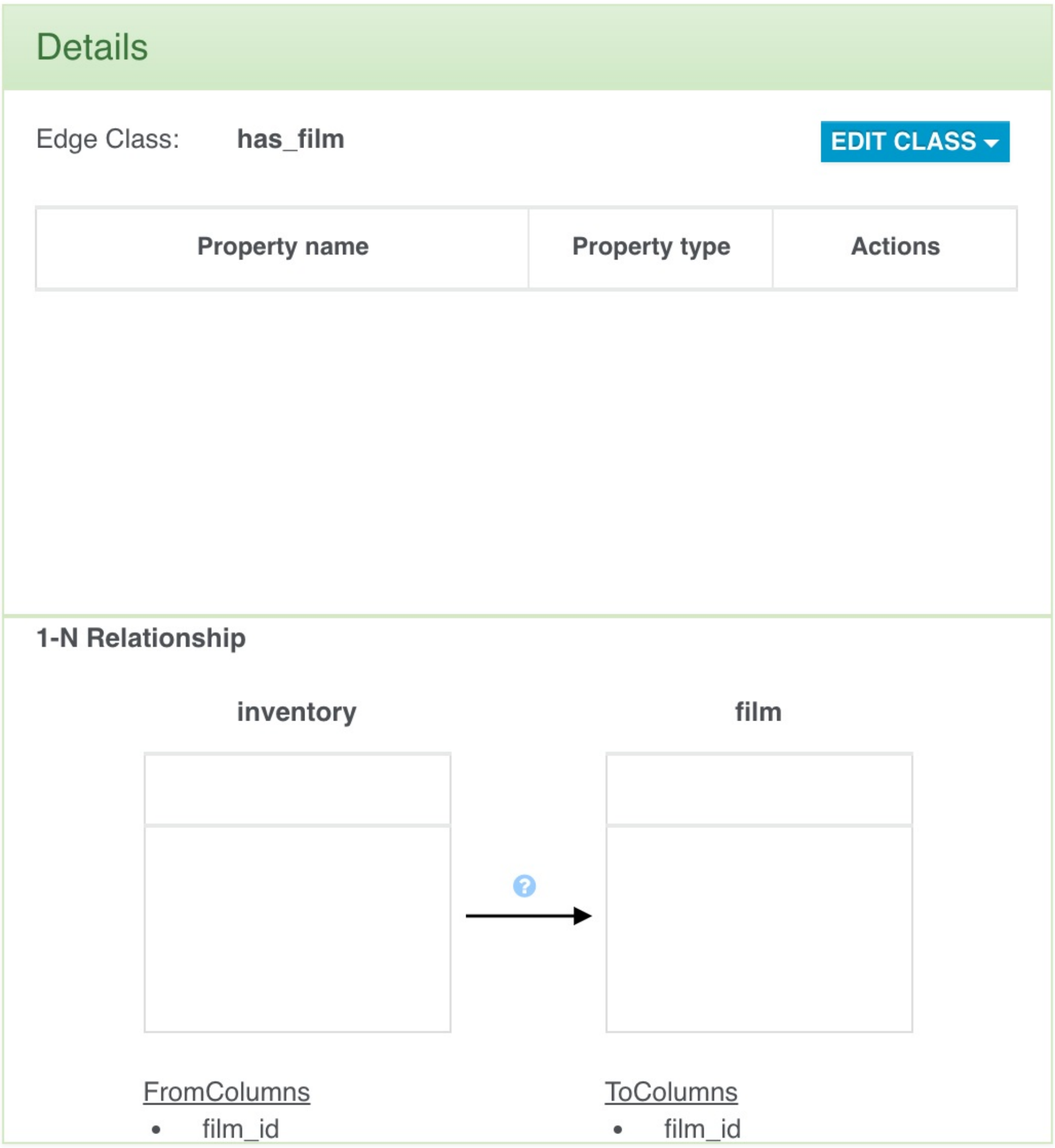
Everytime you select an edge in the graph, you can find out about the original relationship it comes from in the bottom section in the Details Panel. We can have 2 kinds of relationships, and coherently 2 kinds of edge rendering.

- 1-N Relationship

  Edges coming from 1-N Relationships are represented through a continous arrow.



The rendered Relationship involves just two tables of course, the starting table (aka **foreign table**) and the arrival table (aka **parent table**). Clicking the question mark you can also see for each table all the columns involved in the relationship.

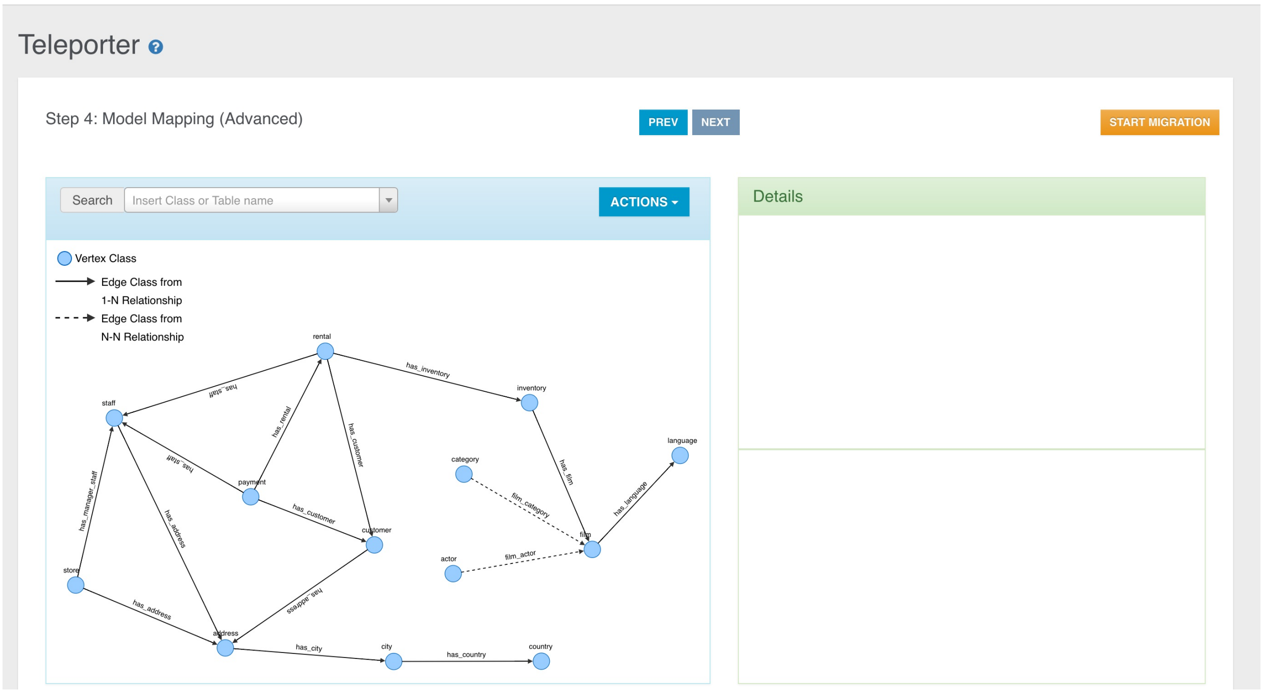


FromColumns
- film_id

ToColumns
- film_id

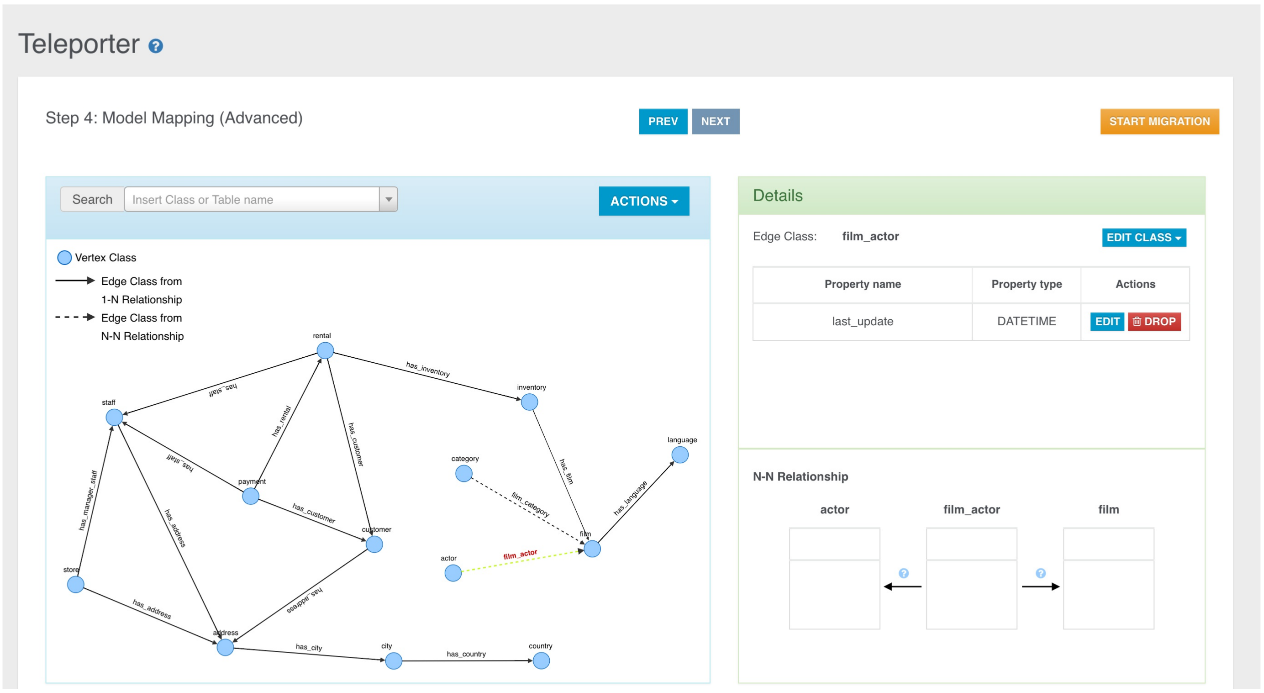
- N-N Relationship

Let's suppose you have got the following graph, obtained performing join tables aggregation through the naive-aggregate strategy.


Teleporter

Step 4: Model Mapping (Advanced)

Edges coming from N-N Relationships are represented through a dashed arrow and in the bottom you can see the 2 relationships involving two external tables and the join table between them.


Teleporter

Step 4: Model Mapping (Advanced)

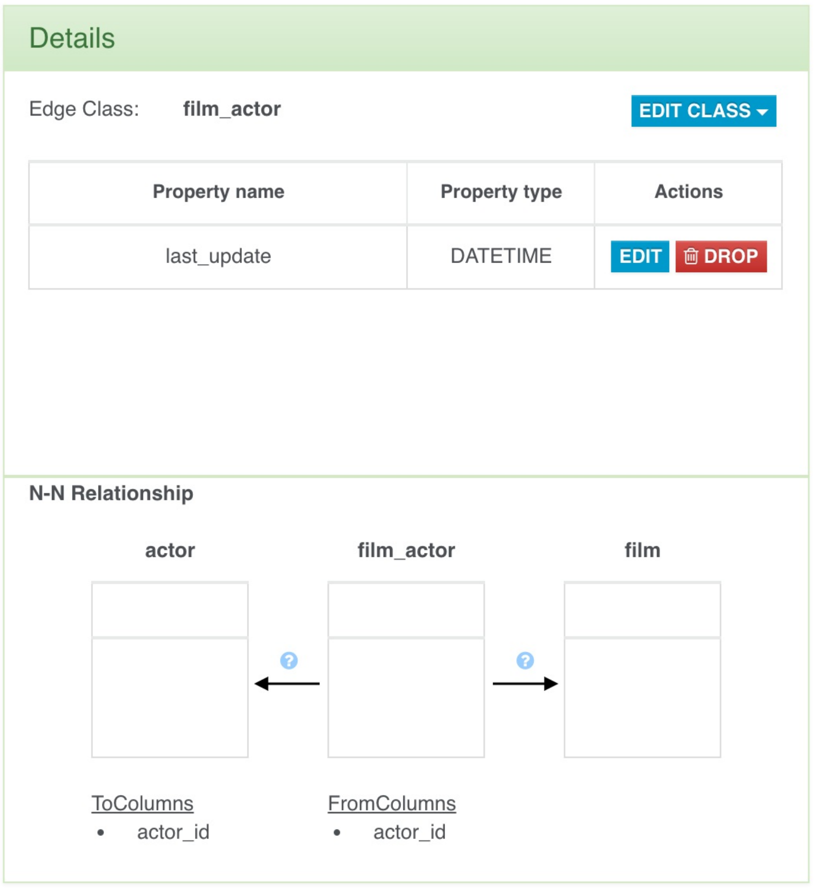
Details

Edge Class: film_actor

Property name	Property type	Actions
last_update	DATETIME	EDIT DROP

N-N Relationship

Here too, clicking the question mark you can inspect the involved columns for both the relationships.



Details

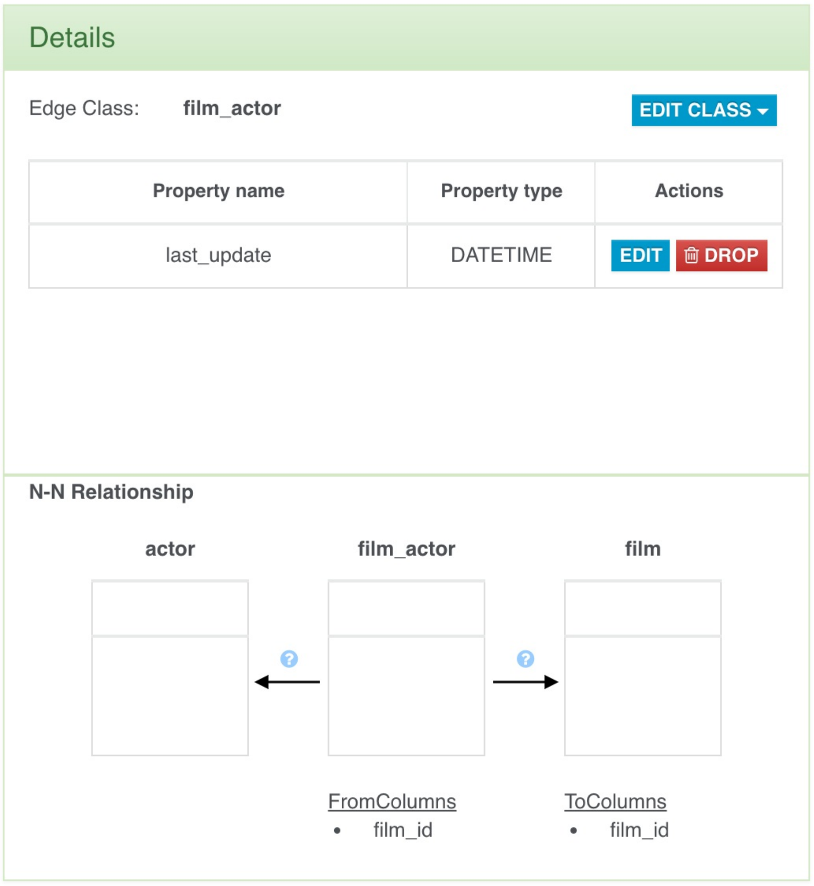
Edge Class: **film_actor**

Property name	Property type	Actions
last_update	DATETIME	EDIT DROP

N-N Relationship

ToColumns
- actor_id

FromColumns
- actor_id



Details

Edge Class: **film_actor**

Property name	Property type	Actions
last_update	DATETIME	EDIT DROP

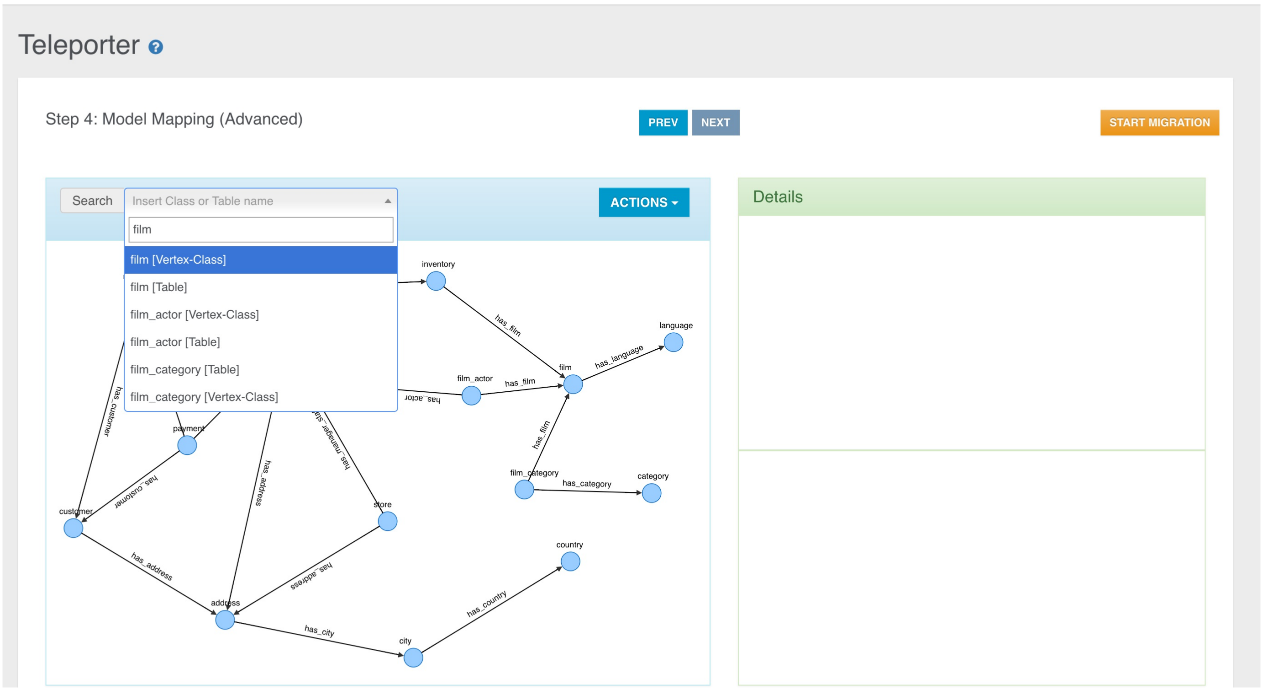
N-N Relationship

FromColumns
- film_id

ToColumns
- film_id

Search Bar

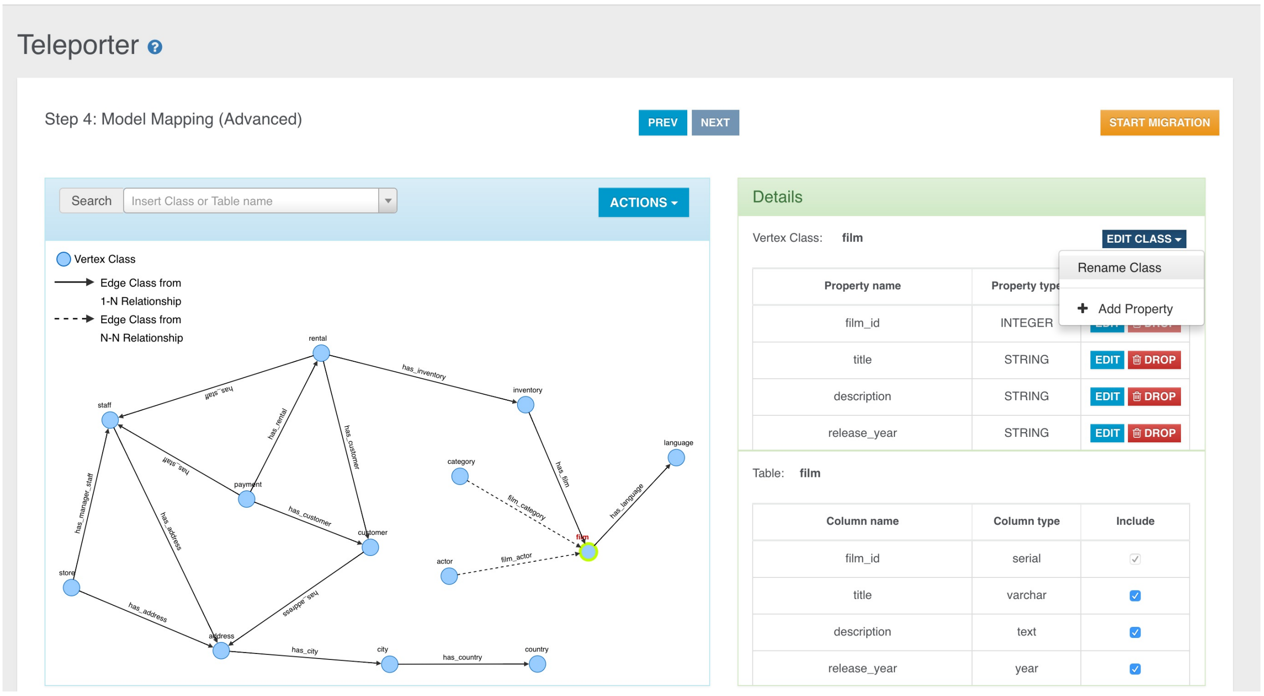
In the Graph Panel a useful search bar is provided to allow you fast vertex selection according to the vertex class name or the source table name.



In the example above you can see that for each class we have two items, the vertex class name and the source table name. In this case each couple of items are equal because no classes were renamed nor a name resolver was adoted during the basic graph model building.

Class Renaming

You can rename a class just selecting an element in the graph (vertex or edge) and clicking the "Rename Class" button in the "Edit Class" dropdown menu.



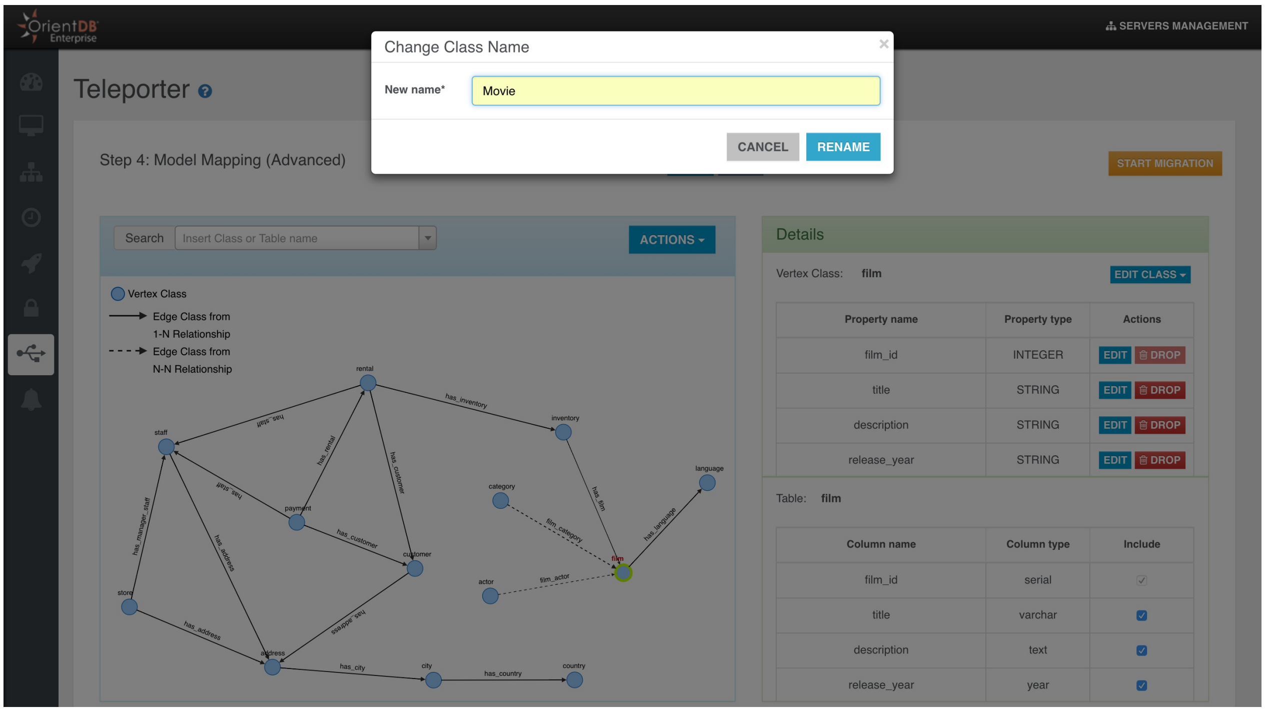
Vertex Class: **film**

Property name	Property type	
film_id	INTEGER	EDIT DROP
title	STRING	EDIT DROP
description	STRING	EDIT DROP
release_year	STRING	EDIT DROP

Table: **film**

Column name	Column type	Include
film_id	serial	✓
title	varchar	✓
description	text	✓
release_year	year	✓

Then you just have to choose the new name for the specific class.



Change Class Name

New name*: Movie

CANCEL | RENAME

Vertex Class: film

Property name	Property type	Actions
film_id	INTEGER	EDIT DROP
title	STRING	EDIT DROP
description	STRING	EDIT DROP
release_year	STRING	EDIT DROP

Table: film

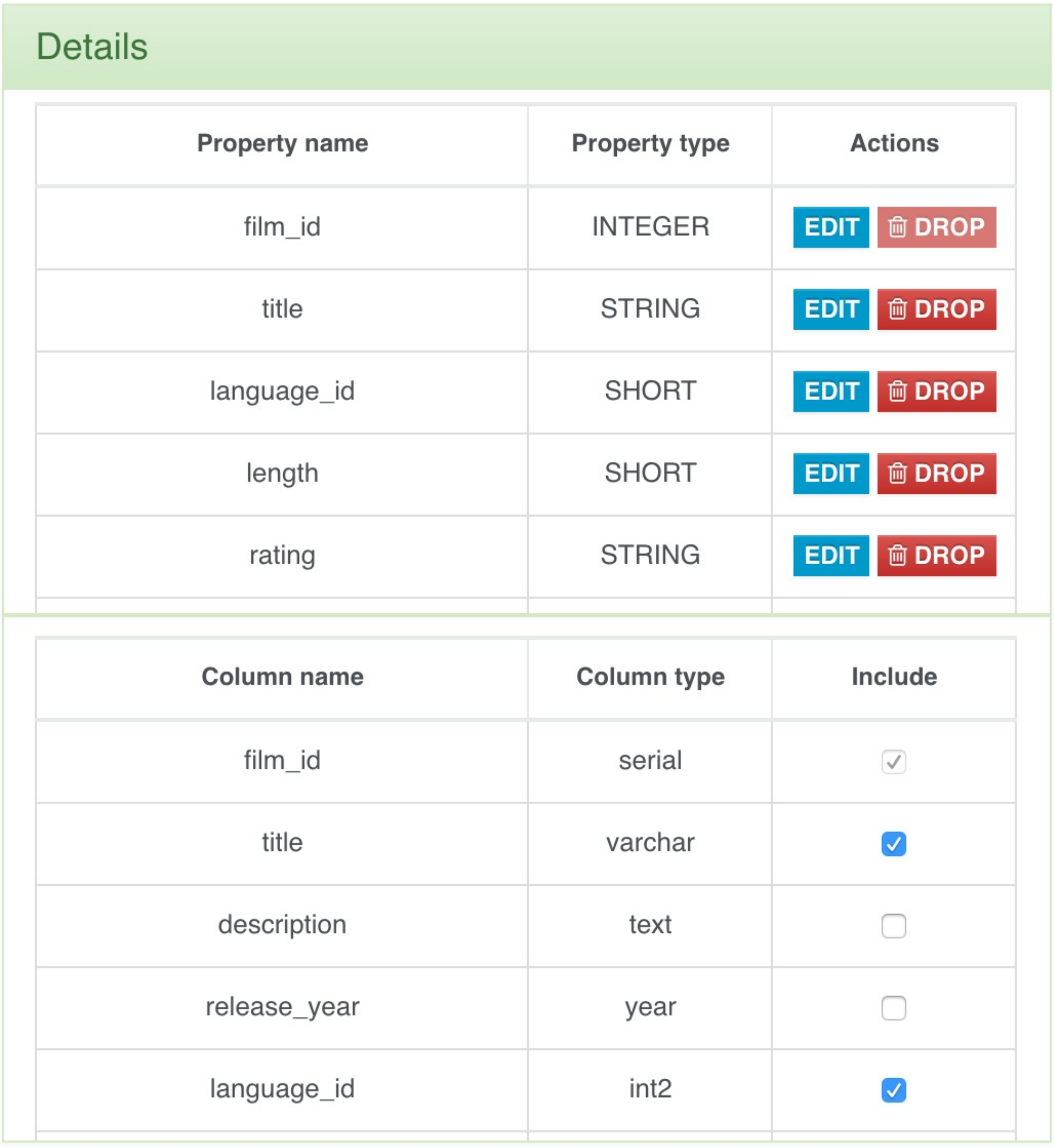
Column name	Column type	Include
film_id	serial	✓
title	varchar	✓
description	text	✓
release_year	year	✓

The class name will be updated in the graph, in the search bar and in the Details Panel of course.

Property Excluding

We have two ways to exclude a property mapped with a column in the source table:

1. Unflagging the correspondent column name in the source table perspective.
2. Dropping the property from the class perspective.



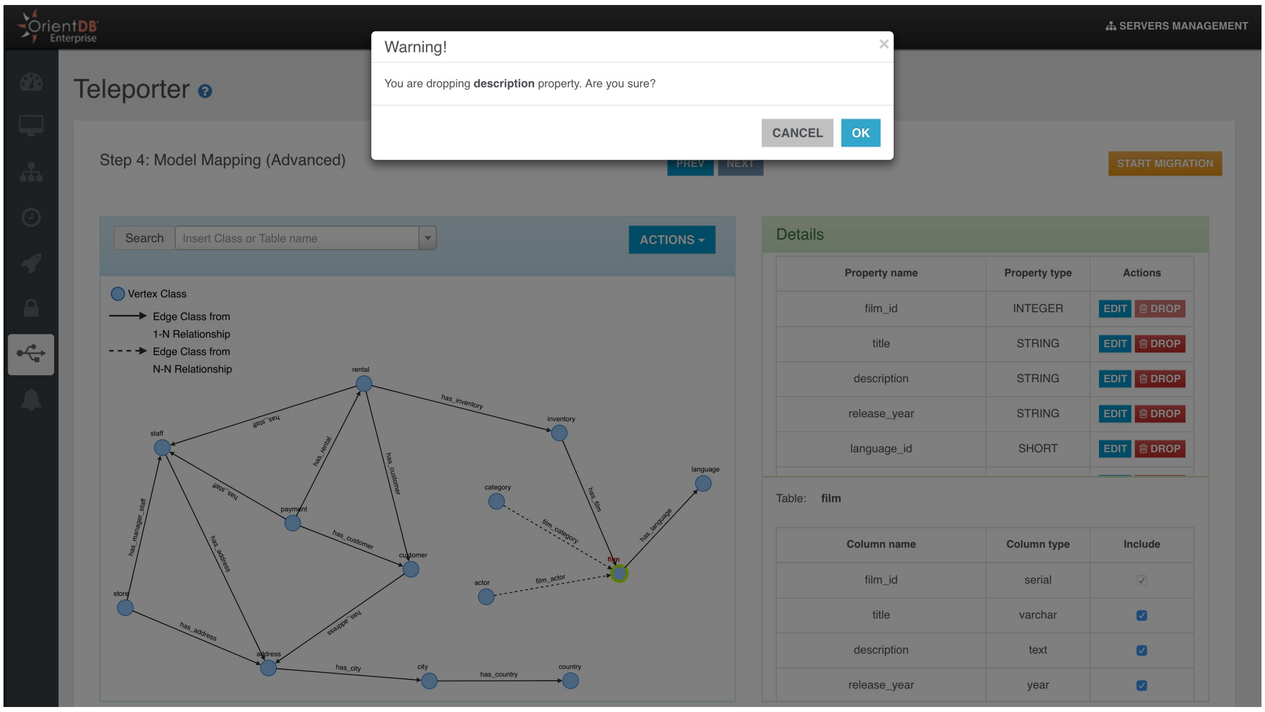
Details

Property name	Property type	Actions
film_id	INTEGER	EDIT DROP
title	STRING	EDIT DROP
language_id	SHORT	EDIT DROP
length	SHORT	EDIT DROP
rating	STRING	EDIT DROP

Column name	Column type	Include
film_id	serial	✓
title	varchar	✓
description	text	☐
release_year	year	☐
language_id	int2	✓

Property Dropping

You can also drop a property via the specific button.


OrientDB Enterprise
SERVERS MANAGEMENT
Teleporter
Step 4: Model Mapping (Advanced)
PREV
NEXT
START MIGRATION
Search
Insert Class or Table name
ACTIONS
Vertex Class
Edge Class from 1-N Relationship
Edge Class from N-N Relationship
Details

Warning!

You are dropping **description** property. Are you sure?

CANCEL OK

Property name	Property type	Actions
film_id	INTEGER	EDIT DROP
title	STRING	EDIT DROP
description	STRING	EDIT DROP
release_year	STRING	EDIT DROP
language_id	SHORT	EDIT DROP

Table: film

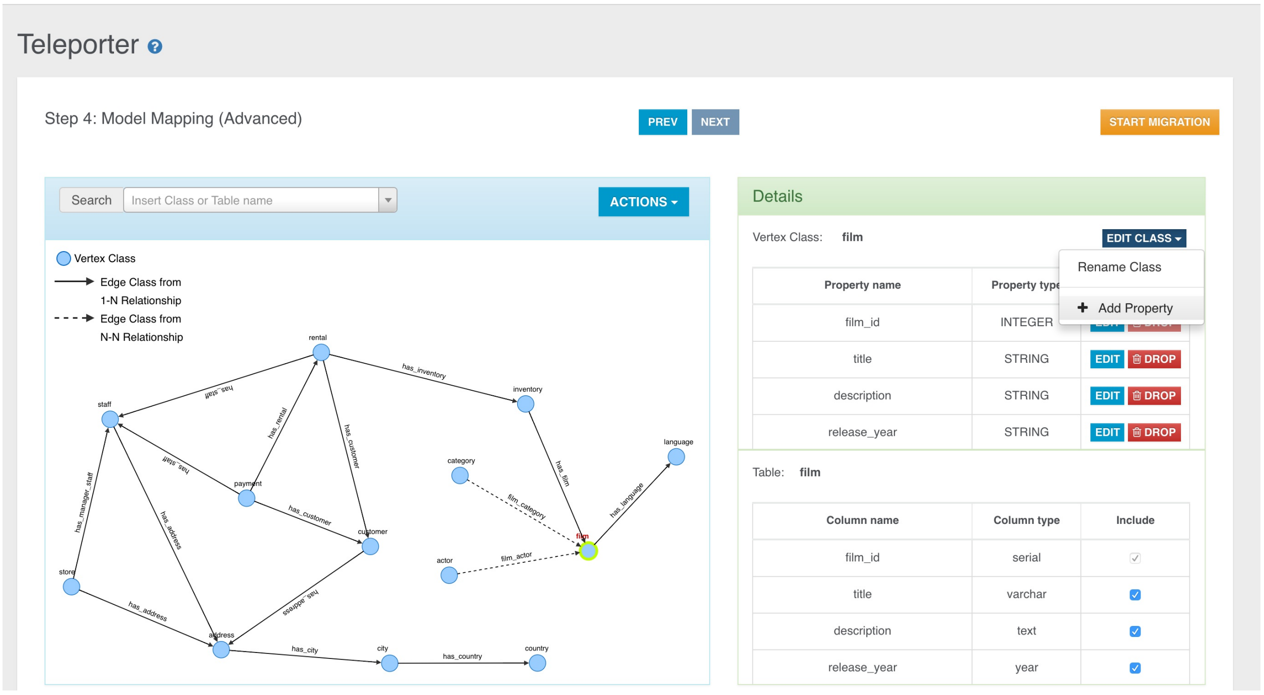
Column name	Column type	Include
film_id	serial	✓
title	varchar	✓
description	text	✓
release_year	year	✓

You can have 2 different behaviours depending on whether the property is bound with a column in the source table or not.

- If the property is bound with a source column, when you drop it you will get the same result as when you exclude it, so it will not be migrated in OrientDB but you can always include it again, as the binding is not deleted at all.
- If the property is not bound with a source column, then when you drop it the property will be definitively deleted.

Property Adding

You can add new properties just clicking the "Add property" button in the "Edit Class" dropdown menu.


Teleporter
Step 4: Model Mapping (Advanced)
PREV
NEXT
START MIGRATION
Search
Insert Class or Table name
ACTIONS
Vertex Class
Edge Class from 1-N Relationship
Edge Class from N-N Relationship
Details
EDIT CLASS
Rename Class
Add Property

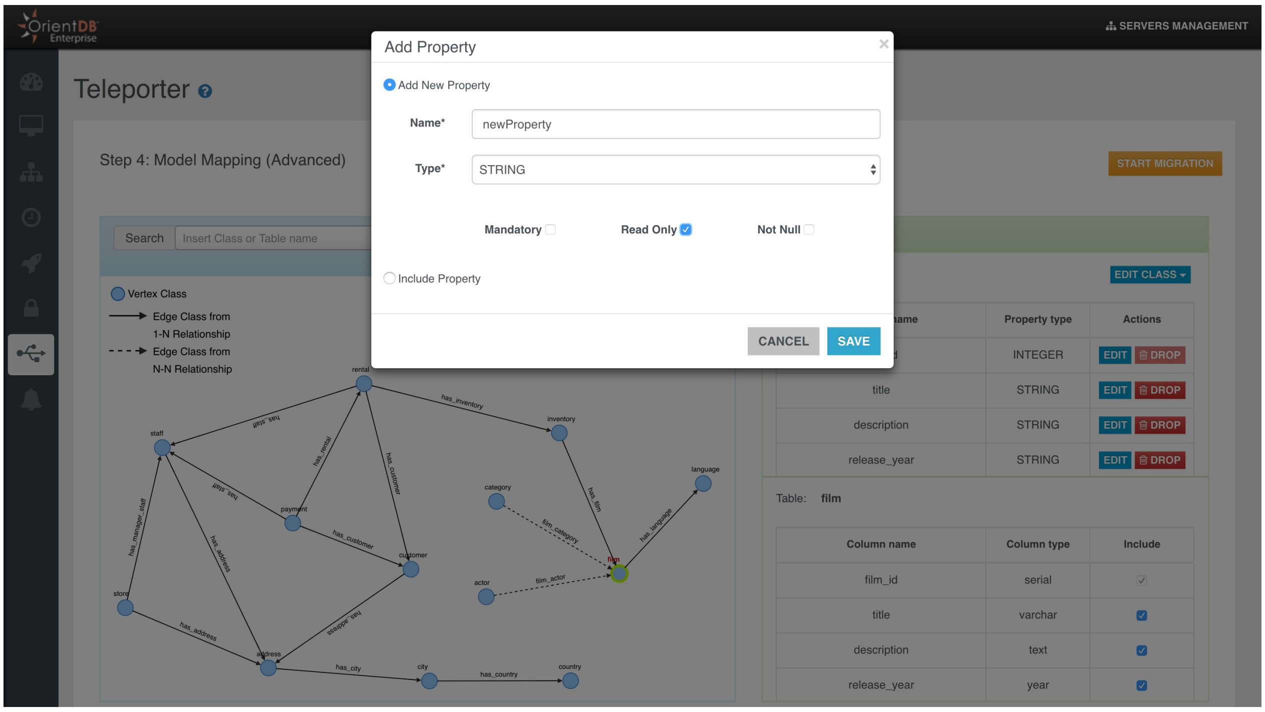
Vertex Class: film

Property name	Property type	
film_id	INTEGER	EDIT DROP
title	STRING	EDIT DROP
description	STRING	EDIT DROP
release_year	STRING	EDIT DROP

Table: film

Column name	Column type	Include
film_id	serial	✓
title	varchar	✓
description	text	✓
release_year	year	✓

In the just opened window you can choose to add a new property never defined before, selecting the "Add new property" radio button,



Add Property

[✓ Add New Property]

Name*: newProperty

Type*: STRING

Mandatory ☐ Read Only ✓ Not Null ☐

Include Property

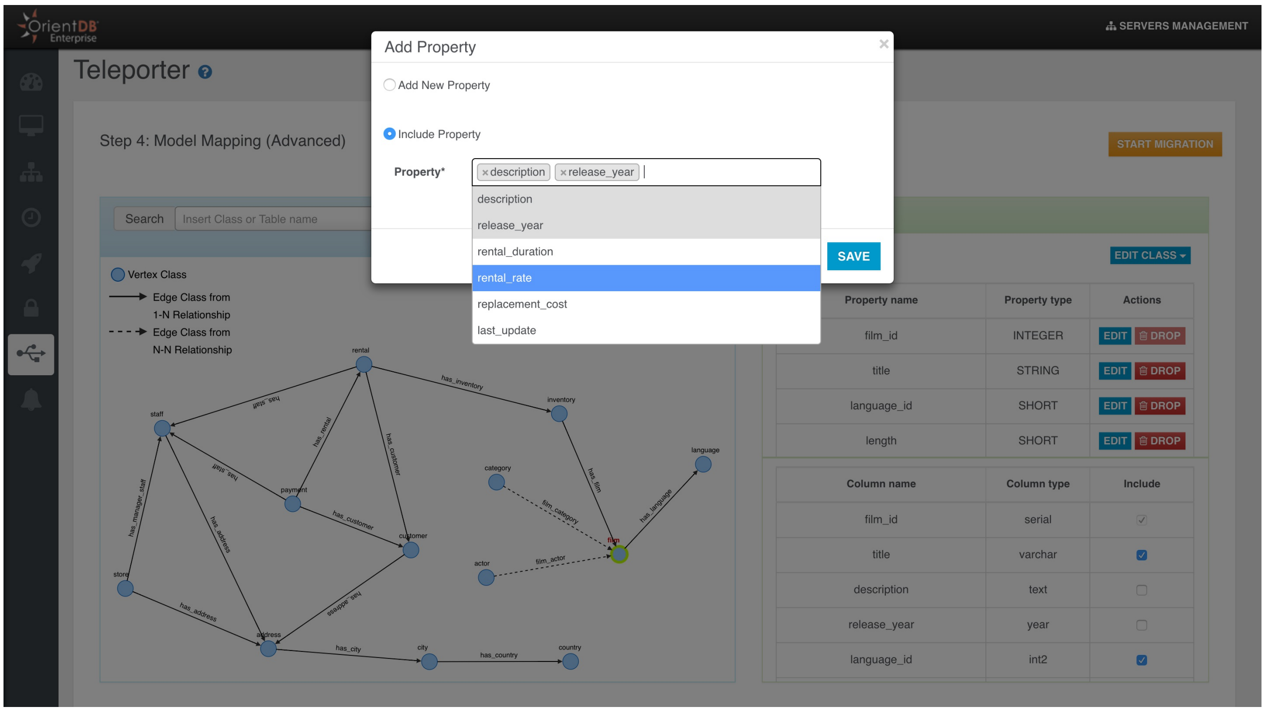
CANCEL SAVE

Property name	Property type	Actions
...d	INTEGER	EDIT DROP
title	STRING	EDIT DROP
description	STRING	EDIT DROP
release_year	STRING	EDIT DROP

Table: film

Column name	Column type	Include
film_id	serial	✓
title	varchar	✓
description	text	✓
release_year	year	✓

or re-include some excluded properties if any, selecting the "Include Property" radio button.



Add Property

Add New Property

[✓ Include Property]

Property*: ×description ×release_year

- description
- release_year
- rental_duration
- rental_rate
- replacement_cost
- last_update

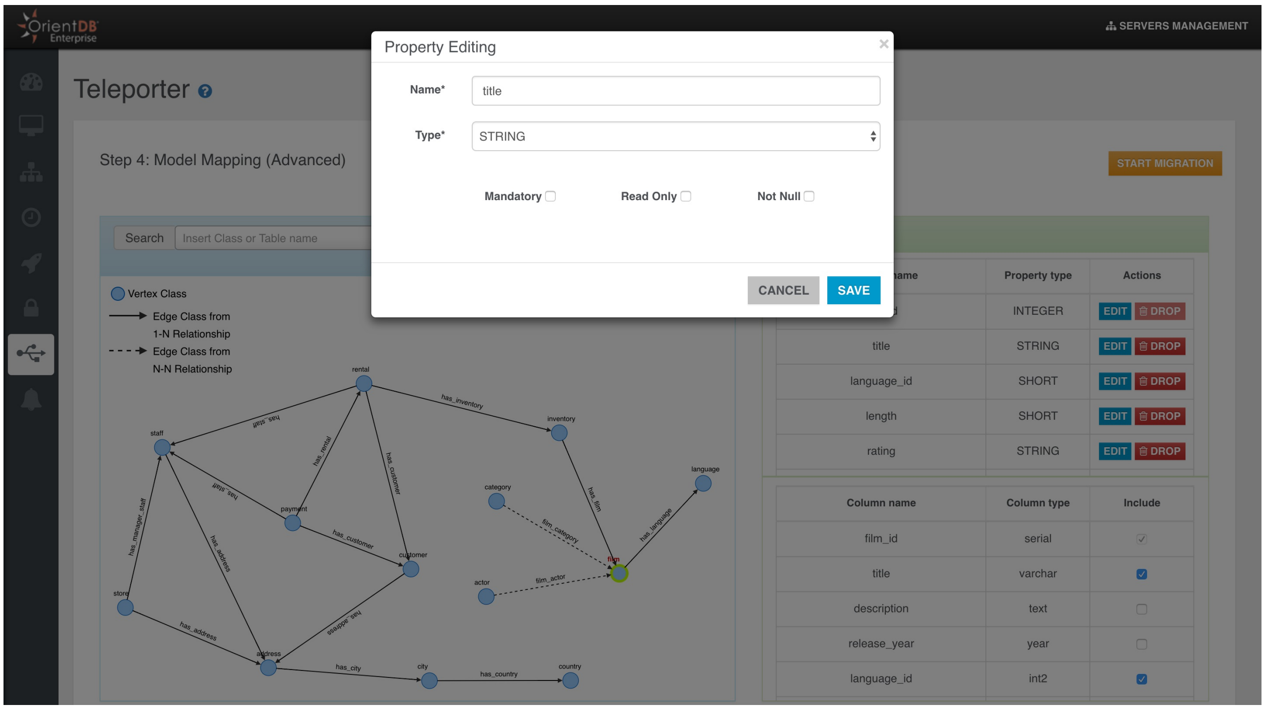
SAVE

Property name	Property type	Actions
film_id	INTEGER	EDIT DROP
title	STRING	EDIT DROP
language_id	SHORT	EDIT DROP
length	SHORT	EDIT DROP

Column name	Column type	Include
film_id	serial	✓
title	varchar	✓
description	text	☐
release_year	year	☐
language_id	int2	✓

Property Editing

You can also edit an existing property: you can choose a different name, type, or just add/remove some constraints.



Property Editing

Name* title

Type* STRING

Mandatory ☐ Read Only ☐ Not Null ☐

CANCEL SAVE

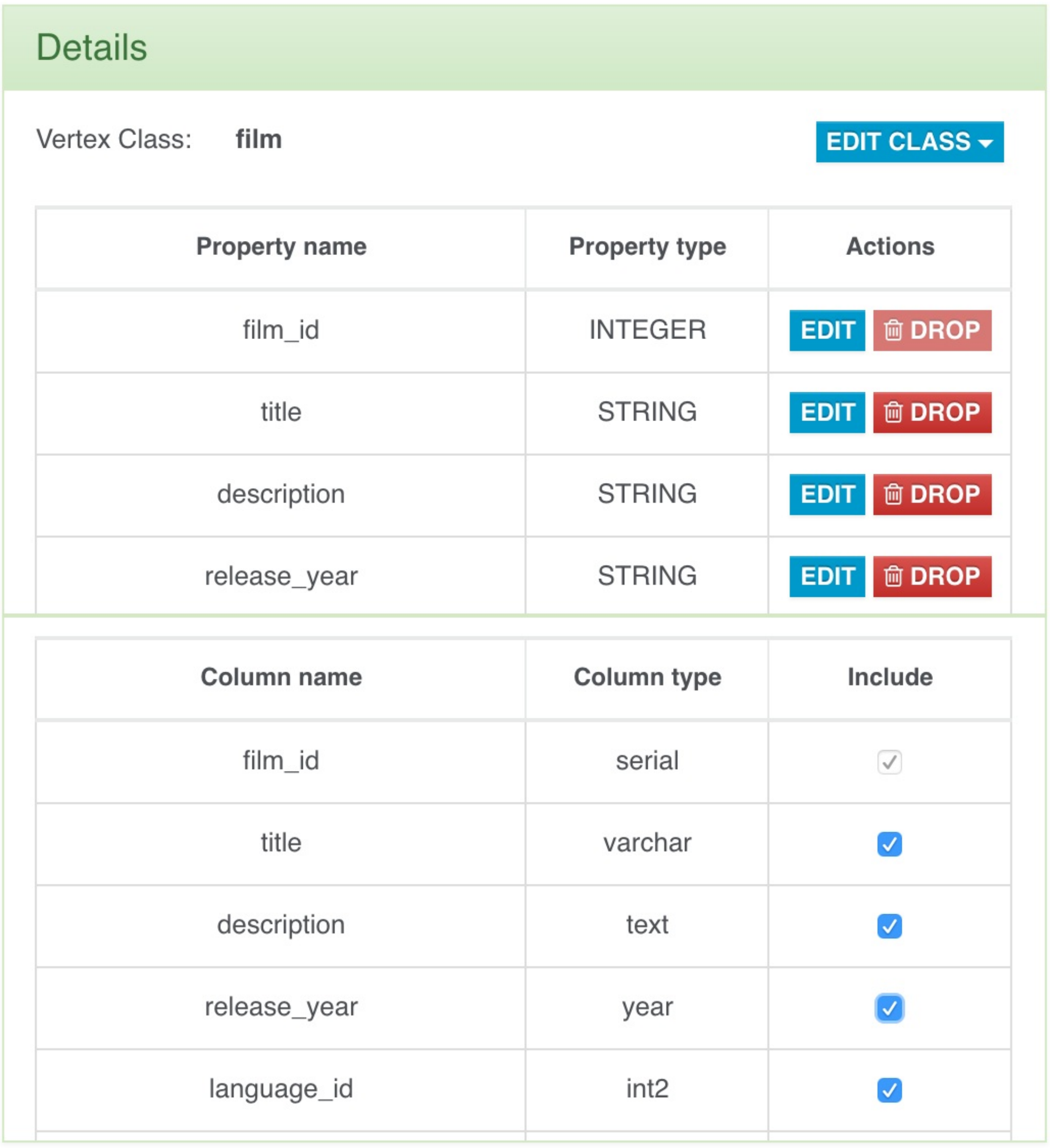
ame	Property type	Actions
	INTEGER	EDIT DROP
title	STRING	EDIT DROP
language_id	SHORT	EDIT DROP
length	SHORT	EDIT DROP
rating	STRING	EDIT DROP

Column name	Column type	Include
film_id	serial	✓
title	varchar	✓
description	text	☐
release_year	year	☐
language_id	int2	✓

Property Including

We have two ways to include a property mapped with a column in the source table:

1. Flagging the correspondent column name in the source table perspective.
2. Including the property from the OrientDB class perspective through the "Add property" button as shown above.



Details

Vertex Class: **film**

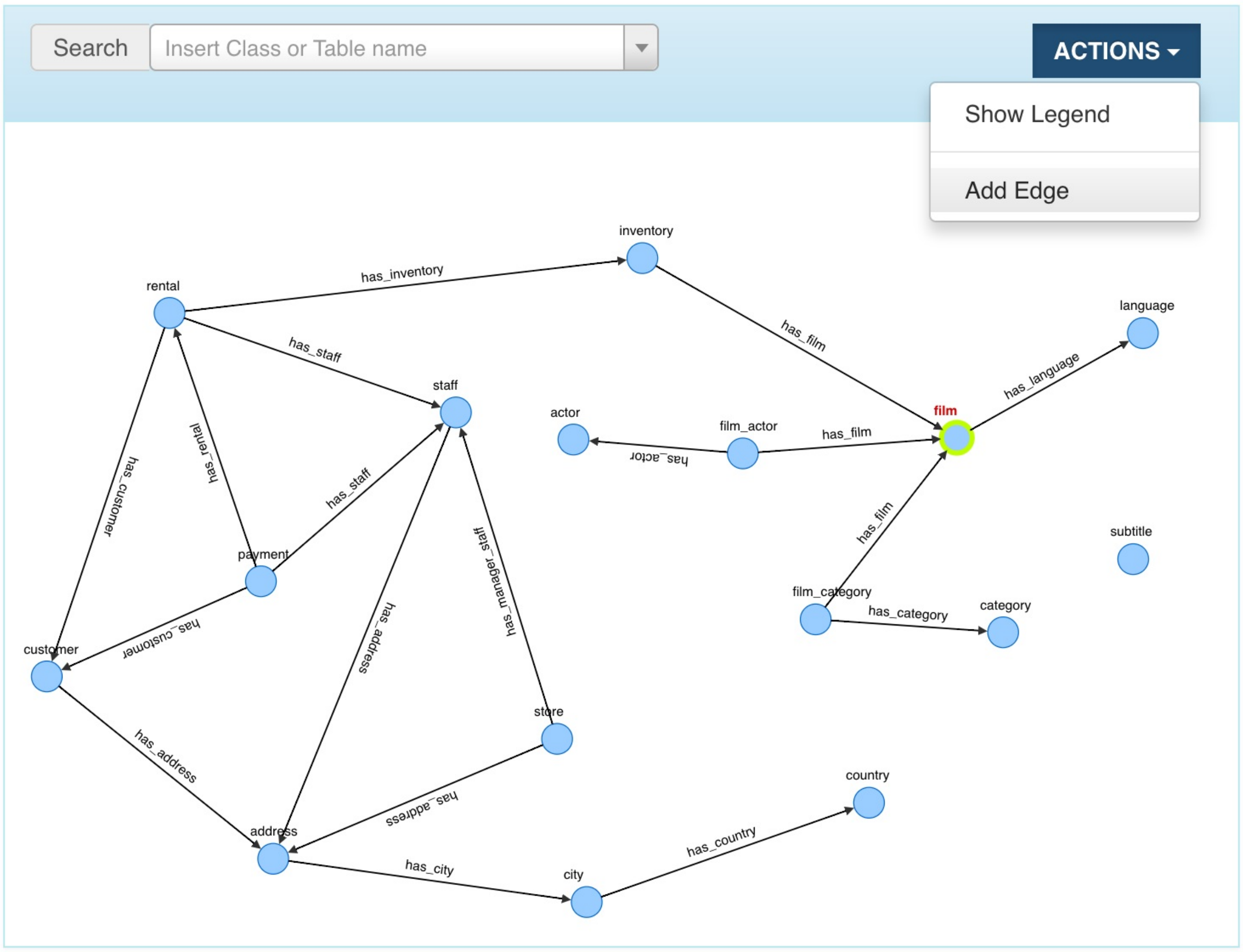
EDIT CLASS

Property name	Property type	Actions
film_id	INTEGER	EDIT DROP
title	STRING	EDIT DROP
description	STRING	EDIT DROP
release_year	STRING	EDIT DROP

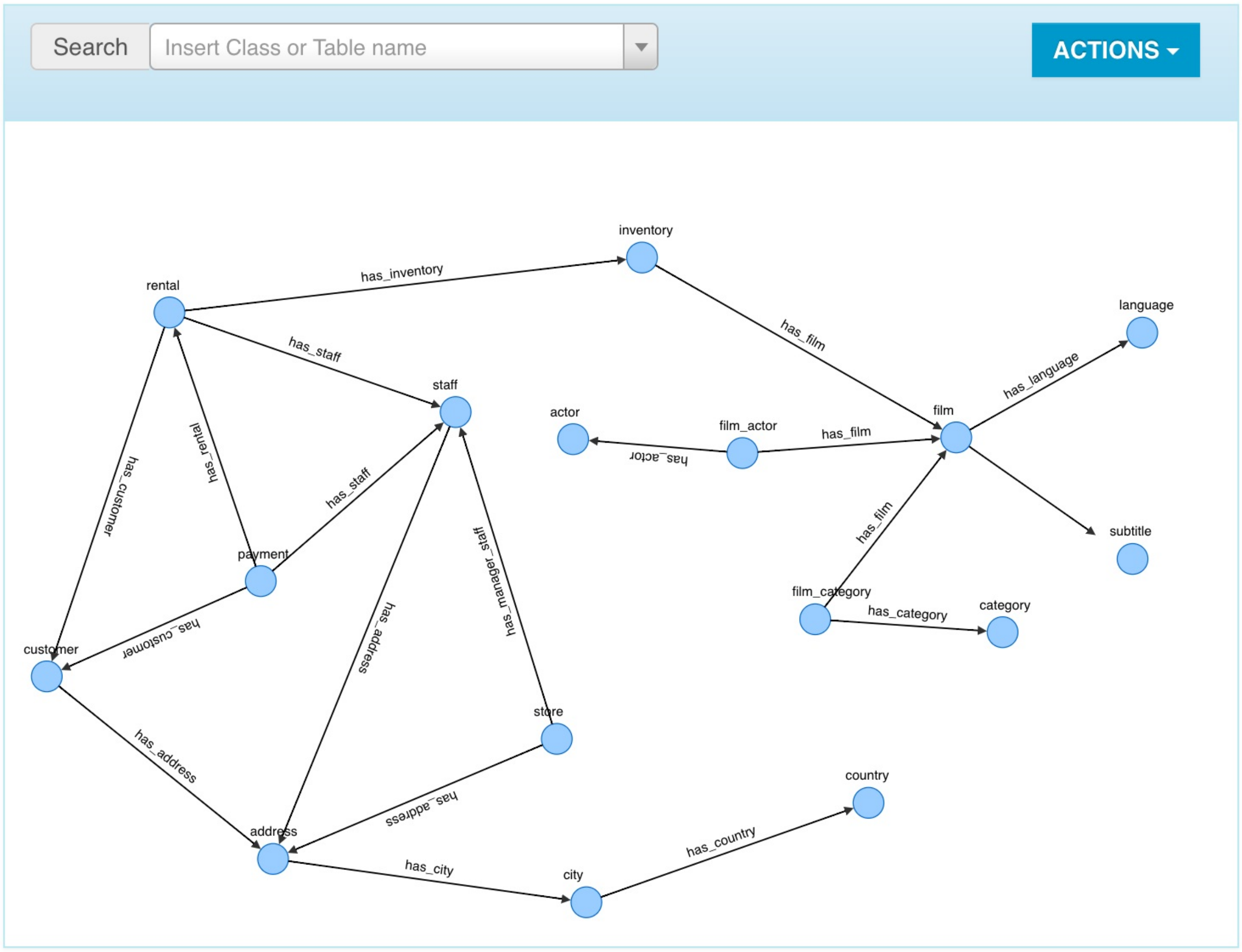
Column name	Column type	Include
film_id	serial	✓
title	varchar	✓
description	text	✓
release_year	year	✓
language_id	int2	✓

Edge Adding

Often you need to add an edge in your graph model, if it's missing for some reason. For example, if you didn't defined some foreign keys between the tables on which you usually perform join operations, you will lose this kind of info during the importing process and you will not have any edges in your final Graph Database. Sometimes you just want to enrich the model adding new edges. In both the cases you have to select a vertex in the graph and then click the "Add Edge" button in the "Action" dropdown menu.



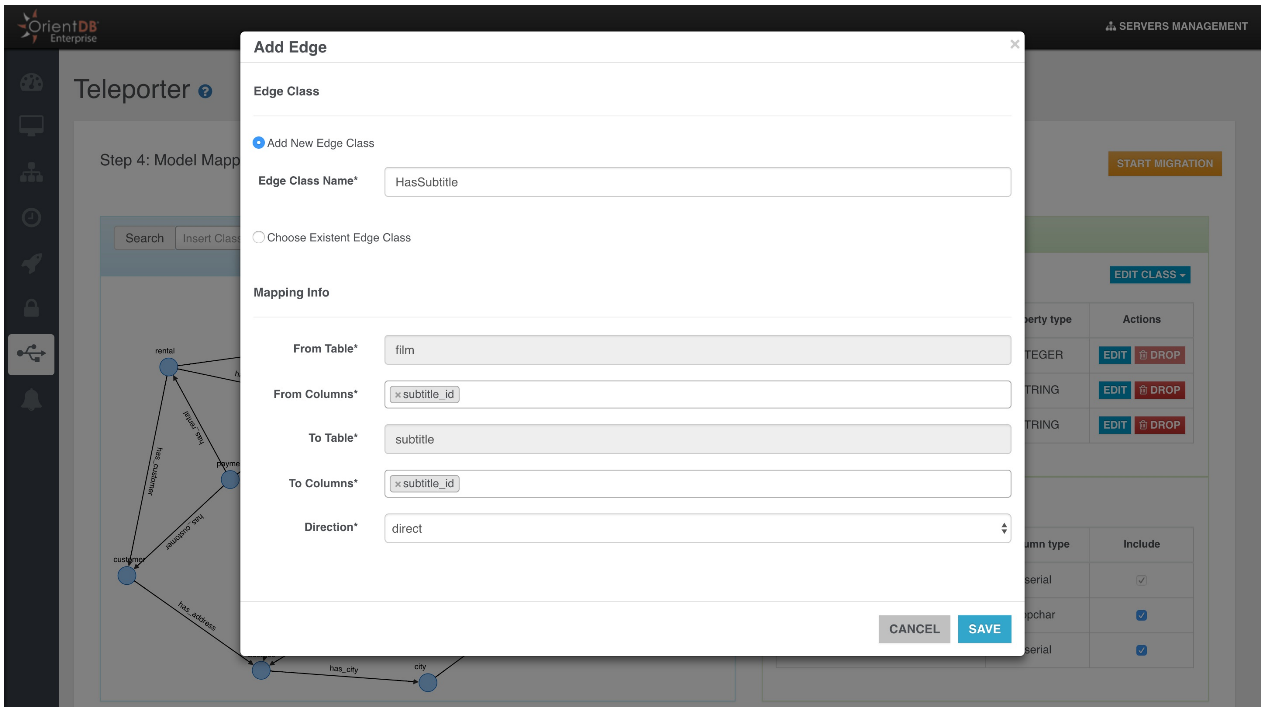
Then you have to drag the edge till the target vertex and click over it.



A new window will open where you have to specifiy the name of the Edge class for the new edge instance and some mapping info:

- fromTable: the foreign table that imports the primary key of the parent table.
- fromColumns: the attributes involved in the foreign key.
- toTable: the parent table whose primary key is imported by the foreign table.
- toColumns: the attributes involved in the imported primary key.

As said above, when we want to create a new edge instance, we can create a new Edge class


Add Edge

Edge Class

[✓ Add New Edge Class]

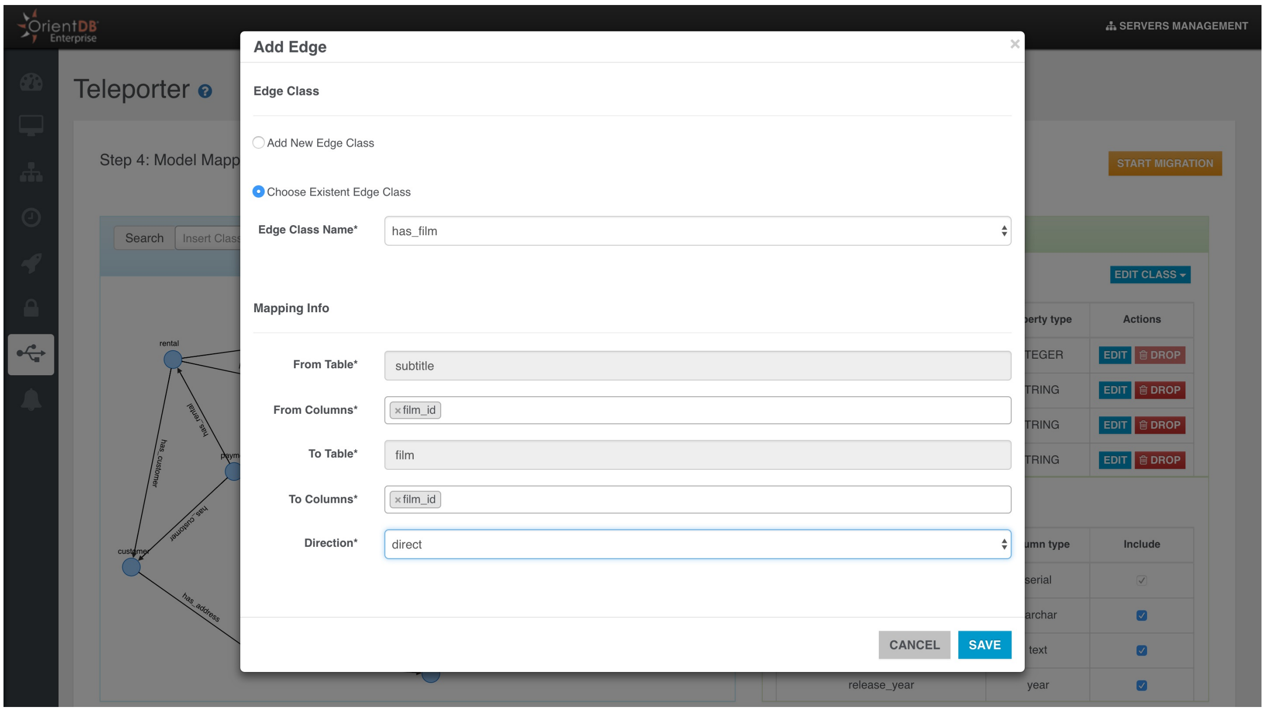
Edge Class Name*: HasSubtitle

Choose Existent Edge Class

Mapping Info

From Table*	film
From Columns*	× subtitle_id
To Table*	subtitle
To Columns*	× subtitle_id
Direction*	direct

or just choose a preexisting Edge class


Add Edge

Edge Class

Add New Edge Class

[✓ Choose Existent Edge Class]

Edge Class Name*: has_film

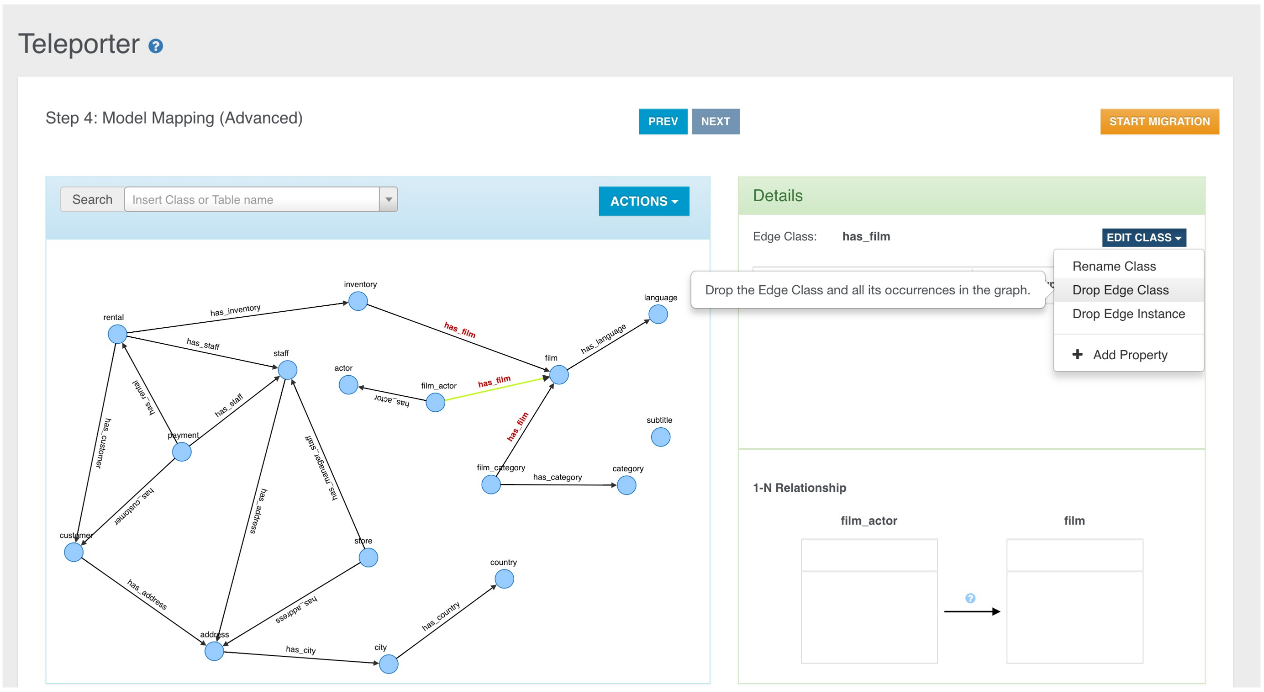
Mapping Info

From Table*	subtitle
From Columns*	× film_id
To Table*	film
To Columns*	× film_id
Direction*	direct

Edge Dropping

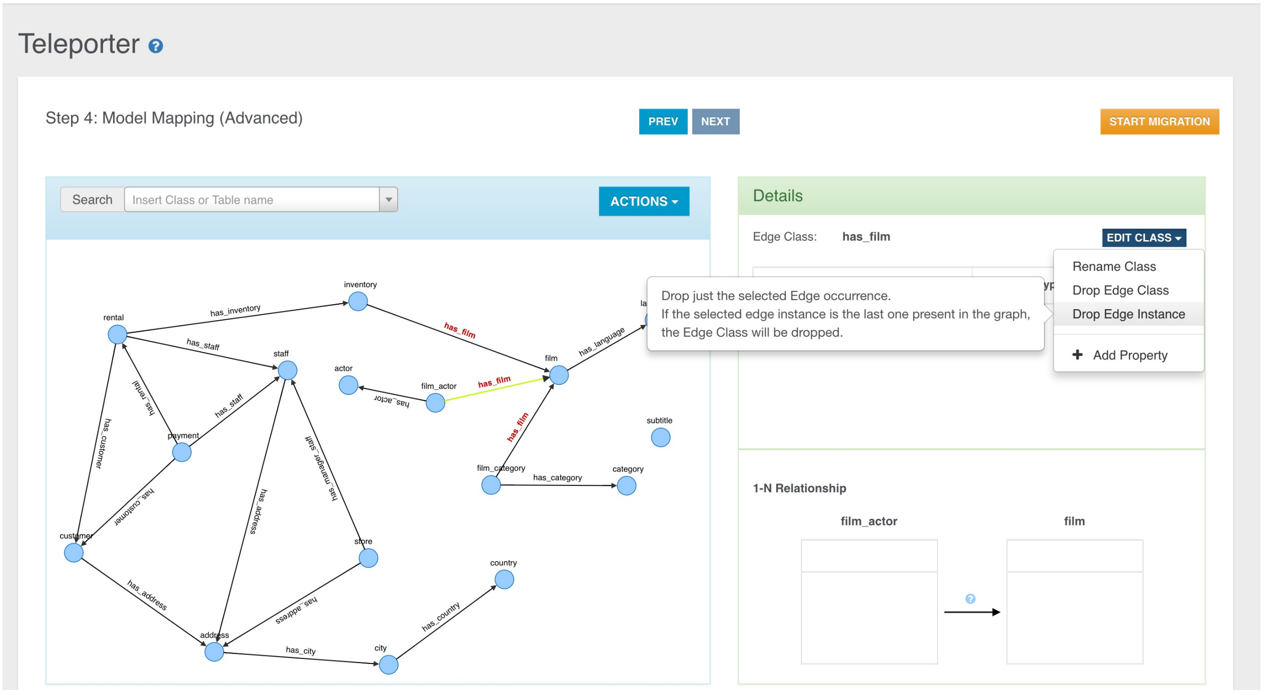
When you select an edge in the graph model, you have 2 choices:

1. Delete the Edge class with all its instances



Drop the Edge Class and all its occurrences in the graph.

1. Delete only the selected instance of the specific Edge Class



Drop just the selected Edge occurrence.
If the selected edge instance is the last one present in the graph, the Edge Class will be dropped.

Neo4j to OrientDB Importer

In Studio 2.2 you can configure the execution of a new plugin, which allows you to import your Neo4j database into OrientDB in few simple steps.

Imported neo4j items are:

- nodes
- relationships
- unique constraints
- indexes

NOTE: This feature is available both for the OrientDB Enterprise Edition and the OrientDB Community Edition.

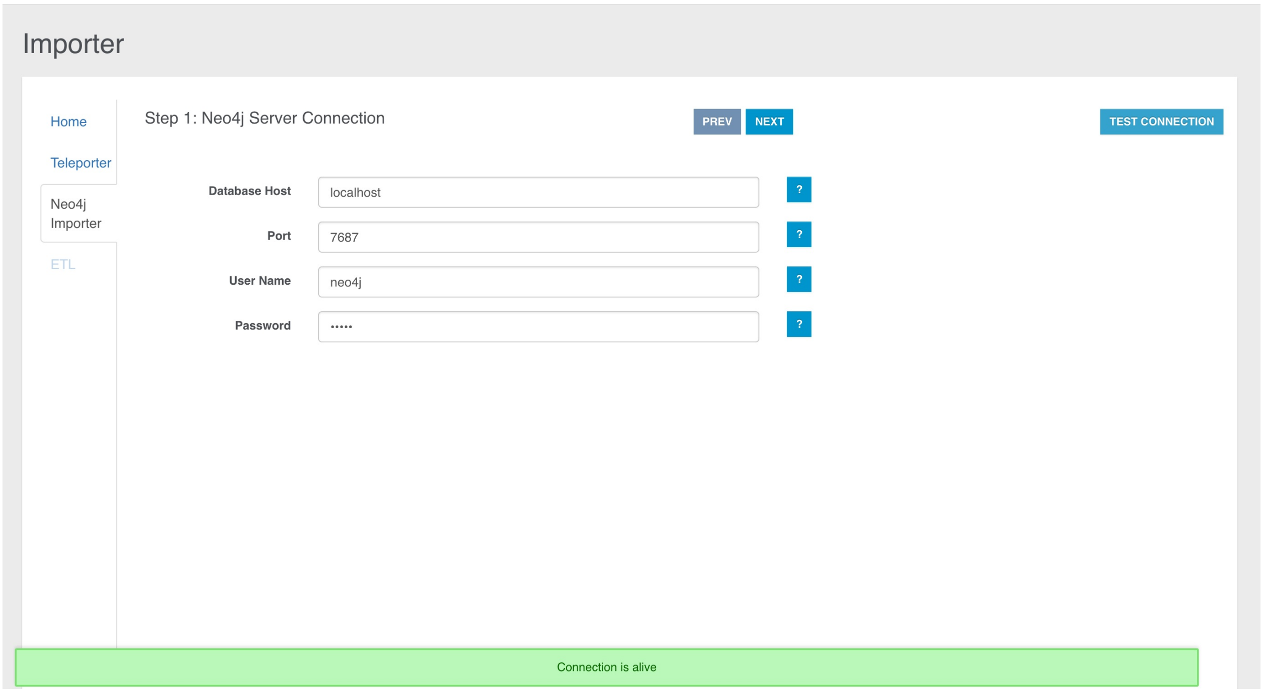
This visual tool consists in a wizard composed of 2 simple steps. Let's have a look at each configuration step.

Step 1

In the first step you have to type the following required parameters:

- `Database Host` , as the address of the host where the neo4j server is available
- `Port` , as the port where your neo4j server is listening for new connections via the bolt binary protocol (default port is 7687)
- `User Name` , as the username to access the neo4j server
- `Password` , as the password to access the neo4j server

After you typed all the required parameters for the migration you can test the connection with the source database.



Step 1: Neo4j Server Connection

Database Host: localhost
Port: 7687
User Name: neo4j
Password: •••••

Connection is alive

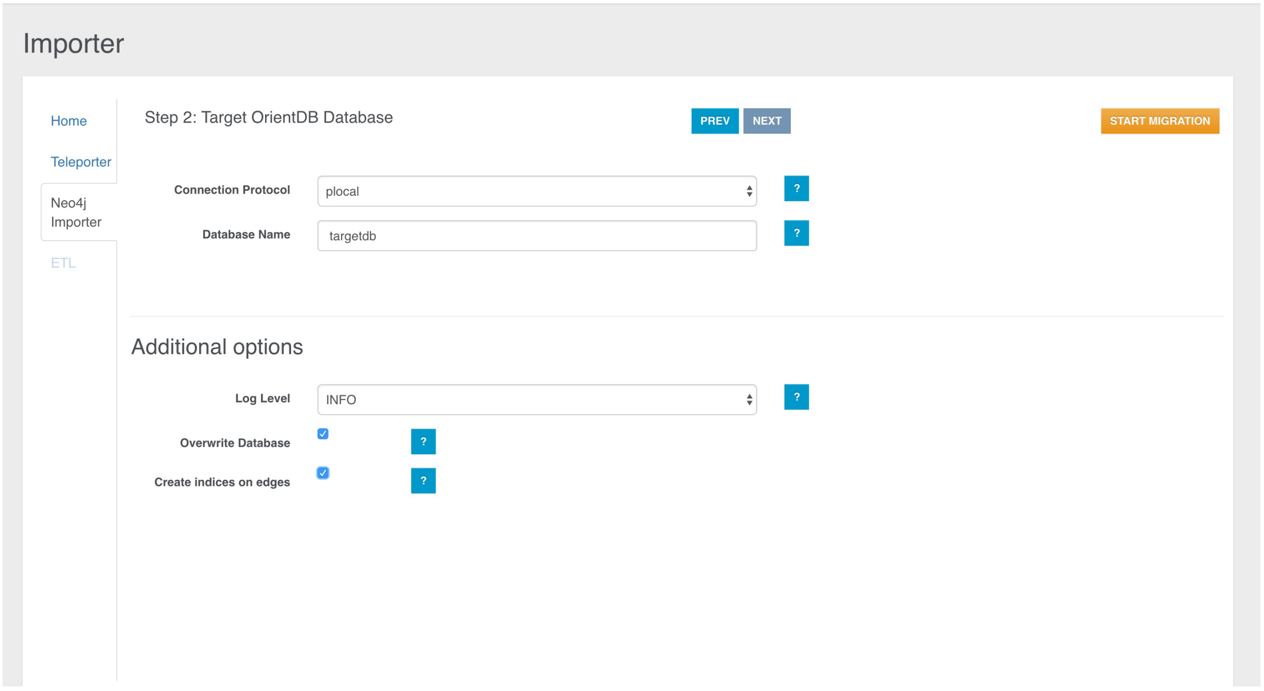
Step 2

In the second step you have to specify the parameters about the OrientDB target database and some additional info:

- `Connection protocol` , as the protocol adopted during the migration in order to connect to OrientDB. You have to choose among:
    - plocal: persistent disk-based, where the access is made in the same JVM process
    - memory: all data remain in memory
- `OrientDB Database Name` , as the target database name where the Neo4j database will be migrated. The database will be created by the import tool if not present. In case the database already exists, the Neo4j to OrientDB Importer will behave accordingly to the checkbox below.
- `Log Level` , as the level of verbosity printed to the output during the execution. You can choose among:

- NO
- DEBUG
- INFO
- WARNING
- ERROR

- `Overwrite Database`, checkbox to overwrite OrientDB target database if it already exists.
- `Create indices on edges`, checkbox to create indices on imported edges in OrientDB. In this way an index will be built for each Edge class on `'Neo4jRelId'` property.



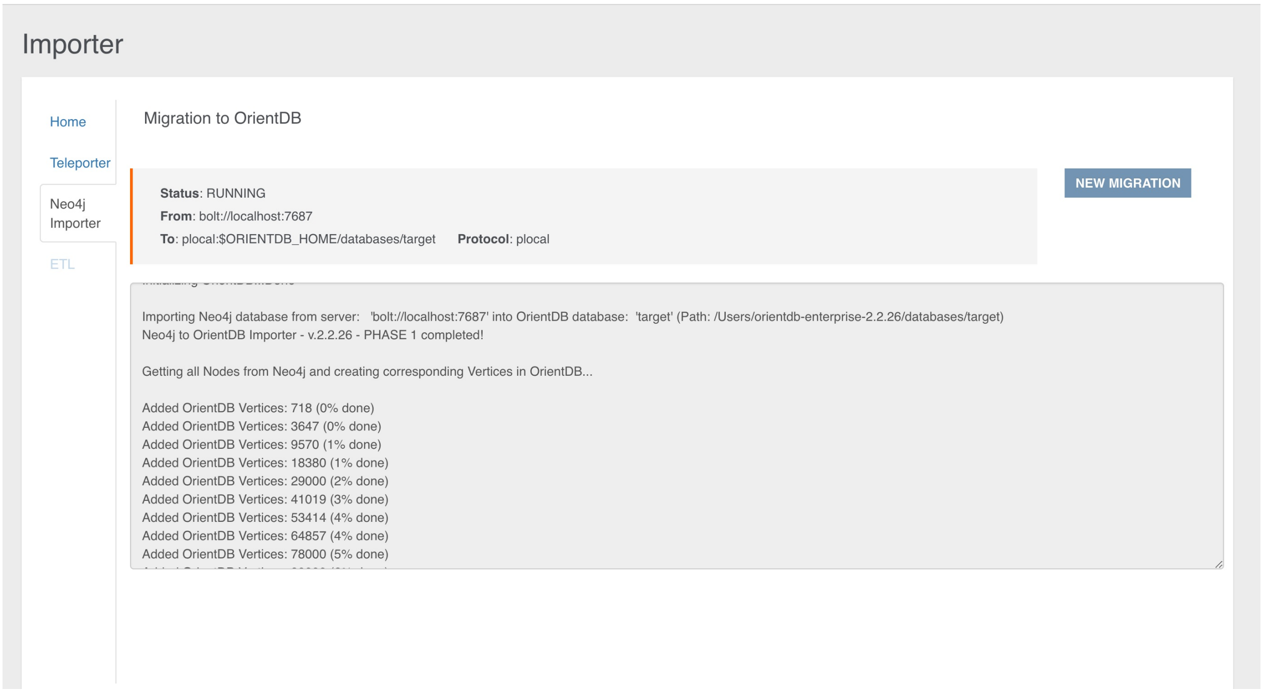
Step 2: Target OrientDB Database

Connection Protocol: plocal
Database Name: targetdb

Additional options

Log Level: INFO
Overwrite Database: ✓
Create indices on edges: ✓

Once we have collected all the info, you can run your configured job through the `START MIGRATION` button, then the job progress monitor will be displayed:

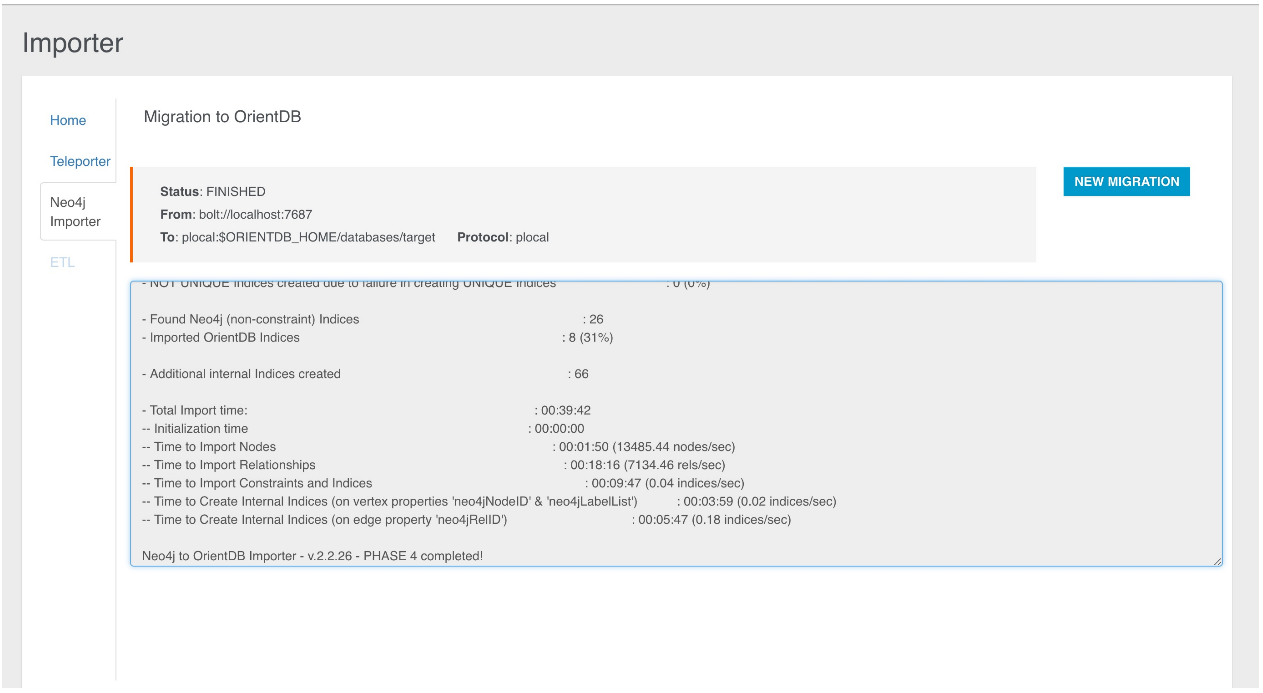


Migration to OrientDB

Status: RUNNING
From: bolt://localhost:7687
To: plocal:$ORIENTDB_HOME/databases/target **Protocol**: plocal

```
Importing Neo4j database from server:  'bolt://localhost:7687' into OrientDB database:  'target' (Path: /Users/orientdb-enterprise-2.2.26/databases/target)
Neo4j to OrientDB Importer - v.2.2.26 - PHASE 1 completed!

Getting all Nodes from Neo4j and creating corresponding Vertices in OrientDB...

Added OrientDB Vertices: 718 (0% done)
Added OrientDB Vertices: 3647 (0% done)
Added OrientDB Vertices: 9570 (1% done)
Added OrientDB Vertices: 18380 (1% done)
Added OrientDB Vertices: 29000 (2% done)
Added OrientDB Vertices: 41019 (3% done)
Added OrientDB Vertices: 53414 (4% done)
Added OrientDB Vertices: 64857 (4% done)
Added OrientDB Vertices: 78000 (5% done)
```

At the end of the migration, statistics about the process are reported as shown below:



Status: FINISHED
From: bolt://localhost:7687
To: plocal:$ORIENTDB_HOME/databases/target **Protocol**: plocal

```
- NOT UNIQUE Indices created due to failure in creating UNIQUE Indices                    : 0 (0%)

- Found Neo4j (non-constraint) Indices                                                    : 26
- Imported OrientDB Indices                                                             : 8 (31%)

- Additional internal Indices created                                                    : 66

- Total Import time:                                                                   : 00:39:42
-- Initialization time                                                                 : 00:00:00
-- Time to Import Nodes                                                                   : 00:01:50 (13485.44 nodes/sec)
-- Time to Import Relationships                                                            : 00:18:16 (7134.46 rels/sec)
-- Time to Import Constraints and Indices                                                    : 00:09:47 (0.04 indices/sec)
-- Time to Create Internal Indices (on vertex properties 'neo4jNodeID' & 'neo4jLabelList')        : 00:03:59 (0.02 indices/sec)
-- Time to Create Internal Indices (on edge property 'neo4jRelID')                               : 00:05:47 (0.18 indices/sec)

Neo4j to OrientDB Importer - v.2.2.26 - PHASE 4 completed!
```

Migration Details

Internally, the *Neo4j to OrientDB Importer* makes use of:

- the Neo4j's bolt connector based on the the Bolt binary protocol to read the graph database from Neo4j
- the OrientDB's java API to store the graph into OrientDB

The migration consists of four phases:

- **Phase 1**: Connection initialization to Neo4j
- **Phase 2**: Migration of nodes and relationships present in the source graph database
- **Phase 3**: Schema migration
- **Phase 4**: Shutdown of the connection to Neo4j and summary info reporting

General Migration Details

The following are some general migration details that is good to keep in mind:

- In case a node in Neo4j has no *Label*, it will be imported in OrientDB into the Class *"GenericClassNeo4jConversion"*.
- In case a node in Neo4j has multiple *Labels*, it will be imported into the `Class` *"MultipleLabelNeo4jConversion"*.
- List of original Neo4j *Labels* are stored as properties in the imported OrientDB vertices (property: *"Neo4jLabelList"*).
- During the import, a not unique index is created on the property *"Neo4jLabelList"*. This allows you to query by *Label* even over nodes migrated into the single `Class` *"MultipleLabelNeo4jConversion"*, using queries like: `SELECT FROM V WHERE Neo4jLabelList CONTAINS 'your_label_here'` or the equivalent with the MATCH syntax: `MATCH {class: V, as: your_alias, where: (Neo4jLabelList CONTAINS 'your_label'} RETURN your_alias`.
- Original Neo4j `IDs` are stored as properties in the imported OrientDB vertices and edges (`Neo4jNodeID` for vertices and `Neo4jRelID` for edges). Such properties can be (manually) removed at the end of the import, if not needed.
- During the import, an OrientDB index is created on the property `Neo4jNodeID` for all imported vertex `classes` (node's *Labels* in Neo4j). This is to speed up vertices lookup during edge creation. The created indexes can be (manually) removed at the end of the import, if not needed.
- In case a Neo4j Relationship has the same name of a Neo4j *Label*, e.g. *"RelationshipName"*, the *Neo4j to OrientDB Importer* will import that relationship into OrientDB in the class `E_RelationshipName` (i.e. prefixing the Neo4j's `RelationshipType` with an `E_`).
- Neo4j Nodes with same *Label* but different case, e.g. *LABEL* and *LAbel* will be aggregated into a single OrientDB vertex `Class` .

- Neo4j Relationship with same name but different case, e.g. *relaTIONship* and *RELATIONSHIP* will be aggregated into a single OrientDB edge `Class`
- Migration of Neo4j's *"existence"* constraints (only available in the Neo4j's Enterprise Edition) is currently not implemented.
- During the creation of properties in OrientDB, Neo4j `Char` data type is mapped to a `String` data type.

Details about Schema Migration

The following are some schema-specific migration details that is good to keep in mind:

- If in Neo4j there are no constraints nor indexes, and if after we drop, after the migration, the properties and indexes created for internal purposes (`Neo4jNodeID` , `Neo4jRelID` , `Neo4jLabelList` and corresponding indexes), the imported OrientDB database is schemaless.
- If in Neo4j there are constraints or indexes, the imported OrientDB database is schema-hybrid (with some properties defined). In particular, for any constraint and index:
    - The Neo4j property where the constraint or index is defined on, is determined.
    - A corresponding property is created in OrientDB (hence the schema-hybrid mode).
- If a Neo4j unique constraint is found, a corresponding unique index is created in OrientDB
    - In case the creation of the unique index fails, a not unique index will be created. Note: this scenario can happen, by design, when migrating nodes that have multiple *Labels*, as they are imported into a single vertex `Class`).
- If a Neo4j index is found, a corresponding (not unique) OrientDB index is created.

Migration Best Practices

According to the migration logic shown so far, we can define the following best practices, adoptable before the migration towards OrientDB:

1. Check if you are using *Labels* with same name but different case, e.g. *LABEL* and *LAbel* and if you really need them. If the correct *Label* is *Label*, change *LABEL* and *LAbel* to *Label* in the original Neo4j database before the import. If you really cannot change them, be aware that with the current version of the Neo4j to OrientDB Importer such nodes will be aggregated into a single OrientDB vertex `Class` .
2. Check if you are using relationships with same name but different case, e.g. *relaTIONship* and *RELATIONSHIP* and if you really need them. If the correct relationship is *Relationship*, change *relaTIONship* and *RELATIONSHIP* to *Relationship* before the import. If you really cannot change them, be aware that with the current version of the Neo4j to OrientDB Importer such relationships will be aggregated into a single OrientDB edge `Class` .
3. Check your constraints and indexes before starting the import. Sometime you have more constraints or indexes than needed, e.g. old ones that you created on *Labels* that you are not using anymore. These constraints will be migrated as well, so a best practice is to check that you have defined, in Neo4j, only those that you really want to import. To check constraints and indexes in Neo4j, you can type `:schema` in the Browser and then click on the "play" icon. Please delete the not needed items.
4. Check if you are using nodes with multiple *Labels*, and if you really need more than one *Label* on them. Be aware that with current version of the Neo4j to OrientDB Importer such nodes with multiple *Labels* will be imported into a single OrientDB `Class` ("*MultipleLabelNeo4jConversion*").

Teleporter

OrientDB Teleporter is a tool that synchronizes a RDBMS to OrientDB database. You can use Teleporter to:

- Import your existing RDBMS to OrientDB
- Keep your OrientDB database synchronized with changes from the RDBMS. In this case the database on RDBMS remains the primary and the database on OrientDB a synchronized copy. Synchronization is one way, so all the changes in OrientDB database will not be propagated to the RDBMS

Teleporter is fully compatible with several RDBMS that have a JDBC driver: we successfully tested Teleporter with Oracle, SQLServer, MySQL, PostgreSQL and HyperSQL. Teleporter manages all the necessary type conversions between the different DBMSs and imports all your data as Graph in OrientDB.

NOTE: This feature is available both for the OrientDB Enterprise Edition and the OrientDB Community Edition. **But beware**: in **community edition** you can migrate your source relational database but **you cannot enjoy the synchronize feature**, only available in the enterprise edition.

How Teleporter works

Teleporter looks for the specific DBMS meta-data in order to perform a logical inference of the source DB schema for the building of a corresponding graph model. Eventually the data importing phase is performed.

Teleporter has a pluggable importing strategy. Two strategies are provided out of the box:

- **naive** strategy, the simplest one
- **naive-aggregate** strategy. It performs a "naive" import of the data source. The data source schema is translated semi-directly in a correspondent and coherent graph model using an aggregation policy on the junction tables of dimension equals to 2

To learn more about the two different execution strategies click here.

Usage

Teleporter is a tool written in Java, but can be used as a tool thanks to the teleporter.sh script (or .bat on Windows).

```
./oteleporter.sh -jdriver <jdbc-driver> -jurl <jdbc-url> -juser <username>
                -jpasswd <password> -ourl <orientdb-url> [-s <strategy>]
                [-nr <name-resolver>] [-v <verbose-level>]
                ([-include <table-names>] | [-exclude <table-names>])
                [-inheritance <orm-technology>:<ORM-file-url>]
                [-conf <configuration-file-location>]
```

Arguments

- **-jdriver** is the driver name of the DBMS from which you want to execute the import (it's not case sensitive)
- **-jurl** is the JDBC URL giving the location of the source database to import
- **-ourl** is the URL for the destination OrientDB graph database
- **-juser (optional)** is the username to access the source database
- **-jpasswd (optional)** is the password to access the source database
- **-s (optional)** is the strategy adopted during the importing phase. If not specified naive-aggregate strategy is adopted. Possible values:
  - **naive**: performs a "naive" import of the data source. The data source schema is translated semi-directly in a correspondent and coherent graph model
  - **naive-aggregate**: performs a "naive" import of the data source. The data source schema is translated semi-directly in a correspondent and coherent graph model using an aggregation policy on the junction tables of dimension equals to 2
- **-nr (optional)** is the name of the resolver which transforms the names of all the elements of the source database according to a specific convention (if not specified original convention is adopted). Possible values:
  - **original**: maintains the original name convention

    - **java**: performs name transformations on all the elements of the data source according to the Java convention
- **-v (optional)** is the level of verbosity printed to the output during the execution (if not specified INFO level will be adopted). Levels:
    - **0**: NO logging messages will be printed
    - **1**: DEBUG level logging messages
    - **2**: INFO level logging messages (default)
    - **3**: WARNING level logging messages
    - **4**: ERROR level logging messages
- **-include (optional)** allows you to import only the listed tables
- **-exclude (optional)** excludes the listed tables from the importing process
- **-inheritance (optional)** executes the import taking advantage of OrientDB's polymorphism
- **-config** allows you to define a custom configuration for your importing job

Access Credentials

By convention three users are always created by default each time a new database is built. Passwords are the same as the user name. Default users are:

- `admin` , with default password " `admin` ", has access to all functions without limitation.
- `reader` , with default password " `reader` ", is the classic read-only user. The reader can read any records but can't modify or delete them and has no access to internal information such as users and roles, themselves.
- `writer` , with the default password " `writer` ", is like the user reader but can also create, update, and delete records.

For further informations about the Security of the OrientDB database click here.

Example of "testdb" importing from PostgreSQL DBMS with default parameters

```
./oteleporter.sh -jdriver postgresql -jurl jdbc:postgresql://localhost:5432/testdb
                -juser username -jpasswd password -ourl plocal:$ORIENTDB_HOME/databases/testdb
```

With these parameters it will be performed an import according to the default settings:

- strategy adopted: **naive-aggregate**
- name resolver: **original name resolver**
- level of verbosity: **INFO** (2nd level)

Example of "testdb" importing from PostgreSQL DBMS with customized optional parameters

```
./oteleporter.sh -jdriver postgresql -jurl jdbc:postgresql://localhost:5432/testdb
                -juser username -jpasswd password -ourl plocal:$ORIENTDB_HOME/databases/testdb
                -s naive -nr java -v 1
```

With these parameters it will be performed an import according to the chosen settings:

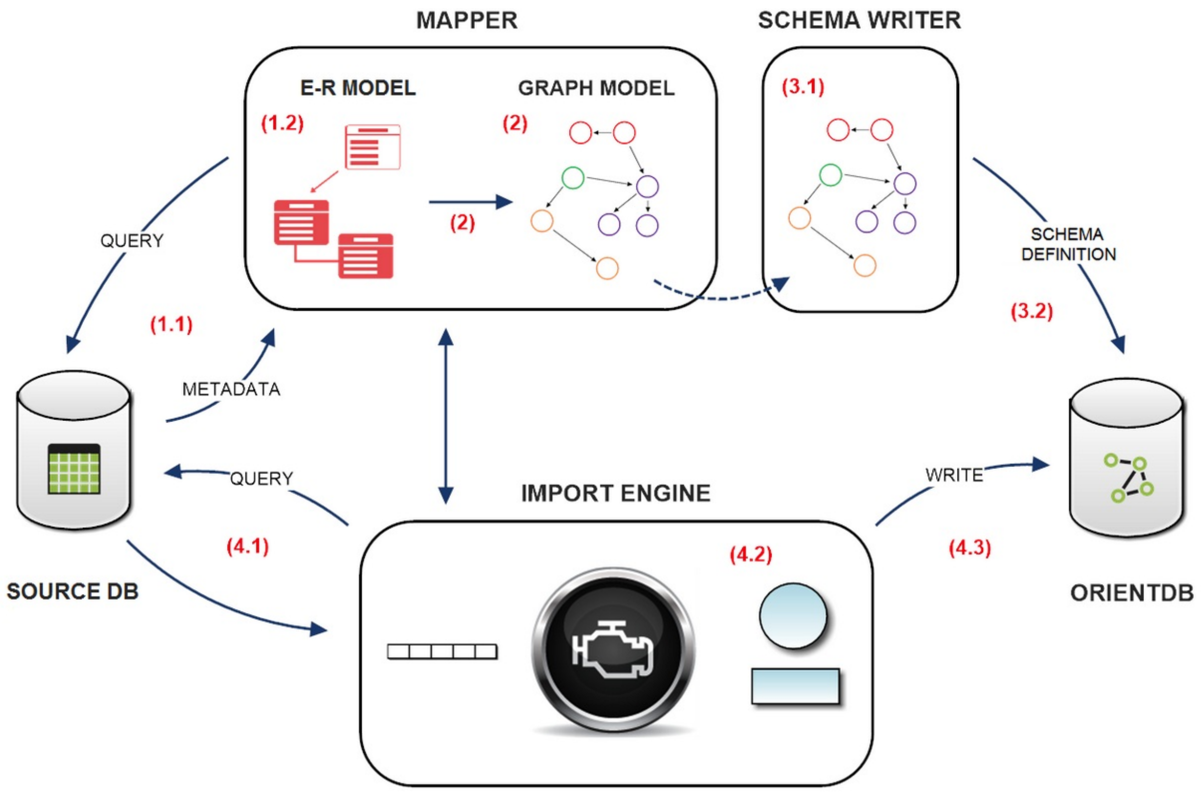
- strategy adopted: **naive**
- name resolver: **java name resolver**
- level of verbosity: **DEBUG** (1st level)

Teleporter Execution

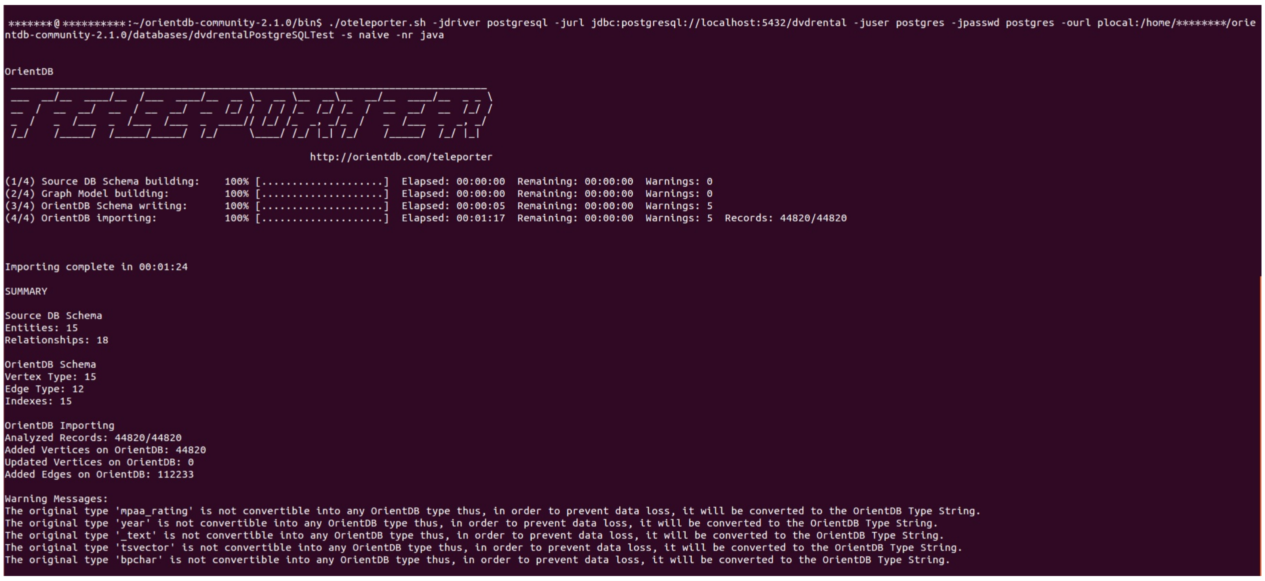
Teleporter execution consists of 4 steps:

1. **Source DB Schema Building:** the source database schema is built by querying the source DB metadata.
2. **Graph Model Building:** a correspondent and coherent Graph Model is built.
3. **OrientDB Schema Writing:** the OrientDB schema is written according to the Graph Model in memory.
4. **OrientDB importing:** importing data from source database to OrientDB.

Thus the whole workflow is:



Below is reported a Teleporter execution dump:



```
*******@**********:~/orientdb-community-2.1.0/bin$ ./oteleporter.sh -jdriver postgresql -jurl jdbc:postgresql://localhost:5432/dvdrental -juser postgres -jpasswd postgres -ourl plocal:/home/********/orie
ntdb-community-2.1.0/databases/dvdrentalPostgreSQLTest -s naive -nr java


OrientDB
 ______________________________________________________________________________
 ___  __/__  ____/__  /___  ____/__  __ \_  __ \__  __ \__  __/__  ____/__  _ _ \
 __  /  __  __/  __  / __  __/  __  /_/ / / / /_  /_/ /_  /  __  __/  __  /_/ /
 _  /   _  /___  _  /___  /___  _  ____// /_/ /_  _, _/_  /   _  /___  _  _, _/
 /_/    /_____/  /_____/_____/  /_/     \____/ /_/ |_| /_/    /_____/  /_/ |_|

                                                 http://orientdb.com/teleporter

(1/4) Source DB Schema building:    100% [....................]  Elapsed: 00:00:00  Remaining: 00:00:00  Warnings: 0
(2/4) Graph Model building:         100% [....................]  Elapsed: 00:00:00  Remaining: 00:00:00  Warnings: 0
(3/4) OrientDB Schema writing:      100% [....................]  Elapsed: 00:00:05  Remaining: 00:00:00  Warnings: 5
(4/4) OrientDB importing:           100% [....................]  Elapsed: 00:01:17  Remaining: 00:00:00  Warnings: 5  Records: 44820/44820



Importing complete in 00:01:24

SUMMARY

Source DB Schema
Entities: 15
Relationships: 18

OrientDB Schema
Vertex Type: 15
Edge Type: 12
Indexes: 15

OrientDB Importing
Analyzed Records: 44820/44820
Added Vertices on OrientDB: 44820
Updated Vertices on OrientDB: 0
Added Edges on OrientDB: 112233

Warning Messages:
The original type 'mpaa_rating' is not convertible into any OrientDB type thus, in order to prevent data loss, it will be converted to the OrientDB Type String.
The original type 'year' is not convertible into any OrientDB type thus, in order to prevent data loss, it will be converted to the OrientDB Type String.
The original type '_text' is not convertible into any OrientDB type thus, in order to prevent data loss, it will be converted to the OrientDB Type String.
The original type 'tsvector' is not convertible into any OrientDB type thus, in order to prevent data loss, it will be converted to the OrientDB Type String.
The original type 'bpchar' is not convertible into any OrientDB type thus, in order to prevent data loss, it will be converted to the OrientDB Type String.
```

Installation and Configuration

Installation

Teleporter is out-of-the-box both in Community and Enterprise Edition, so you don't need any configuration or modification. **But beware**: in **Community Edition** you can migrate your source relational database but **you cannot enjoy the synchronize feature**, only available in the **Enterprise Edition**.

You can run the tool through the script as described in the Home page or just execute it via OrientDB Studio as described here.

Driver Configuration.

Automatic Driver Configuration

Teleporter provides an automaic driver configuration: when the application starts, it looks for the required driver. If the driver is not found the application will download it and it will automatically configure the classpath, not delegating anything to the end user.
So when you run Teleporter you just have to indicate the name of the DBMS you want to connect. Teleporter is compatible with Oracle, MySQL, PostgreSQL and HyperSQL products, thus you have to type one of the following parameters **(not case sensitive)**:

- **Oracle**
- **SQLServer**
- **MySQL**
- **PostgreSQL**
- **HyperSQL**

Teleporter will search for the correspondent driver in the $ORIENTDB_HOME/lib folder and if it's not present, it will download the last available driver version. If a driver is already present in the folder, then it will be used for the connection to the source DB. Therefore if you want use a new driver version, you just have to delete the older version and run Teleporter which will download and configure for you the last available version.

```
./oteleporter.sh -jdriver postgresql -jurl jdbc:postgresql://localhost:5432/testdb
                -juser username -jpasswd password -ourl plocal:$ORIENTDB_HOME/databases/testdb
                -s naive -nr java -v 2
```

Manual Driver configuration

It's possible to perform a manual configuration downloading own favourite driver version and properly defining the classpath in the application. Below are reported last driver tested versions with some useful information for download, configuration and use.

Driver	Last Tested Version	Path pattern	Path Example
Oracle	12c	jdbc:oracle:thin:@HOST:PORT:SID	jdbc:oracle:thin:@localhost:1521:orcl
SQLServer	SQLServer 2014	jdbc:sqlserver://HOST:PORT;databaseName=DB	jdbc:sqlserver://localhost:1433;databaseName **(*)**
MySQL	5.1.35	jdbc:mysql://HOST:PORT/DB	jdbc:mysql://localhost:3306/testdb
PostgreSQL	9.4-1201	jdbc:postgresql://HOST:PORT/DB	jdbc:postgresql://localhost:5432/testdb
HyperSQL	2.3.2	jdbc:hsqldb:hsql://HOST:PORT/DB OR jdbc:hsqldb:file:FILEPATH	jdbc:hsqldb:hsql://localhost:9500/testdb OR jdbc:hsqldb:file:testdb

(*) If the source database contains spaces in the name you have to use a URL like this:

“Source DB” → -jurl “jdbc:sqlserver://localhost:1433;databaseName={Source DB};”

Execution Strategies

Teleporter provides two different import strategies:

- **naive** strategy
- **naive-aggregate** strategy

Both strategies build a schema in OrientDB starting from the source DB schema: each table (known also as Entity) and each Relationship in the DB is inferred from these metadata, therefore if you didn't defined some constraints, such as foreign keys between the tables on which you usually perform join operations, you will lose this kind of info during the import process.
For example if foreign keys are missing, you will not have any edges in your final Graph Database.
You can overcome this limit by defining an Import Configuration that allows you to add new relationships or modify those already present in your source database schema.
Once built the OrientDB schema, the real import process begins.

Now both strategies will be individually discussed below.

Naive Strategy

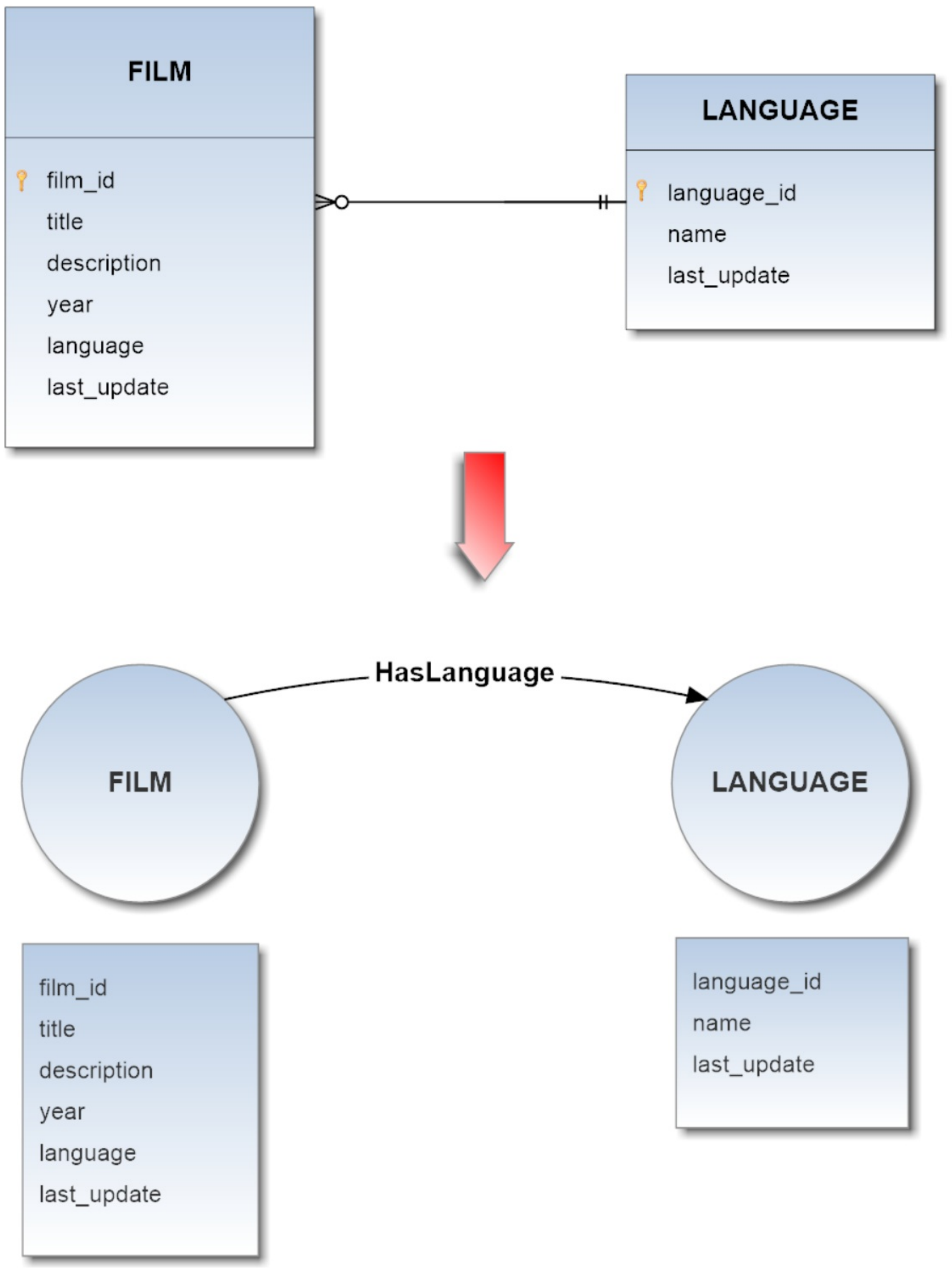
This strategy follows a basic approach for the import. The source DB schema is directly translated in the OrientDB schema as follows:

1. Each Entity in the source DB is converted into a Vertex Type.
2. Each Relationship between two Entities in the source DB is converted into an Edge Type (remember, relationships in your DB schema are represented by the foreign keys).

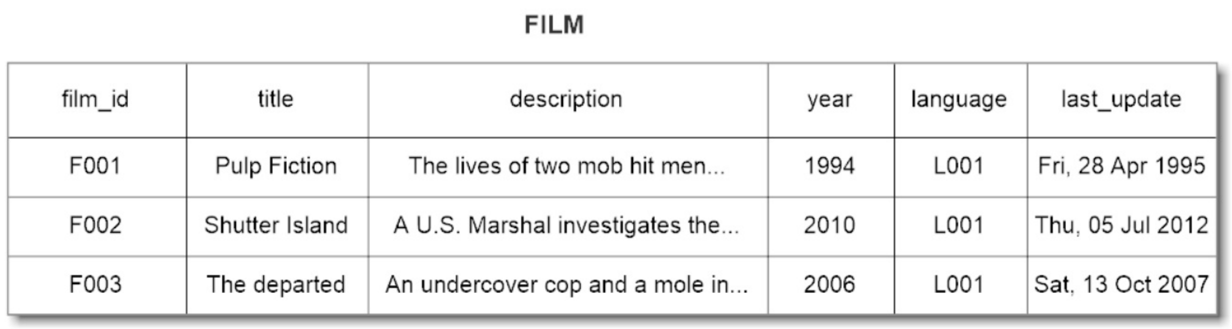
Thus all records of each table are imported according to this "schemas-mapping": each pair of records on which it's possible to perform a join operation, will correspond to a pair of vertices connected by an edge of a specific Edge Type.

Example 1 - Without Join Table

Source DB schema translation in OrientDB schema:

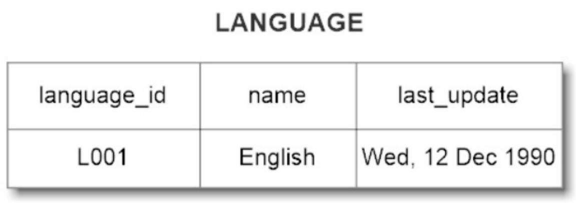


Correspondent records import:



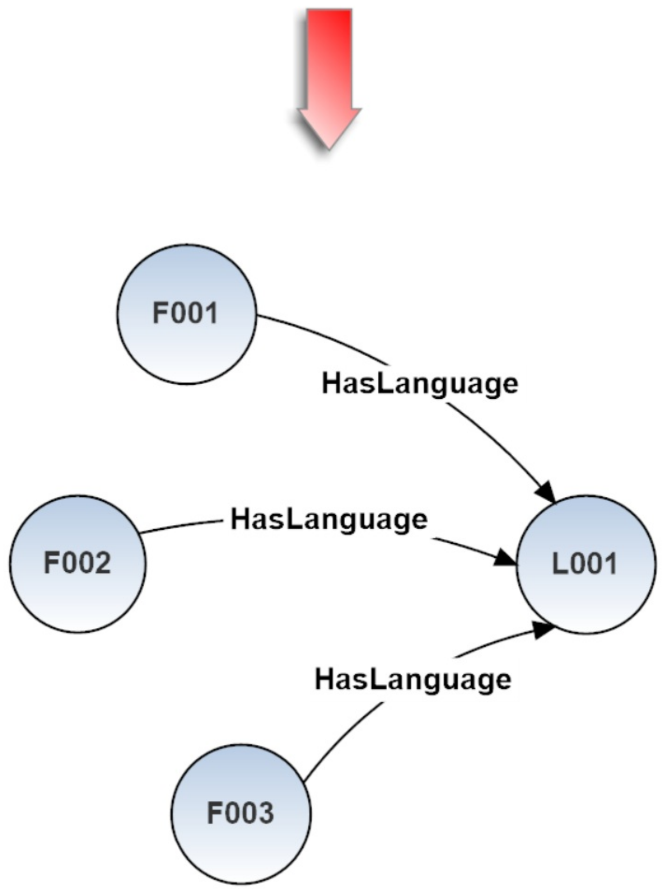
FILM

film_id	title	description	year	language	last_update
F001	Pulp Fiction	The lives of two mob hit men...	1994	L001	Fri, 28 Apr 1995
F002	Shutter Island	A U.S. Marshal investigates the...	2010	L001	Thu, 05 Jul 2012
F003	The departed	An undercover cop and a mole in...	2006	L001	Sat, 13 Oct 2007



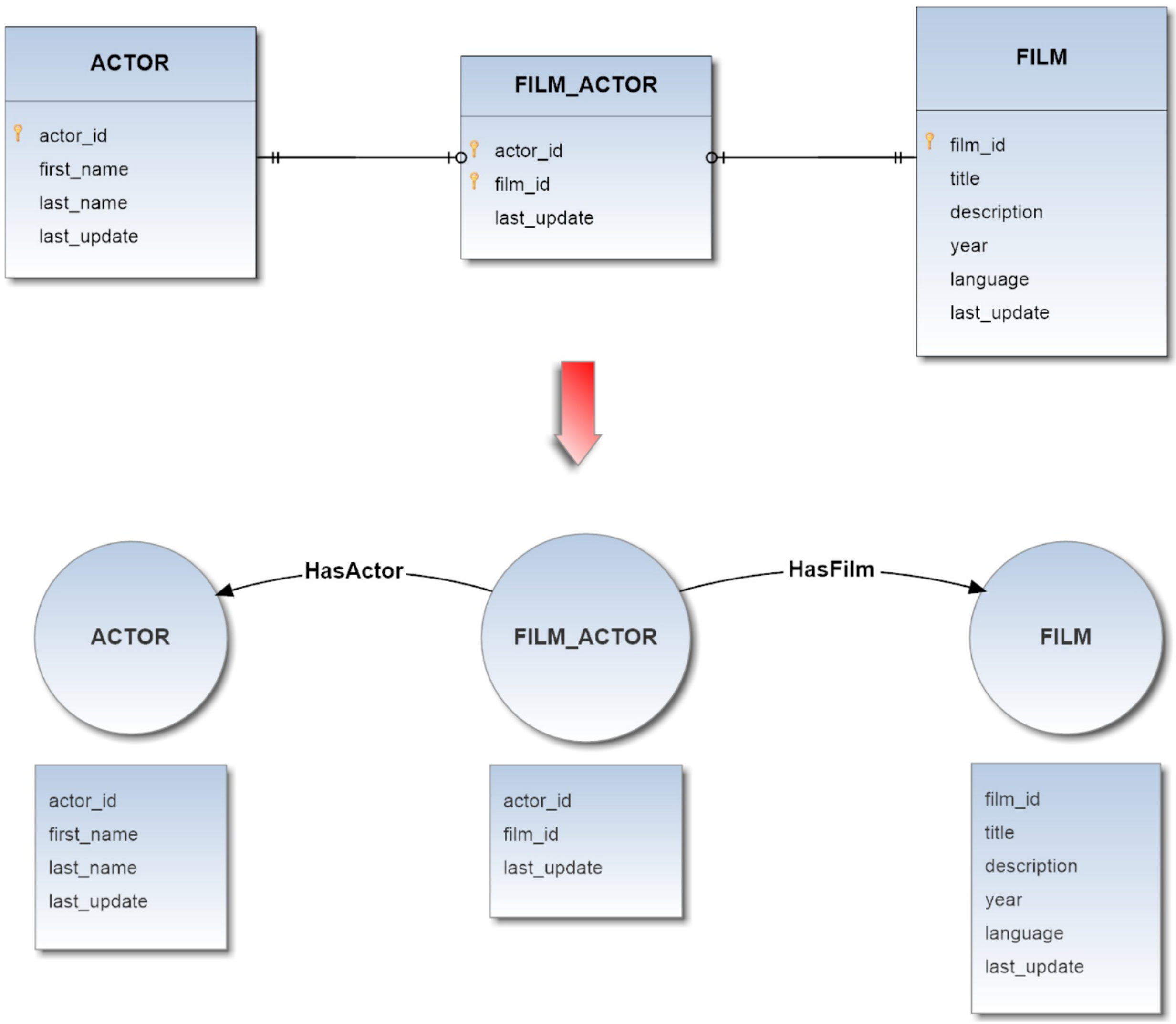
LANGUAGE

language_id	name	last_update
L001	English	Wed, 12 Dec 1990

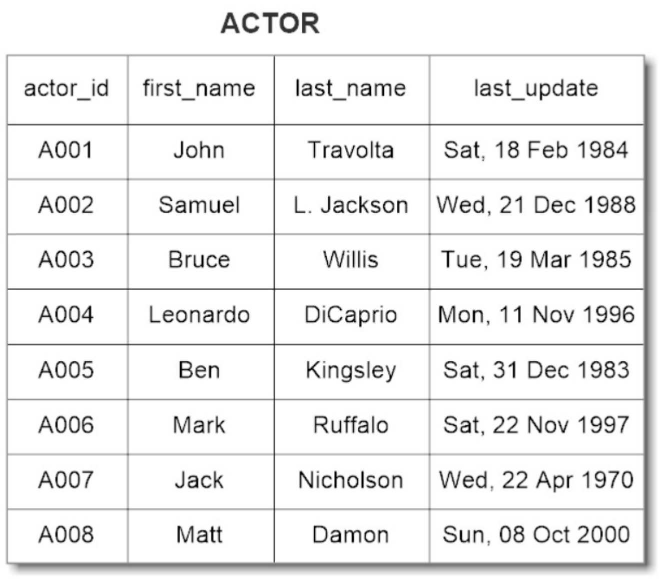


Example 2 - With Aggregable Join Table

Source DB schema translation in OrientDB schema:

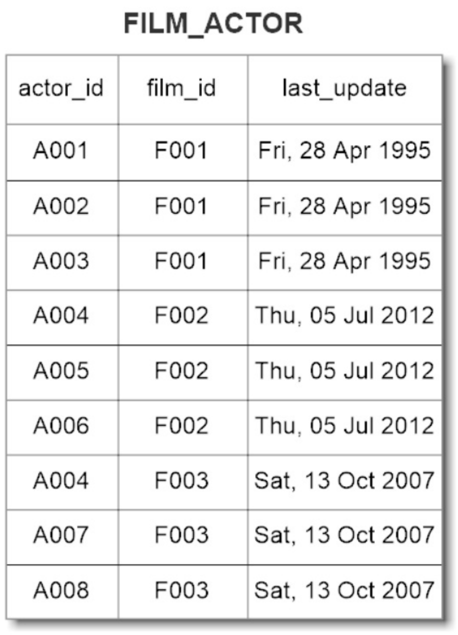


Starting from the following tables



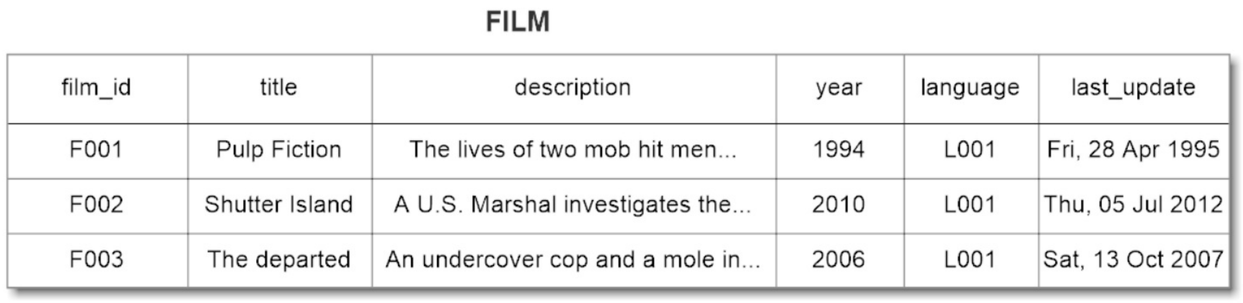
ACTOR

actor_id	first_name	last_name	last_update
A001	John	Travolta	Sat, 18 Feb 1984
A002	Samuel	L. Jackson	Wed, 21 Dec 1988
A003	Bruce	Willis	Tue, 19 Mar 1985
A004	Leonardo	DiCaprio	Mon, 11 Nov 1996
A005	Ben	Kingsley	Sat, 31 Dec 1983
A006	Mark	Ruffalo	Sat, 22 Nov 1997
A007	Jack	Nicholson	Wed, 22 Apr 1970
A008	Matt	Damon	Sun, 08 Oct 2000



FILM_ACTOR

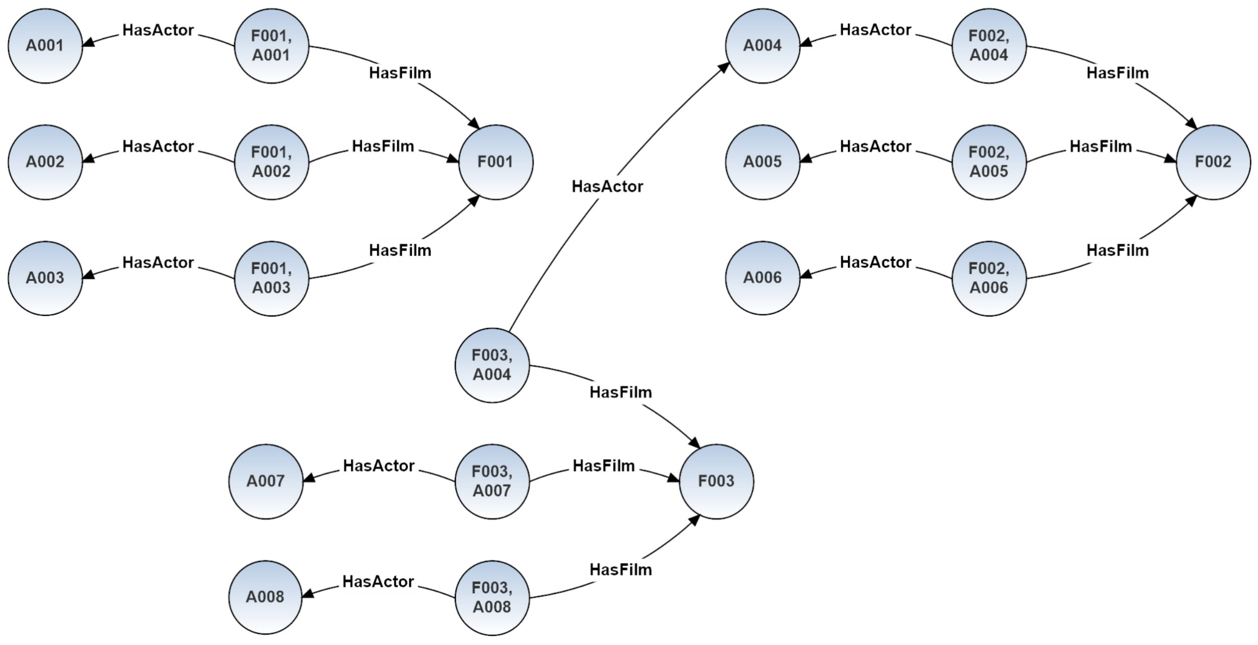
actor_id	film_id	last_update
A001	F001	Fri, 28 Apr 1995
A002	F001	Fri, 28 Apr 1995
A003	F001	Fri, 28 Apr 1995
A004	F002	Thu, 05 Jul 2012
A005	F002	Thu, 05 Jul 2012
A006	F002	Thu, 05 Jul 2012
A004	F003	Sat, 13 Oct 2007
A007	F003	Sat, 13 Oct 2007
A008	F003	Sat, 13 Oct 2007



FILM

film_id	title	description	year	language	last_update
F001	Pulp Fiction	The lives of two mob hit men...	1994	L001	Fri, 28 Apr 1995
F002	Shutter Island	A U.S. Marshal investigates the...	2010	L001	Thu, 05 Jul 2012
F003	The departed	An undercover cop and a mole in...	2006	L001	Sat, 13 Oct 2007

we will obtain the following graph:



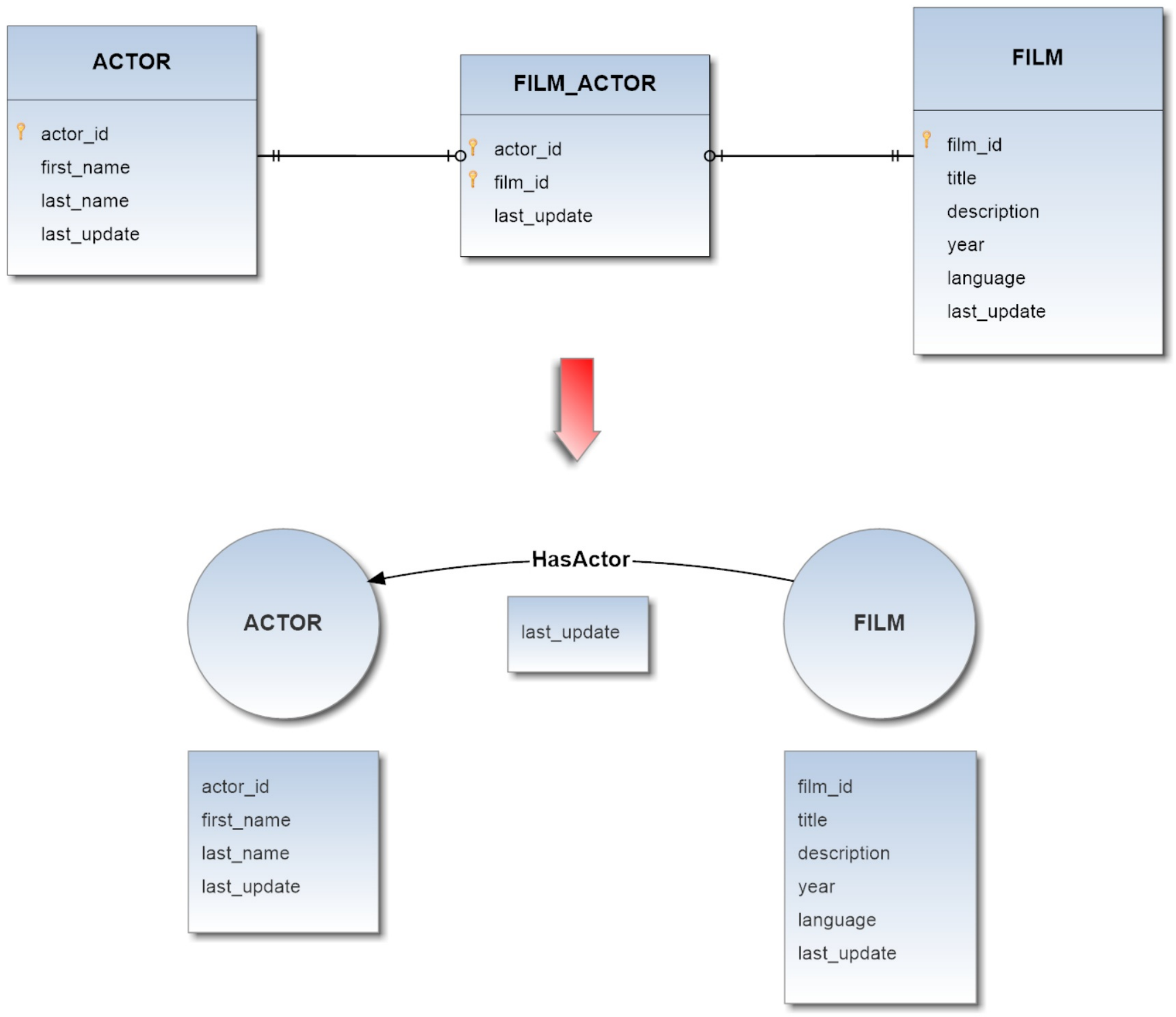
Naive-Aggregate Strategy

Unlike the first strategy, this one performs aggregation on join tables of dimension equals to 2, that is to say those tables which map two tables together by referencing the primary keys of each data table through a foreing key. The join tables of dimension greater than 2 are ignored by the aggregation algorithm. Thus each candidate join table is converted into an appropriate edge, and each field not involved in any relationship with other tables (hence not involved in any foreign key in the source DB schema) is aggregated in the properties of the new built edge.

Referring to the scenario of the last example is evident as even if the new DB doesn't reflect the original DB model, the aggregation leads to a great saving in terms of resources and avoids a substantial overhead. The OrientDB schema after the aggregation process comes out simpler, hence also the import result it is.

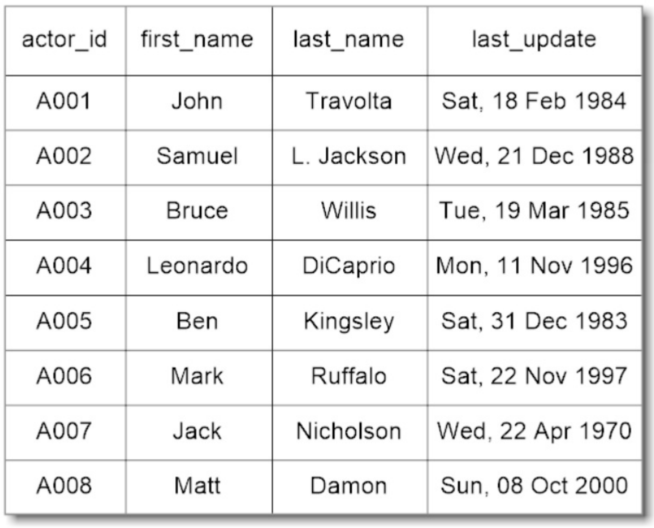
Example 3 - With Aggregable Join Table

Source DB schema translation in OrientDB schema:



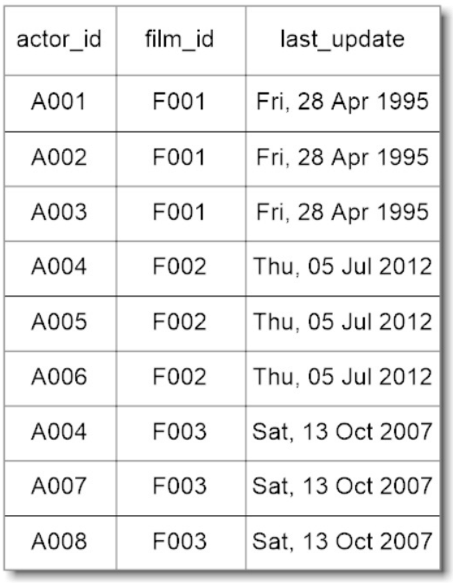
Through this strategy, starting from the same previous scenario

ACTOR



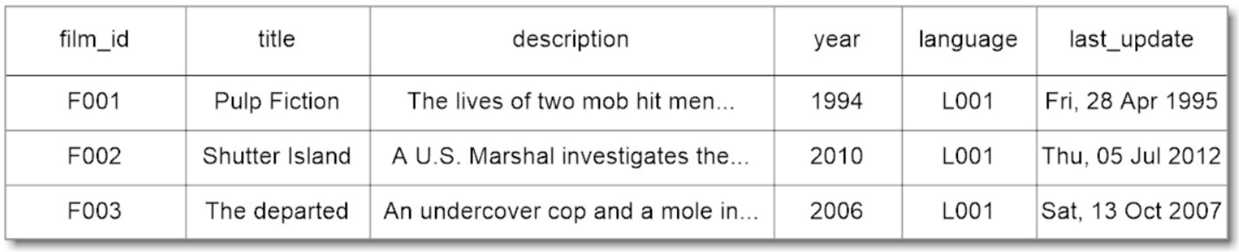
actor_id	first_name	last_name	last_update
A001	John	Travolta	Sat, 18 Feb 1984
A002	Samuel	L. Jackson	Wed, 21 Dec 1988
A003	Bruce	Willis	Tue, 19 Mar 1985
A004	Leonardo	DiCaprio	Mon, 11 Nov 1996
A005	Ben	Kingsley	Sat, 31 Dec 1983
A006	Mark	Ruffalo	Sat, 22 Nov 1997
A007	Jack	Nicholson	Wed, 22 Apr 1970
A008	Matt	Damon	Sun, 08 Oct 2000

FILM_ACTOR



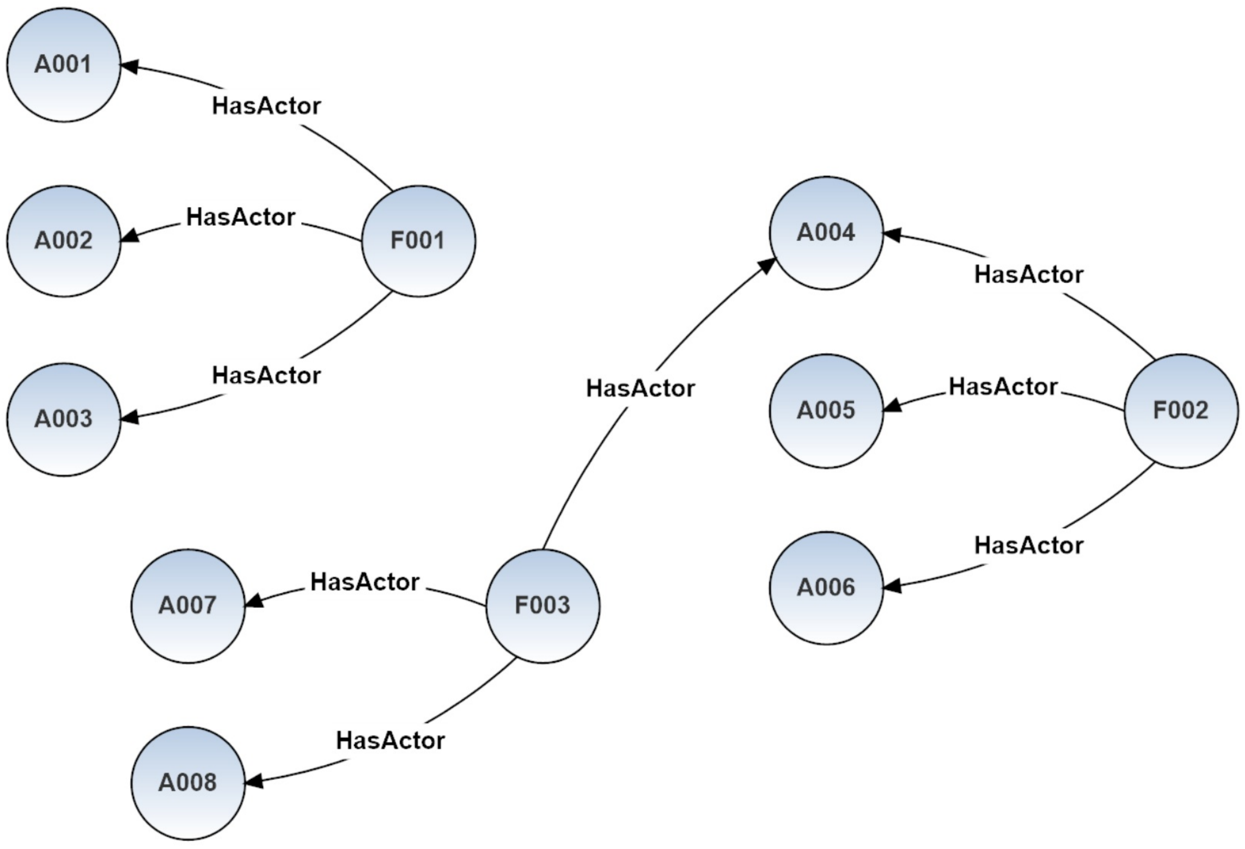
actor_id	film_id	last_update
A001	F001	Fri, 28 Apr 1995
A002	F001	Fri, 28 Apr 1995
A003	F001	Fri, 28 Apr 1995
A004	F002	Thu, 05 Jul 2012
A005	F002	Thu, 05 Jul 2012
A006	F002	Thu, 05 Jul 2012
A004	F003	Sat, 13 Oct 2007
A007	F003	Sat, 13 Oct 2007
A008	F003	Sat, 13 Oct 2007

FILM



film_id	title	description	year	language	last_update
F001	Pulp Fiction	The lives of two mob hit men...	1994	L001	Fri, 28 Apr 1995
F002	Shutter Island	A U.S. Marshal investigates the...	2010	L001	Thu, 05 Jul 2012
F003	The departed	An undercover cop and a mole in...	2006	L001	Sat, 13 Oct 2007

this time we will obtain a less complex graph:

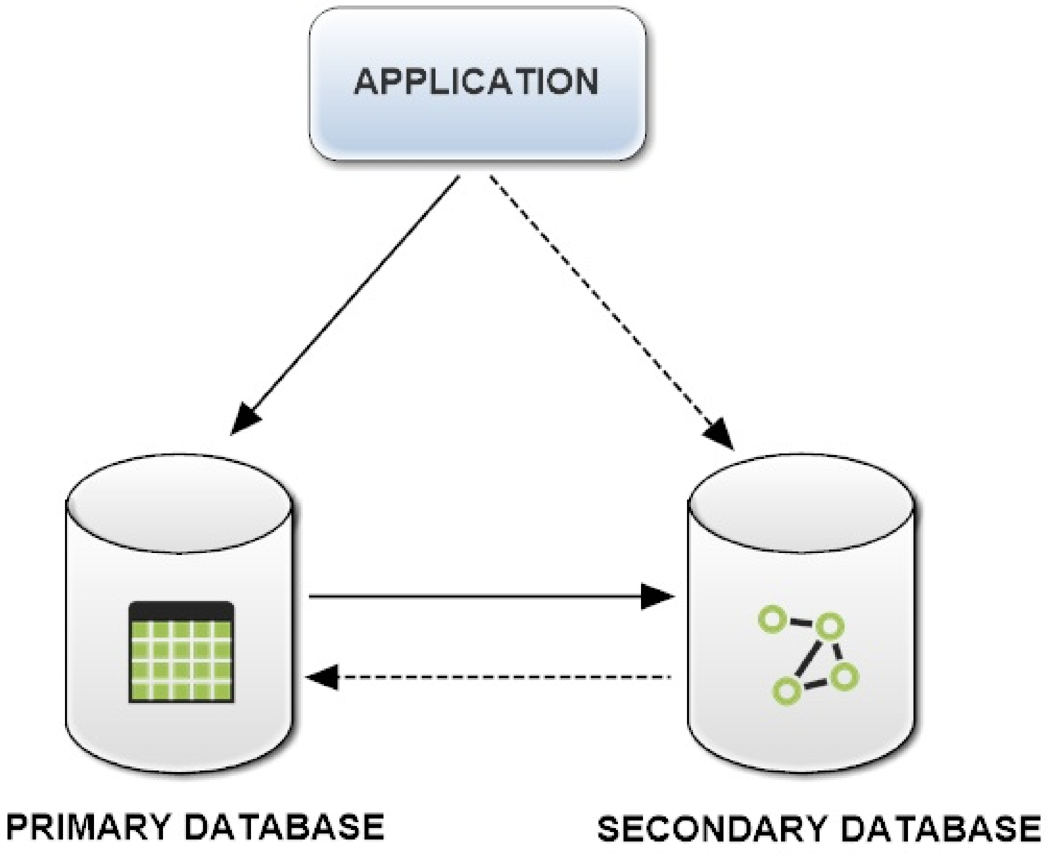


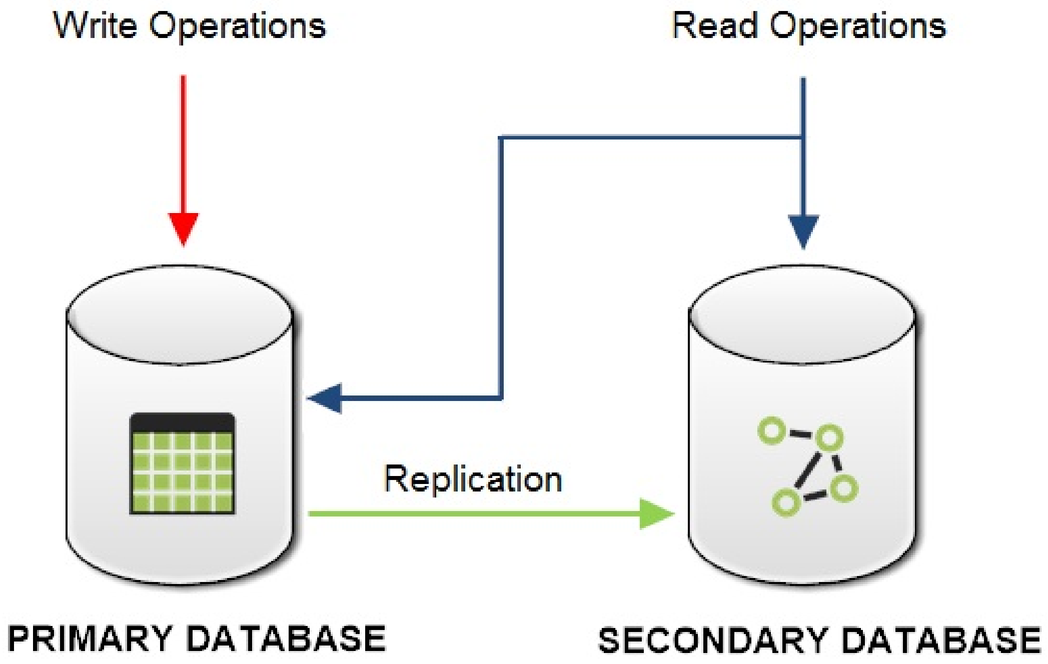
Sequential Executions and One-Way Synchronizer

Teleporter is conceived to support many sequential executions from the same source DB to the same graph DB of OrientDB, in this way you can:

- **personalize your import,** combining the different strategies and settings by including or excluding the chosen tables and by running Teleport more times in order to obtain a more complex and customized import strategy
- **use it as a one-way synchronizer** and maintain a copy of your DB: all the changes applied to the source DB (primary DB) are propagated to the imported graph DB, but not vice versa.

A sample migration scenario is reported below:





Synchronization policy

Teleporter propagates the applied changes of the source DB both for the schema and for the records following the policy described below:

1. SCHEMA SYNCHRONIZATION (Full Synchronization)

SOURCE DB SCHEMA		TARGET ORIENTDB SCHEMA	SYNCH
Add Operation	-->	Add Operation	YES
Delete Operation	-->	Delete Operation	YES
Update Operation	-->	Update Operation	YES

2. RECORDS SYNCHRONIZATION (Delete-less Synchronization)

SOURCE DB		TARGET ORIENTDB GRAPHDB	SYNCH
Add Operation	-->	Add Operation	YES
Delete Operation	-->	No Operation	NO
Update Operation	-->	Update Operation	YES

Import Filters

It's possible to apply filters to the import process through the **include** and **exclude** arguments.
With the **include** argument you'll import the listed tables according to the following syntax:

```
-include <tableName1>,<tableName2>,...,<tableNameX>
```

With the **exclude** argument you'll import all the tables except for the listed ones according to the following syntax:

```
-exclude <tableName1>,<tableName2>,...,<tableNameX>
```

For both arguments recognizing tables is **case sensitive.**
These arguments are **mutually exclusive,** thus you can use just one of them during the same execution.

Example 1: include usage

Importing only the "actor" and "film" tables from the source DB.

```
./oteleporter.sh -jdriver postgresql -jurl jdbc:postgresql://localhost:5432/dvdrental
                -juser username -jpasswd password -ourl plocal:$ORIENTDB_HOME/databases/dvdrental
                -include actor,film
```

Example 2: exclude usage

Importing all tables from the source DB except for the "actor" table.

```
./oteleporter.sh -jdriver postgresql -jurl jdbc:postgresql://localhost:5432/dvdrental
                -juser username -jpasswd password -ourl plocal:$ORIENTDB_HOME/databases/dvdrental
                -exclude actor
```

Inheritance

Teleporter allows you to take advantage of OrientDB's polymorphism. Infact you can enrich the import phase via an ORM file which describes the inheritance relationships being between different tables (or Entities) of your source DB.

At the moment **Hibernate's syntax** is supported, and you can exploit this feature even if you don't use the Hibernate framework, which can automatically build the requested file for you. In fact the ORM file is simply interpreted as a "mapping file" between Relational and Object-Oriented models. Thus you can also write the file by yourself and give it as input to Teleporter, this is all you need.

Inheritance Patterns in Relational Databases

Because relational databases have no concept of inheritance, there isn't a standard way of implementing inheritance in a database, so the hardest part of persisting inheritance is choosing how to represent the inheritance in the database. There are three main patterns commonly used:

- **Single Table Inheritance**
- **Table Per Class Inheritance**
- **Table Per Concrete Class Inheritance**

Teleporter can faithfully reproduce all inheritance relationships present in your source DB using the argument **'-inheritance'** and the following the syntax:

```
./oteleporter.sh -jdriver <jdbc-driver> -jurl <jdbc-url> -juser <username>
                -jpasswd <password> -ourl <orientdb-url> -s <strategy>
                -inheritance hibernate:<ORM-file-url>
```

Example:

```
./oteleporter.sh -jdriver <jdbc-driver> -jurl <jdbc-url> -juser <username>
                -jpasswd <password> -ourl <orientdb-url> -s <strategy>
                -inheritance hibernate:/home/orientdb-user/mapping.xml
```

The resulting hierarchy in OrientDB is the same for each adopted pattern, as shown in the specific pattern descriptions.

Hibernate Syntax

The mapping file is an XML document having \ as the root element which contains all the elements. Now we analyze some details about the Hibernate syntax used for the mapping file definition:

- The \ elements are used to define the correspondence between Java classes and the database tables. The **name** attribute of the class element specifies the Java class name and the **table** attribute specifies the database table name.
- The \ element is an optional element which can contain a class description.
- The \ element maps the unique ID attribute of the Java class to the primary key of the correspondent database table. This element can have a **name** attribute and a **column** attribute which manage the correspondence between the Object Model and the Relational Model as previously described: here the column attribute refers to the column in the table corresponding to that name. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type. If you write this file by yourself you should know that this element with its three attributes are superfluous for Teleporter.
- The \ element within the **id** element is used to automatically generate the primary key values. Also this element is superfluous for Teleport.
- The \ element maps a Java class property to a column in the database table. The **name** attribute and the **column** have the same role in the Object and the Relational models mapping. The **type** attribute holds the hibernate mapping type.

There are other attributes and elements available among which:

- \ element, used in the Single Table Inheritance pattern.

- \ with a nested \ element, used in the Table Per Class Inheritance pattern.
- \ element, used in the Table Per Class Inheritance pattern.
- \ element, used in the Table Per Concrete Class Inheritance pattern.

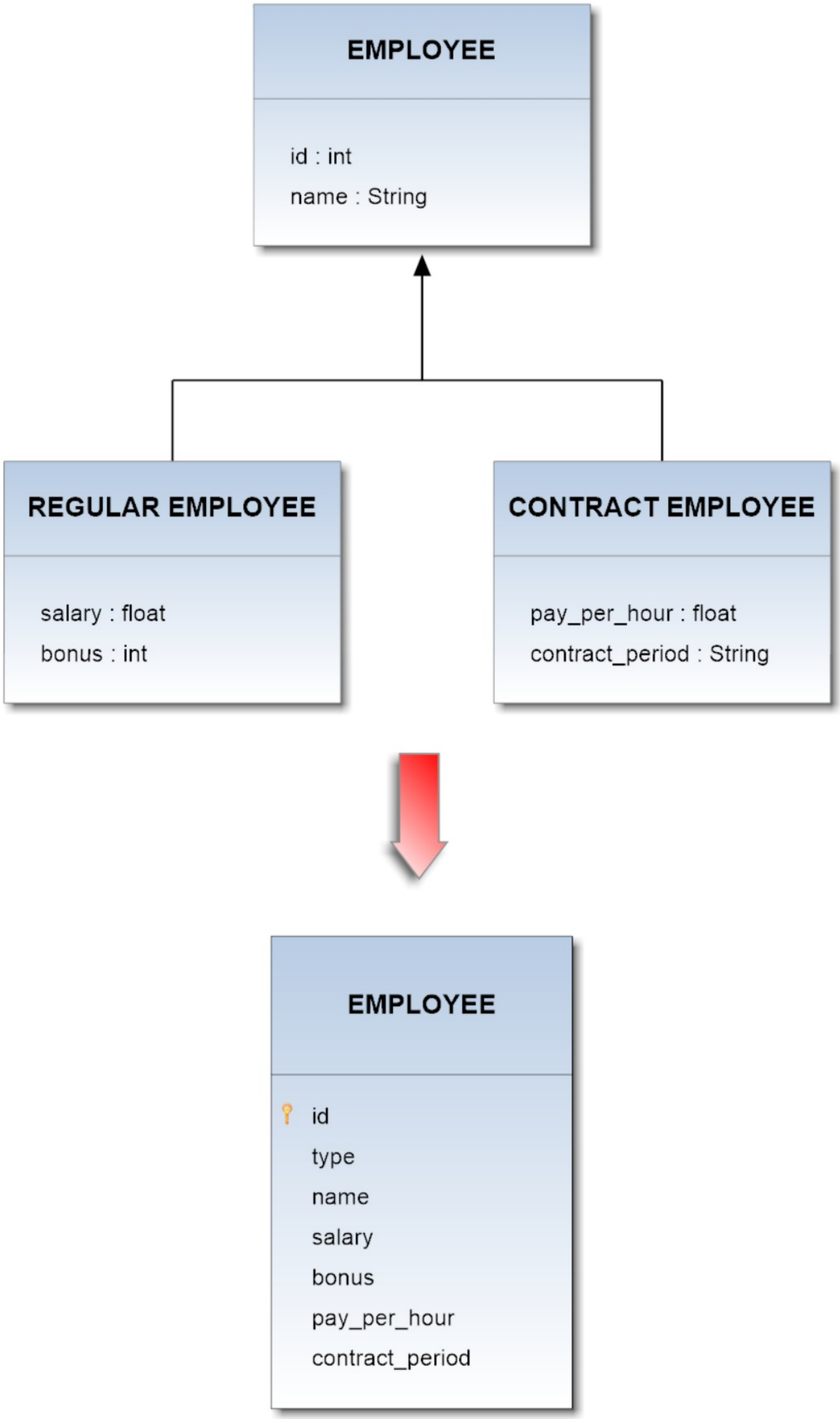
Their usage will be explained specifically in the description of each individual pattern.

Single Table Inheritance

Single Table strategy is the simplest and typically the best performing solution. By this inheritance strategy, we can map the whole hierarchy through a single table. The table will have a column for every attribute of every class in the hierarchy and an extra column (also known as **discriminator** column) is created in the table to identify the class.

Example

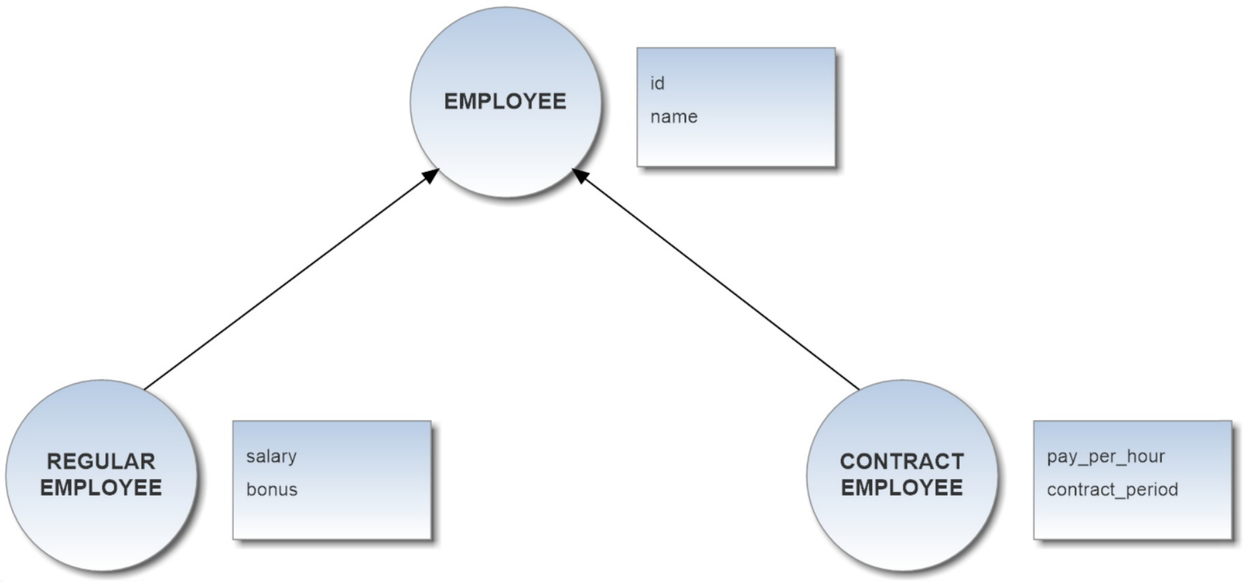
Now suppose you want to map the whole hierarchy given below into a coherent relational database schema. The **Employee** class is a superclass both for **Regular_Employee** and **Contract_Employee** classes. The **type** attribute acts as discriminator column.
The application of the above described pattern leads to the DB schema shown in the following diagram:


EMPLOYEE
id : int
name : String
REGULAR EMPLOYEE
salary : float
bonus : int
CONTRACT EMPLOYEE
pay_per_hour : float
contract_period : String
EMPLOYEE
id
type
name
salary
bonus
pay_per_hour
contract_period

The correspondent mapping file for this hierarchy should be:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="Employee" table="EMPLOYEE"
        discriminator-value="emp">
        <id name="id" column="id">
            <generator class="increment"></generator>
        </id>

        <discriminator column="type" type="string"></discriminator>
        <property name="name" column="name"></property>

        <subclass name="Regular_Employee"
            discriminator-value="reg_emp">
            <property name="salary" column="salary"></property>
            <property name="bonus" column="bonus"></property>
        </subclass>

        <subclass name="Contract_Employee"
            discriminator-value="cont_emp">
            <property name="payPerHour" column="pay_per_hour"></property>
            <property name="contractDuration" column="contract_duration"></property>
        </subclass>
    </class>
</hibernate-mapping>
```

Taking advantage of this inheritance-feature on the proposed model above, you will get the following schema in OrientDB:



If you deal with a multi-level inheritance relationships in the DB, you have to represent them in the ORM file by recursively nesting each definition according to the hierarchical dependences being between the Entities of the model.

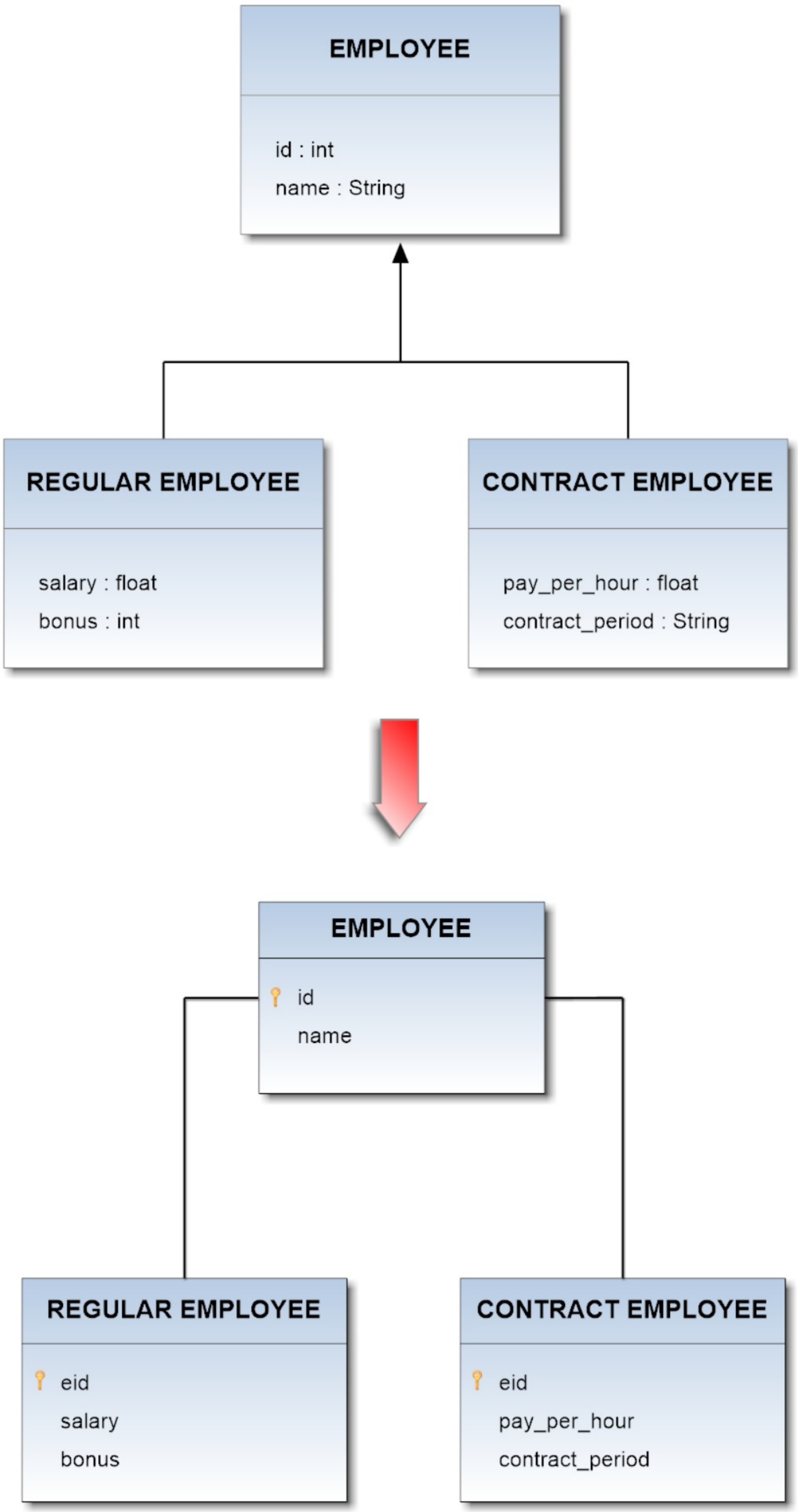
Table Per Class Inheritance

Table Per Class strategy is the most logical inheritance solution because it mirrors the object model in the data model. In this pattern a table is defined for each class in the inheritance hierarchy to store only the local attributes of that class. All classes in the hierarchy must share the same id attribute.

Some JPA providers support Table Per Class Inheritance with or without a discriminator column, some required the discriminator column, and some don't support the discriminator column. This pattern doesn't seem to be fully standardized yet. On Hibernate a discriminator column is supported but not required.

Example

Now suppose you want to map the whole hierarchy given below into a coherent relational database schema. The **Employee** class is a superclass both for **Regular_Employee** and **Contract_Employee** classes.
The application of the above described pattern leads to the DB schema shown in the following diagram:


EMPLOYEE
id : int
name : String
REGULAR EMPLOYEE
salary : float
bonus : int
CONTRACT EMPLOYEE
pay_per_hour : float
contract_period : String
EMPLOYEE
id
name
REGULAR EMPLOYEE
eid
salary
bonus
CONTRACT EMPLOYEE
eid
pay_per_hour
contract_period

There are two equivalent mapping file to represent this hierarchy:

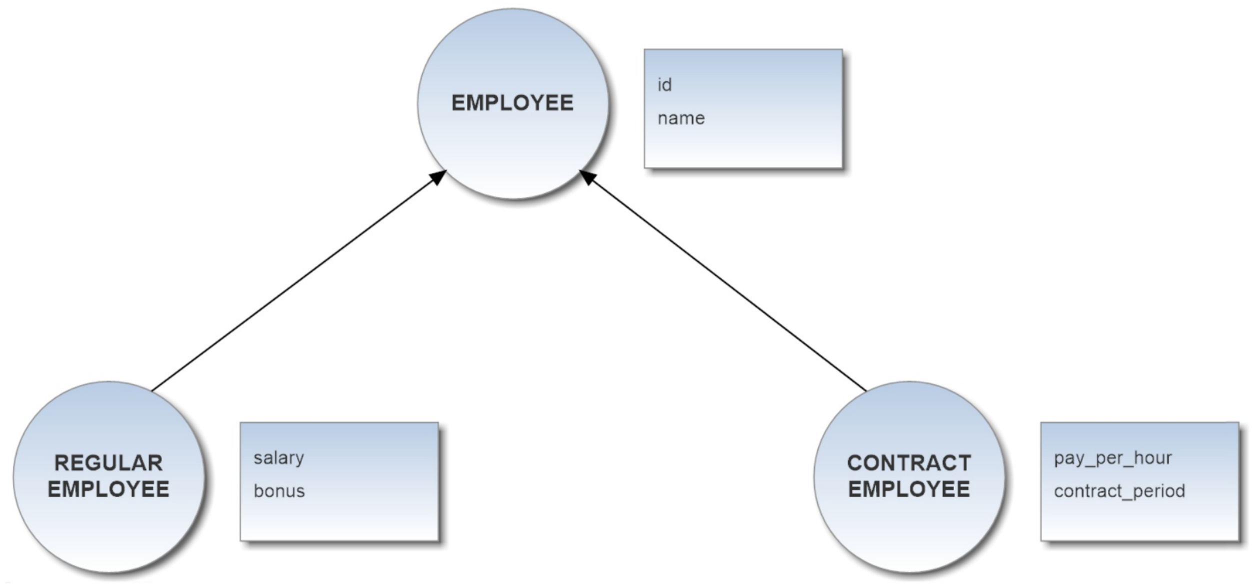
1.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="Employee" table="EMPLOOYEE">
        <id name="id" column="id">
            <generator class="increment"></generator>
        </id>

        <property name="name" column="name"></property>

        <joined-subclass name="Regular_Employee"
            table="REGULAR EMPLOYEE">
            <key column="eid"></key>
            <property name="salary" column="salary"></property>
            <property name="bonus" column="bonus"></property>
        </joined-subclass>

        <joined-subclass name="Contract_Employee"
            table="CONTRACT EMPLOYEE">
            <key column="eid"></key>
            <property name="payPerHour" column="pay_per_hour"></property>
            <property name="contractDuration" column="contract_duration"></property>
        </joined-subclass>
    </class>
</hibernate-mapping>
```

2.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="Employee" table="EMPLOYEE">
        <id name="id" column="id">
            <generator class="increment"></generator>
        </id>
        <discriminator column="type" type="string" />
        <property name="name" column="name"></property>

        <subclass name="Regular_Employee" discriminator-value="R">
            <join table="REGULAR EMPLOYEE">
                <key column="eid"></key>
                <property name="salary" column="salary"></property>
                <property name="bonus" column="bonus"></property>
            </join>
        </subclass>

        <subclass name="Contract_Employee" discriminator-value="C">
            <join table="CONTRACT EMPLOYEE">
                <key column="eid"></key>
                <property name="payPerHour" column="par_per_hour"></property>
                <property name="contractDuration" column="contract_duration"></property>
            </join>
        </subclass>
    </class>
</hibernate-mapping>
```

Taking advantage of this inheritance-feature on the proposed model above, you will get the following schema in OrientDB:



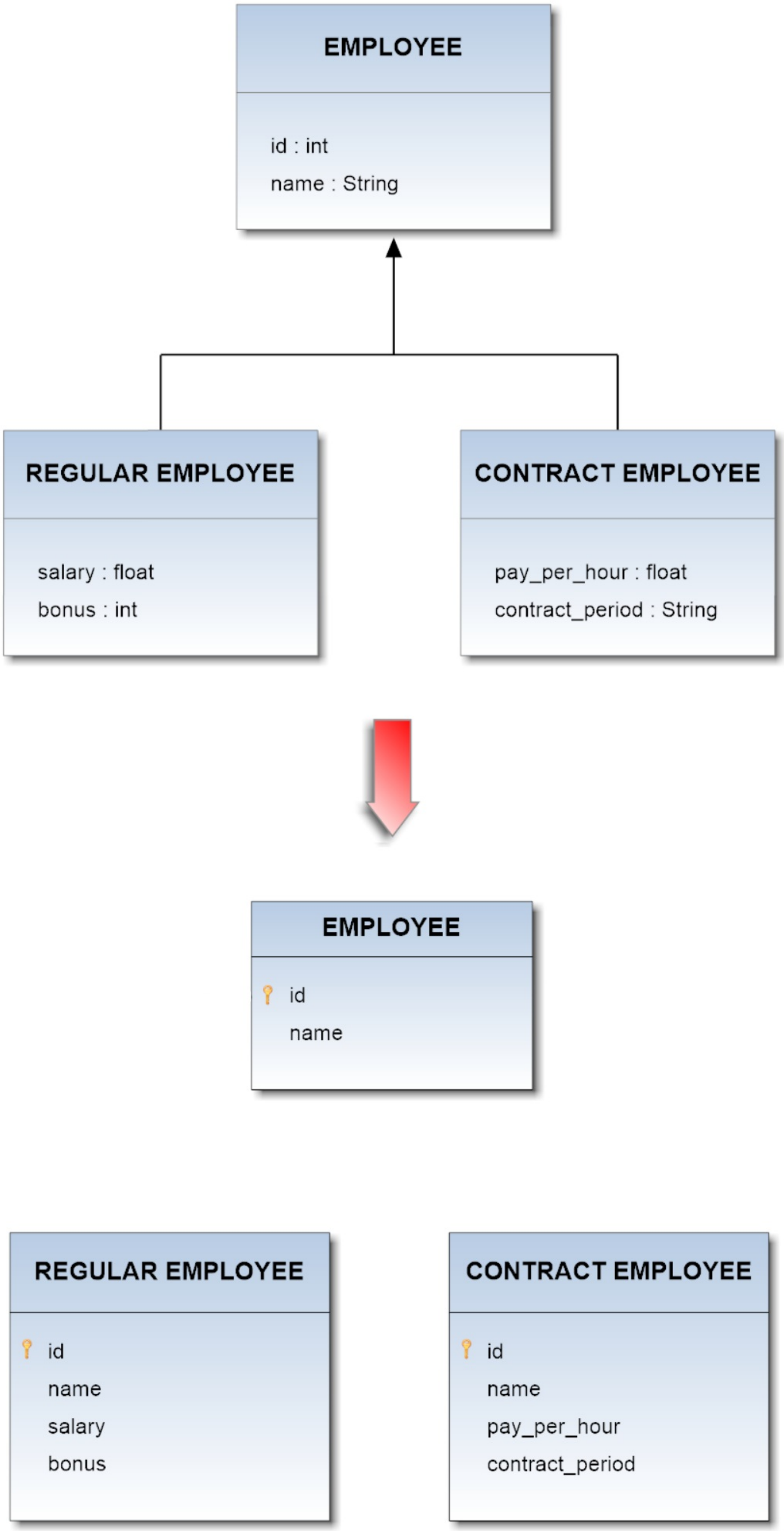
If you deal with a multi-level inheritance relationships in the DB, you have to represent them in the ORM file by recursively nesting each definition according to the hierarchical dependences being between the Entities of the model.

Table Per Concrete Class Inheritance

In Table Per Concrete Class strategy a table is defined for each concrete class in the inheritance hierarchy to store all the attributes of that class and all of its superclasses. This strategy is optional in several ORM technologies (e.g. JPA), and querying root or branch classes can be very difficult and inefficient.

Example

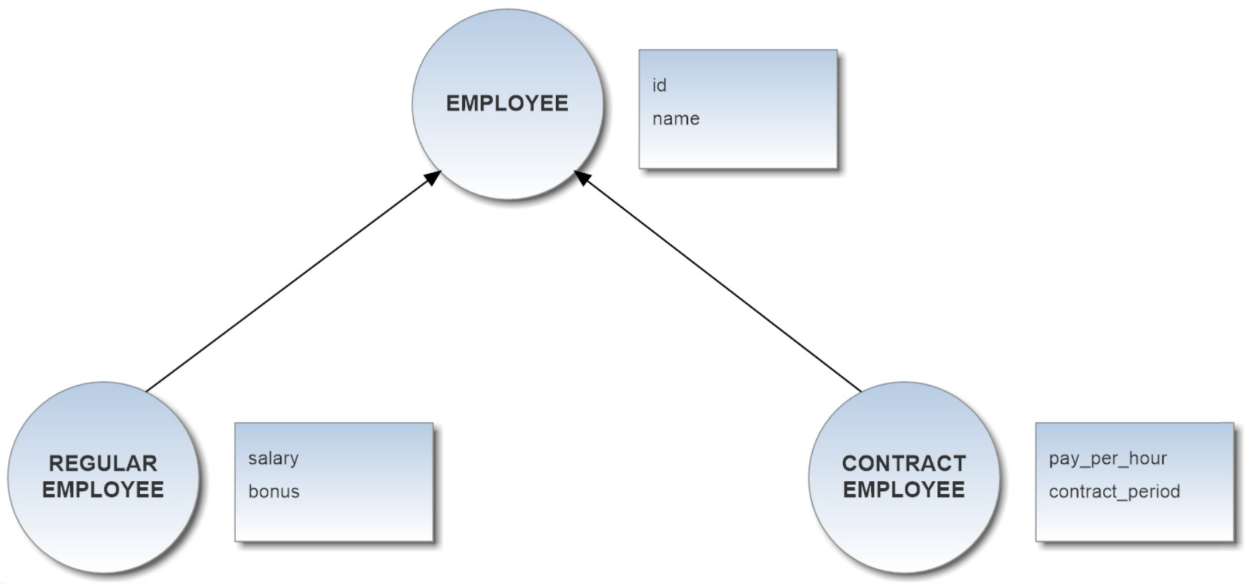
Now suppose you want to map the whole hierarchy given below into a coherent relational database schema. The **Employee** class is a superclass both for **Regular_Employee** and **Contract_Employee** classes.
The application of the above described pattern leads to the DB schema shown in the following diagram:


EMPLOYEE
id : int
name : String
REGULAR EMPLOYEE
salary : float
bonus : int
CONTRACT EMPLOYEE
pay_per_hour : float
contract_period : String
EMPLOYEE
id
name
REGULAR EMPLOYEE
id
name
salary
bonus
CONTRACT EMPLOYEE
id
name
pay_per_hour
contract_period

The correspondent mapping file for this hierarchy should be:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="Employee" table="EMPLOYEE">
        <id name="id" column="id">
            <generator class="increment"></generator>
        </id>

        <property name="name"  column="name"></property>

        <union-subclass name="Regular_Employee"
            table="REGULAR EMPLOYEE">
            <property name="salary" column="salary"></property>
            <property name="bonus" column="bonus"></property>
        </union-subclass>

        <union-subclass name="Contract_Employee"
            table="CONTRACT EMPLOYEE">
            <property name="payPerHour" column="pay_per_hour"></property>
            <property name="contractDuration" column="contract_duration"></property>
        </union-subclass>
    </class>
</hibernate-mapping>
```

Taking advantage of this inheritance-feature on the proposed model above, you will get the following schema in OrientDB:



If you deal with a multi-level inheritance relationships in the DB, you have to represent them in the ORM file by recursively nesting each definition according to the hierarchical dependences being between the Entities of the model.

Import Configuration

It's possible to specify an import configuration by writing down a **configuration file in JSON format** and passing its location to Teleporter through the argument `-conf` :

```
./oteleporter.sh -jdriver <jdbc-driver> -jurl <jdbc-url> -juser <username>
                 -jpasswd <password> -ourl <orientdb-url> [-s <strategy>]
                 [-nr <name-resolver>] [-v <verbose-level>]
                 ([-include <table-names>] | [-exclude <table-names>])
                 [-inheritance <orm-technology>:<ORM-file-url>]
                 [-conf <configuration-file-location>]
```

For example if you want enrich your migration from Postgresql with a configuration file `migration-config.json` located in the `/tmp` folder you can type:

```
./oteleporter.sh -jdriver postgresql -jurl jdbc:postgresql://localhost:5432/testdb
                 -juser username -jpasswd password -ourl plocal:$ORIENTDB_HOME/databases/testdb
                 -conf /tmp/migration-config.json
```

After the first migration, the graph database will be built and the configuration you passed as argument to Teleporter **will be copied into the database** folder in a path like that:

```
ORIENDB_HOME/testdb/teleporter-config/migration-config.json
```

In the following executions the new configuration in your database will be processed automatically, making coherent and simpler the synchronization procedure. If you want change any setting you can modify directly that file.
In fact Teleporter, at execution time, **sequentially looks for**:

1. the configuration file `migration-config.json` in the database directory **ORIENDB_HOME//teleporter-config/**
2. if no config file will be found, then a **potential input config** will be considered
3. if no config file was passed as argument the **migration will be performed without any configuration**

Relationship configuration

The configuration allows you to **manage the relationships of your database domain**.
To comprehend the importance of this feature we have to consider that Teleporter builds the schema in OrientDB and carries out the migration starting from the source DB schema: **Vertices and Edges are built starting from Entities (tables) and Relationships (foreign keys)** which are inferred from your database metadata.
Therefore if you didn't defined some constraints, such as foreign keys between the tables on which you usually perform join operations, you will lose this kind of info during the importing process.
To be clear **if no foreign keys are declared in the schema, you will not have any edges in your final Graph Database**.

So if some constraints are not defined in your schema for performance reasons, submitting a configuration file is essential in order to obtain a complete graph model and perform a good and effective migration to OrientDB.

You can do that by **enriching the basic mapping** of Teleporter between the **E-R model and the Graph Model** to customize your importing. You can add new relationships or modify info about relationships already defined in your database schema, interacting directly on the domains-mapping carried out by Teleporter.
Each Relationship expressed in your schema through a foreign key will be transformed into an Edge class in the graph model according to automatic choices that implicate:

- the **name** of the Edge
- the **direction** of the Edge
- the **properties** of the Edge
- the **type** of each property and potential constraints (mandatory, readOnly, notNull)

So you can intervene in this mapping and make your personal choices.

Let's start to analyse the syntax in order to examine the two main actions you can manage during the migration:

- **adding relationships** not declared in your schema
- **modifying relationships** present in your schema

Adding Relationships

The JSON syntax of the **configuration file** will appear very intuitive if you bear in mind that it **reflects the mapping between the E-R model and the Graph Model**.
Let's consider the configuration below:

```
{
    "edges": [{
        "WorksAtProject": {
            "mapping": {
                "fromTable": "EMPLOYEE",
                "fromColumns": ["PROJECT"],
                "toTable": "PROJECT",
                "toColumns": ["ID"],
                "direction": "direct"
            },
            "properties": {
                "updatedOn": {
                    "type": "DATE",
                    "mandatory": true,
                    "readOnly": false,
                    "notNull": false
                }
            }
        }
    }]
}
```

We are defining all the edges we want to map through the key `edges` which contains an array of elements. Each element in the array is an **Edge class definition containing the mapping with a Relationship** in the relational model.
Let's suppose we have two entities "Employee" and "Project" in our database with a logical Relationship between them: starting from an Employee you can navigate the Projects he's working at.

```
          EMPLOYEE                                                     PROJECT
          (Foreign Table)                                              (Parent Table)
 ________________________________________________                ____________________________________
|       |              |             |           |              |       |         |                   |
|  ID   |  FIRST_NAME  |  LAST_NAME  |  PROJECT  |              |  ID   |  TITLE  |  PROJECT_MANAGER  |
|_______|______________|_____________|___________|              |_______|_________|___________________|
|       |              |             |           |              |       |         |                   |
|       |              |             |           |              |       |         |                   |
|_______|______________|_____________|___________|              |_______|_________|___________________|
```

Without a foreign key definition we lose this Relationship and we obtain a graph model without the correspondent Edge class; consequently no edges between vertices of class "Employee" and vertices of class "Project" will be present.

graph model wihtout edge class --> graph without edges>

Through this mapping we can **overcome the lack of a foreign key** and recover the lost info.
Let's take a look closer to the **edge mapping**:

```
{
    "WorksAtProject": {
        "mapping": {
            "fromTable": "EMPLOYEE",
            "fromColumns": ["PROJECT"],
            "toTable": "PROJECT",
            "toColumns": ["ID"],
            "direction": "direct"
        },
        "properties": {
            "updatedOn": {
                "type": "DATE",
                "mandatory": true,
                "readOnly": false,
                "notNull": false
            }
        }
    }
}
```

We are mapping the Edge class "WorksAtProject" with a Relationship with cardinality 1-N between "Employee" and "Project" on the basis of 4 essential values:

- **fromTable**: the foreign entity that import the primary key of the parent table. In the example is the table "EMPLOYEE".
- **fromColumns**: the attributes involved in the foreign key. In the example it's the field "PROJECT" in the table "EMPLOYEE".
- **toTable**: the parent entity whose primary key is imported by the foreign table. In the example is the table "PROJECT". - **toColumns**: the attributes involved in the primary key imported. In the example it's the field "ID" in the table "PROJECT".

As this Relationship is not declared in your database, it will be added and the correspondent Edge will be built according to the other info you can set.

With the key `direction` you can express the direction of the edges between the vertices. You can set this argument with two different values:

- **direct**: the edge will reflect the direction of the relationship.
- **inverse**: the edge will have opposite direction respect to the relationship.

So if we define in the configuration file a relationship as follows:

```
"WorksAtProject": {
        "mapping": {
            "fromTable": "EMPLOYEE",
            "fromColumns": ["PROJECT"],
            "toTable": "PROJECT",
            "toColumns": ["ID"],
            "direction": "direct"
        },
         ...
}
```

a new Relationship will be added

```
EMPLOYEE -----------------------------> PROJECT
       fromTable    =  EMPOYEE
       toTable      =  PROJECT
       fromColumns  =  [PROJECT]
       TOColumns    =  [ID]
```

and choosing "direct" direction, or don't declaring anything about that, we will obtain an Edge like that:

```
Employee ----[WorksAtProject]----> Project
```

Suppose we want have the **inverse logical navigation** in the graph database that we could not express in the relational model. Here is the configuration we must use:

```
        "HasCommittedEmployee": {
            "mapping": {
                "fromTable": "EMPLOYEE",
                "fromColumns": ["PROJECT"],
                "toTable": "PROJECT",
                "toColumns": ["ID"],
                "direction": "inverse"
            },
            ...
        }
```

In this case the **same relationship** of the previous example will be built

```
EMPLOYEE -----------------------------> PROJECT
       fromTable    =  EMPOYEE
       toTable      =  PROJECT
       fromColumns  =  [PROJECT]
       TOColumns    =  [ID]
```

but the correspondent **Edge will have inverse direction**

```
Employee <----[???]----- Project
```

and for this reason we have changed the name of the Edge in "HasCommittedEmployee" so that the name of the class makes sense:

```
Employee <----[HasCommittedEmployee]----- Project
```

Remember: direction refers to edges in the graph, not to relationships in your database. Relationships must be always coherent with the structure of the tables.

As you can see it's possible to **define additional properties** for the final edge:

```
{
    "WorksAtProject": {
        "mapping": {
            "fromTable": "EMPLOYEE",
            "fromColumns": ["PROJECT"],
            "toTable": "PROJECT",
            "toColumns": ["ID"],
            "direction": "direct"
        },
        "properties": {
            "updatedOn": {
                "type": "DATE",
                "mandatory": true,
                "readOnly": false,
                "notNull": false
            }
        }
    }
}
```

In the example above we added a property named `updatedOn` of type OType.DATE to our Edge class.
For each new defined property you can declare the following values:

- **type**: it's the OrientDB type. This value is mandatory, if not declared the property is not added to the Edge.
- **mandatory**: adds the mandatory constraint to the property and applies to it the specified value (true or false).
- **readOnly**: adds the readOnly constraint to the property and applies to it the specified value (true or false).
- **notNull**: adds the notNull constraint to the property and applies to it the specified value (true or false).

By omitting a constraint or setting it to false you will have the same result: the constraint is not considered for the specific property.

Modifying existent Relationships

If the relationship you are mapping is already present in your database, it will be overridden with the parameters you defined in the configuration.
In this way you can **change the name and the direction of the Edge class correspondent to a relationship already present** in the database schema.
Let's suppose we have a foreign key between "Employee" and "Project":

```
         EMPLOYEE                                                 PROJECT
         (Foreign Table)                                          (Parent Table)
  ______________________________________________            ___________________________________
 |       |              |             |           |        R        |       |         |                  |
 |  ID   |  FIRST_NAME  |  LAST_NAME  |  PROJECT  |   ----------->  |  ID   |  TITLE  |  PROJECT_MANAGER |
 |_______|______________|_____________|___________|                 |_______|_________|__________________|
 |       |              |             |           |                 |       |         |                  |
 |       |              |             |           |                 |       |         |                  |
 |_______|______________|_____________|___________|                 |_______|_________|__________________|
```

In this case through the automated mapping of Teleporter we will obtain the following graph model:

```
Employee ----[HasProject]----> Project
```

In case we want reach a different result from the migration we can **change the attributes of the relationship** declaring them in the mapping.
Teleporter will **recognize the relationship** you want override on the basis of the values:

- **fromTable**
- **fromColumns**
- **toTable**
- **toColumns**

These values must to be coherent with the direction of the relationship defined in the db schema, otherwise Teleporter will interpret the relationship as a new one.
So if for example we want override the Edge built starting from the relationship

```
EMPLOYEE -----------------------------> PROJECT
       fromTable    =  EMPOYEE
       toTable      =  PROJECT
       fromColumns  =  [PROJECT]
       TOColumns    =  [ID]
```

but we define the mapping as follows:

```
    "WorksAtProject": {
        "mapping": {
            "fromTable": "PROJECT",
            "fromColumns": ["ID"],
            "toTable": "EMPLOYEE",
            "toColumns": ["PROJECT"],
            "direction": "direct"
        },
        ...
    }
```

as result we will obtain the adding of a second relationship with inverted direction between the two tables:

```
PROJECT ----------------------------> EMPLOYEE
       fromTable    =  PROJECT
       toTable      =  EMPLOYEE
       fromColumns  =  [ID]
       TOColumns    =  [PROJECT]
```

So in the graph model we will have two Edge classes (the second one is totally wrong):

```
Employee ----[HasProject]-------> Project
Employee <---[WorksAtProject]---- Project
```

So remember to **be coherent with the underlying schema** during the mapping definition:

Mapping:

```
    "WorksAtProject": {
        "mapping": {
            "fromTable": "EMPLOYEE",
            "fromColumns": ["PROJECT"],
            "toTable": "PROJECT",
            "toColumns": ["ID"],
            "direction": "direct"
        },
        ...
    }
```

Relationship in the E-R model:

```
EMPLOYEE -----------------------------> PROJECT
       fromTable    =  EMPOYEE
       toTable      =  PROJECT
       fromColumns  =  [PROJECT]
       TOColumns    =  [ID]
```

Resulting Graph Model:

```
Employee ----[WorksAtProject]-------> Project
```

If you want **change the direction of the Edge** you can exploit the option `direction` as described in the previous paragraph:

```
    "HasCommittedEmployee": {
        "mapping": {
            "fromTable": "EMPLOYEE",
            "fromColumns": ["PROJECT"],
            "toTable": "PROJECT",
            "toColumns": ["ID"],
            "direction": "inverse"
        }
    }
```

Relationship in the E-R model:

```
EMPLOYEE -----------------------------> PROJECT
       fromTable    =  EMPOYEE
       toTable      =  PROJECT
       fromColumns  =  [PROJECT]
       TOColumns    =  [ID]
```

Resulting Graph Model:

```
Project ----[HasCommittedEmployee]-------> Employee
```

You can also **add properties** to the Edge class using the syntax already defined in the previous paragraph:

```
{
    "WorksAtProject": {
        "mapping": {
            "fromTable": "EMPLOYEE",
            "fromColumns": ["PROJECT"],
            "toTable": "PROJECT",
            "toColumns": ["ID"],
            "direction": "direct"
        },
        "properties": {
            "updatedOn": {
                "type": "DATE",
                "mandatory": true,
                "readOnly": false,
                "notNull": false
            }
        }
    }
}
```

In the example above we added a properties with name "updatedOn" of type OType.DATE to our Edge class and we set only the constraint mandatory.

Configuring aggregation strategy

Teleporter offers two importing strategies as described in the Execution strategies page:

- **naive** strategy: no aggregations are executed
- **naive-aggregate** strategy: aggregations can be executed

The aggregation is performed on join tables of dimension equals to 2 (other join tables are ignored), that is to say those tables which allow joins only between two tables.
Each candidate join table is converted into an appropriate Edge class, and each field not involved in any relationship with other tables (hence not involved in any foreign key in the source database schema) is aggregated in the properties of the new built Edge.

If no foreign keys are defined for a specific join table the aggregation will not be performed and no edges will represent the N-N relationship.
Through the configuration you can overcome this limit and kill two birds with one stone: in fact you can **declare the two relationships with the external tables and define the mapping with an Aggregator-Edge in one shot**.

Let's suppose we have a N-N relationship between two tables "Film" and "Actor" without foreign keys defined in the schema.

```
                ACTOR                                        ACTOR_FILM                                   FILM
                                                            (Join Table)

   ____________________________________          ____________________________________          ______________________________
  |       |              |             |        |            |           |           |        |       |         |            |
  |  ID   |  FIRST_NAME  |  LAST_NAME  |        |  ACTOR_ID  |  FILM_ID  |  PAYMENT  |        |  ID   |  TITLE  |  CATEGORY  |
  |_______|______________|_____________|        |____________|___________|___________|        |_______|_________|____________|
  |       |              |             |        |            |           |           |        |       |         |            |
  |       |              |             |        |            |           |           |        |       |         |            |
  |_______|______________|_____________|        |____________|___________|___________|        |_______|_________|____________|
```

We want obtain an aggregated structure as follows:

```
Actor ----[Performs]-------> Film
```

In this case we have to use this syntax:

```
{
    "Performs": {
        "mapping": {
            "fromTable": "ACTOR",
            "fromColumns": ["ID"],
            "toTable": "FILM",
            "toColumns": ["ID"],
            "joinTable": {
                "tableName": "ACTOR_FILM",
                "fromColumns": ["ACTOR_ID"],
                "toColumns": ["FILM_ID"]
            },
            "direction": "direct"
        },
        "properties": {
         ...
        }
    }
}
```

We can implicitly define the **direction** of the Edge by choosing:

- the **from-table**
- the **to-table**

In our example we decided to express the relationship between Actors and Films through a "Performs" Edge starting from Actor vertices and ending into Film vertices.

Once again we can exploit the semantic of the `direction` key to reverse the final edge:

```
{
    "HasActor": {
        "mapping": {
            "fromTable": "ACTOR",
            "fromColumns": ["ID"],
            "toTable": "FILM",
            "toColumns": ["ID"],
            "joinTable": {
                "tableName": "ACTOR_FILM",
                "fromColumns": ["ACTOR_ID"],
                "toColumns": ["FILM_ID"]
            },
            "direction": "inverse"
        },
        "properties": {

        }
    }
}
```

In this way we can have the following schema:

```
Film ------[HasActor]------> Actor
```

When you are taking advantage of this aggregating feature you have to define an **additional field** `join table` as shown in the example.

```
    "joinTable": {
        "tableName": "ACTOR_FILM",
        "fromColumns": ["ACTOR_ID"],
        "toColumns": ["FILM_ID"]
    }
```

In this field you have to specify:

- **tableName**: the name of the join table which will be aggregated into the declared Edge.
- **fromColumns**: the columns of the join table involved in the relationship with the "from-table".
- **toColumns**: the columns of the join table involved in the relationship with the "to-table".

This info are essential for Teleporter to infer all the single relationships between the records of the two external tables "ACTOR" and "FILM" and to build all the edges coherently, so if you don't declare any of these fields an **exception** will be thrown.

Remember that this syntax offers a shortcut to configure relationships and aggregation choices, thus **you can use it only when you are executing the aggregation strategy**.
Performing your migration with a naive strategy using this syntax makes no sense, and here again an exception will be thrown.

Troubleshooting

This page aims to link all the guides to Problems and Troubleshooting.

Sub sections

- Troubleshooting Java API

Topics

Best practice to map the RID in REST Full friendly ID representation

Take a look at HashID. HashID should get you a hashed Rid, which is also convertible, so it won't take up more storage space (like with a UUID). It will just take a small bit of CPU time.

Please note, this little tool is not in any way a true hash, as in, it makes it very hard to crack the hash. It is more about good obfuscation. If you are at all worried about the Rids being known, this isn't a proper solution.

Why can't I see all the edges?

OrientDB, by default, manages edges as "lightweight" edges if they have no properties. This means that if an edge has no properties, it's not stored as physical record. But don't worry, your edge is still there but encoded in a separate data structure. For this reason if you execute a `select from E` no edges or less edges than expected are returned. It's extremely rare the need to have the list of edges, but if this is your case you can disable this feature by issuing this command once (with a slow down and a bigger database size):

```
ALTER DATABASE custom useLightweightEdges=false
```

Use ISO 8601 Dates

According to ISO 8601, Combined date and time in UTC: 2014-12-20T00:00:00. To use this standard change the datetimeformat in the database:

```
ALTER DATABASE DATETIMEFORMAT yyyy-MM-dd'T'HH:mm:ss.SSS'Z'
```

JVM crash on Solaris and other *NIX platforms.

The reason of this issue is massive usage of sun.misc.Unsafe which may have different contract than it is implemented for Linux and Windows JDKs. To avoid this error please use following settings during server start:

```
java ... -Dmemory.useUnsafe=false and -Dstorage.compressionMethod=gzip ...
```

Error occurred while locking memory: Unable to lock JVM memory. This can result in part of the JVM being swapped out, especially if mmapping of files enabled. Increase RLIMIT_MEMLOCK or run OrientDB server as root(ENOMEM)

Don't be scared about it: your OrientDB installation will work perfectly, just it could be slower with database larger than memory.

This lock is needed in case of you work on OS which uses aggressive swapping like Linux. If there is the case when amount of available RAM is not enough to cache all MMAP content OS can swap out rarely used parts of Java heap to the disk and when GC is started to collect garbage we will have performance degradation, to prevent such situation Java heap is locked into memory and prohibited to be flushed on the disk.

com.orientechnologies.orient.core.exception.OStorageException: Error on reading record from file 'default.0.oda', position 2333, size 122,14Mb: the record size is bigger then the file itself (233,99Kb)

This usually happens because the database has been corrupted by a hw/sw crash or a hard kill of the process during the writing to disk. If this happens on index clusters just rebuild indexes, otherwise re-import a previously exported database.

Class 'OUSER' or 'OROLE' was not found in current database

Look at: Restore admin user.

User 'admin' was not found in current database

Look at: Restore admin user.

WARNING: Connection re-acquired transparently after XXXms and Y retries: no errors will be thrown at application level

This means that probably default timeouts are too low and server side operation need more time to complete. Follow these Performance Tuning.

Record id invalid -1:-2

This message is relative to a temporary record id generated inside a transaction. For more information look at Transactions. This means that the record hasn't been correctly serialized.

Brand new records are created with version greater than 0

This happens in graphs. Think to this graph of records:

A -> B -> C -> A

When OrientDB starts to serialize records goes recursively from the root A. When A is encountered again to avoid loops it saves the record as empty just to get the RecordID to store into the record C. When the serialization stack ends the record A (that was the first of the stack) is updated because has been created as first but empty.

Error: com.orientechnologies.orient.core.exception.OStorageException: Cannot open local storage '/tmp/databases/demo' with mode=rw

com.orientechnologies.common.concur.lock.OLockException: File '/tmp/databases/demo/default.0.oda' is locked by another process, maybe the database is in use by another process. Use the remote mode with a OrientDB server to allow multiple access to the same database

Both errors have the same meaning: a "plocal" database can't be opened by multiple JVM at the same time. To fix:

- check if there's no process using OrientDB (most of the times a OrientDB Server is running in the background). Just shutdown that server and retry
- if you need multiple access to the same database, don't use "plocal" directly, but rather start a server and access to the database by using "remote" protocol. In this way the server is able to share the same database with multiple clients.

Caused by: java.lang.NumberFormatException: For input string: "500Mb"

You're using different version of libraries. For example the client is using 1.3 and the server 1.4. Align the libraries to the same version (last is suggested). Or probably you've different versions of the same jars in the classpath.

Troubleshooting using Java API

OConcurrentModificationException: Cannot update record #X:Y in storage 'Z' because the version is not the latest. Probably you are updating an old record or it has been modified by another user (db=vA your=vB)

This exception happens because you're running in a Multi Version Control Check (MVCC) system and another thread/user has updated the record you're saving. For more information about this topic look at Concurrency. To fix this problem you can:

- Change the Graph consistency level to don't use transactions.
- Or write code concurrency proof.

Example:

```
for (int retry = 0; retry < maxRetries; ++retry) {
  try {
    // APPLY CHANGES
    document.field(name, "Luca");

    document.save();
    break;
  } catch(ONeedRetryException e) {
    // RELOAD IT TO GET LAST VERSION
    document.reload();
  }
}
```

The same in transactions:

```
for (int retry = 0; retry < maxRetries; ++retry) {
  db.begin();
  try {
    // CREATE A NEW ITEM
    ODocument invoiceItem = new ODocument("InvoiceItem");
    invoiceItem.field(price, 213231);
    invoiceItem.save();

    // ADD IT TO THE INVOICE
    Collection<ODocument> items = invoice.field(items);
    items.add(invoiceItem);
    invoice.save();

    db.commit();
    break;
  } catch (OTransactionException e) {
    // RELOAD IT TO GET LAST VERSION
    invoice.reload();
  }
}
```

Where `maxRetries` is the maximum number of attempt of reloading.

Run in OSGi context

(by Raman Gupta) OrientDB uses ServiceRegistry to load OIndexFactory and some OSGi containers might not work with it.

One solution is to set the TCCL so that the ServiceRegistry lookup works inside of OSGi:

```
ODatabaseObjectTx db = null;
ClassLoader origClassLoader = Thread.currentThread().getContextClassLoader();
try {
  ClassLoader orientClassLoader = OIndexes.class.getClassLoader();
  Thread.currentThread().setContextClassLoader(orientClassLoader);
  db = objectConnectionPool.acquire(dbUrl, username, password);
} finally {
  Thread.currentThread().setContextClassLoader(origClassLoader);
}
```

Because the ServiceLoader uses the thread context classloader, you can configure it to use the classloader of the OrientDB bundle so that it finds the entries in META-INF/services.

Another way is to embed the dependencies in configuration in the Maven pom.xml file under plugin(maven-bundle-plugin)/configuration/instructions:

```
<Embed-Dependency>
  orientdb-client,
  orient-commons,
  orientdb-core,
  orientdb-enterprise,
  orientdb-object,
  javassist
</Embed-Dependency>
```

Including only the jars you need. Look at Which library do I use?

Database instance has been released to the pool. Get another database instance from the pool with the right username and password

This is a generic error telling that the database has been found closed while using it.

Check the stack trace to find the reason of it:

OLazyObjectIterator

This is the case when you're working with Object Database API and a field contains a collection or a map loaded in lazy. On iteration it needs an open database to fetch linked records.

Solutions:

- assure to leave the database open while browsing the field
- or early load all the instances (just iterate the items)
- define a fetch-plan to load the entire object tree in one shoot and then work offline. If you need to save the object back to the database then reopen the database and call `db.save( object )` .

Stack Overflow on saving objects

This could be due to the high deep of the graph, usually when you create many records. To fix it save the records more often.

Query Examples

This pages collects example of query from users. Feel free to add your own use case and query to help further users.

How to ask the graph what relationships exist between two vertices? In my case I have two known 'Person' nodes each connected via a 'member_of' edge to a shared 'Organization' node. Each 'Person' is also a 'member_of' other 'Organization's.

```
select intersect(out('member_of').org_name) from
 Person where name in ["nagu", "rohit"]
```

This example shows how to form where clause in order to query/ filter based on properties of connected vertices.

DocElem and Model are subclasses of V and hasModel of E.

```
    insert into DocElem set uri = 'domain.tdl', type = "paragraph"
    insert into Model set hash = '0e1f', model = "hello world"
    create edge hasModel from #12:2738 to #13:2658
```

User wishes to query those vertices filtering on certain properties of DocElem and Model.

To fetch the Model vertices where DocElem.type = "paragraph" and connected vertex Model has the property model like '%world%'

```
    select from (select expand(out('hasModel')) from DocElem where
      type = "paragraph") where model like "%world%"
```

To find instead the DocElem vertices, use this (assuming that a DocElem is only connected to one Model):

```
    select * from DocElem where type = "paragraph" and
      out('hasModel')[0].model like '%world%'
```

How to apply built-in math functions on projections? For example, to use the sum() function over 2 values returned from sub-queries using projections, the following syntax may be used:

```
  select sum($a[0].count,$b[0].count)
    let $a = (select count(*) from e),
        $b = (select count(*) from v)
```

Given the following schema: Vertices are connected with Edges of type *RELATED* which have property *count*. 2 vertices can have connection in both ways at the same time.

V1--RELATED(count=17)-->V2

V2--RELATED(count=3)-->V1

Need to build a query that, for a given vertex *Vn*, will find all vertices connected with *RELATED* edge to this *Vn* and also, for each pair *[Vn, Vx]* will calculate *SUM* of *in_RELATED.count* and *out_RELATED.count*.

For that simple example above, this query result for V1 would be

Vertex	Count
V2	20

Solution:

```
insert into V (name) values ('V1'),('V2')
create edge E from (select from V where name = 'V1') to (select from V where name = 'V2') set count = 17
create edge E from (select from V where name = 'V2') to (select from V where name = 'V1') set count = 3

select v.name, sum(count) as cnt from (
  select if(eval("in=#9:0"),out,in) as v,count from E where (
    in=#9:0 or out=#9:0)
  ) group by v order by cnt desc
```

(replace `#9:0` in the select statement to fit your specific case).

This was discussed in the google groups over here: "https://groups.google.com/forum/#!topic/orient-database/CRR-simpmLg". Thanks to Andrey for posing the problem.

Performance Tuning

This guide contains the general tips to optimize your application that use the OrientDB. Below you can find links for the specific guides different per database type used. Look at the specific guides based on the database type you're using:

- Document Database performance tuning
- Object Database performance tuning
- Distributed Configuration tuning

I/O benchmark

The main requirement for a fast DBMS is having good I/O. In order to understand the performance of your hw/sw configuration. If you have a Unix derived OS (like Linux, MacOSX, etc.), the simplest way to have your raw I/O performance is running this two commands:

```
dd if=/dev/zero of=/tmp/output.img bs=8k count=256k
rm /tmp/output.img
```

This is the output on a fast SSD (1.4 GB/sec):

```
262144+0 records in
262144+0 records out
2147483648 bytes transferred in 1.467536 secs (1463326070 bytes/sec)
```

And this is what you usually get with a HD connected with a USB 3.0 (90 MB/sec):

```
262144+0 records in
262144+0 records out
2147483648 bytes transferred in 23.699740 secs (90612119 bytes/sec)
```

As you can notice the first configuration (SSD) is 16x faster than the second configuration (HD). Sensible differences can be found between bare metal hw and Virtual Machines.

Java

OrientDB is written in Java, so it runs on top of Java Virtual Machine (JVM). OrientDB is compatible with Java 8 and we suggest to use this version to run OrientDB. Java 8 is faster than Java 7 and previous ones.

JMX

Starting from v2.1, OrientDB exposes internal metrics through JMX Beans. Use this information to track and profile OrientDB.

Memory settings

Server and Embedded settings

These settings are valid for both Server component and the JVM where is running the Java application that use OrientDB in Embedded Mode, by using directly plocal.

The most important thing on tuning is assuring the memory settings are correct. What can make the real difference is the right balancing between the heap and the virtual memory used by Memory Mapping, specially on large datasets (GBs, TBs and more) where the in memory cache structures count less than raw IO.

For example if you can assign maximum 8GB to the Java process, it's usually better assigning small heap and large disk cache buffer (off-heap memory). So rather than:

```
java -Xmx8g ...
```

You could instead try this:

```
java -Xmx800m -Dstorage.diskCache.bufferSize=7200 ...
```

The **storage.diskCache.bufferSize** setting (with old "local" storage it was **file.mmap.maxMemory**) is in MB and tells how much memory to use for Disk Cache component. By default is 4GB.

NOTE: If the sum of maximum heap and disk cache buffer is too high, could cause the OS to swap with huge slow down.

JVM settings

JVM settings are encoded in server.sh (and server.bat) batch files. You can change them to tune the JVM according to your usage and hw/sw settings. We found these setting work well on most configurations:

```
-server -XX:+PerfDisableSharedMem
```

This setting will disable writing debug information about the JVM. In case you need to profile the JVM, just remove this setting. For more information look at this post: http://www.evanjones.ca/jvm-mmap-pause.html.

High concurrent updates

OrientDB has an optimistic concurrency control system, but on very high concurrent updates on the few records it could be more efficient locking records to avoid retries. You could synchronize the access by yourself or by using the storage API. Note that this works only with non-remote databases.

```
((OStorageEmbedded)db.getStorage()).acquireWriteLock(final ORID iRid)
((OStorageEmbedded)db.getStorage()).acquireSharedLock(final ORID iRid)
((OStorageEmbedded)db.getStorage()).releaseWriteLock(final ORID iRid)
((OStorageEmbedded)db.getStorage()).releaseSharedLock(final ORID iRid)
```

Example of usage. Writer threads:

```
try{
  ((OStorageEmbedded)db.getStorage()).acquireWriteLock(record.getIdentity());

  // DO SOMETHING
} finally {
  ((OStorageEmbedded)db.getStorage()).releaseWriteLock(record.getIdentity());
}
```

Reader threads:

```
try{
  ((OStorageEmbedded)db.getStorage()).acquireSharedLock(record.getIdentity());
  // DO SOMETHING

} finally {
  ((OStorageEmbedded)db.getStorage()).releaseSharedLock(record.getIdentity());
}
```

Remote connections

There are many ways to improve performance when you access to the database using the remote connection.

Fetching strategy

When you work with a remote database you've to pay attention to the fetching strategy used. By default OrientDB Client loads only the record contained in the result set. For example if a query returns 100 elements, but then you cross these elements from the client, then OrientDB client lazily loads the elements with one more network call to the server foreach missed record.

By specifying a fetch plan when you execute a command you're telling to OrientDB to prefetch the elements you know the client application will access. By specifying a complete fetch plan you could receive the entire result in *just one network call*.

For more information look at: Fetching-Strategies.

Network Connection Pool

Each client, by default, uses only one network connection to talk with the server. Multiple threads on the same client share the same network connection pool.

When you've multiple threads could be a bottleneck since a lot of time is spent on waiting for a free network connection. This is the reason why is much important to configure the network connection pool.

The configurations is very simple, just 2 parameters:

- **minPool**, is the initial size of the connection pool. The default value is configured as global parameters "client.channel.minPool" (see parameters)
- **maxPool**, is the maximum size the connection pool can reach. The default value is configured as global parameters "client.channel.maxPool" (see parameters)

At first connection the **minPool** is used to pre-create network connections against the server. When a client thread is asking for a connection and all the pool is busy, then it tries to create a new connection until **maxPool** is reached.

If all the pool connections are busy, then the client thread will wait for the first free connection.

Example of configuration by using database properties:

```
database = new ODatabaseDocumentTx("remote:localhost/demo");
database.setProperty("minPool", 2);
database.setProperty("maxPool", 5);

database.open("admin", "admin");
```

Enlarge timeouts

If you see a lot of messages like:

```
WARNING: Connection re-acquired transparently after XXXms and Y retries: no errors will be thrown at application level
```

means that probably default timeouts are too low and server side operation need more time to complete. It's strongly suggested you enlarge your timeout only after tried to enlarge the Network Connection Pool. The timeout parameters to tune are:

- `network.lockTimeout` , the timeout in ms to acquire a lock against a channel. The default is 15 seconds.
- `network.socketTimeout` , the TCP/IP Socket timeout in ms. The default is 10 seconds.

Query

Use of indexes

The first improvement to speed up queries is to create Indexes against the fields used in WHERE conditions. For example this query:

```
SELECT FROM Profile WHERE name = 'Jay'
```

Browses the entire "profile" cluster looking for records that satisfy the conditions. The solution is to create an index against the 'name' property with:

```
CREATE INDEX profile.name UNIQUE
```

Use NOTUNIQUE instead of UNIQUE if the value is not unique.

For more complex queries like

```
SELECT * FROM testClass WHERE prop1 = ? AND prop2 = ?
```

Composite index should be used

```
CREATE INDEX compositeIndex ON testClass (prop1, prop2) UNIQUE
```

or via Java API:

```
oClass.createIndex("compositeIndex", OClass.INDEX_TYPE.UNIQUE, "prop1", "prop2");
```

Moreover, because of partial match searching, this index will be used for optimizing query like

```
SELECT * FROM testClass WHERE prop1 = ?
```

For deep understanding of query optimization look at the unit test.

Use parameters instead of hardwired values

Query parsing is not an extremely expensive operation, but zero cost is better than low cost, right?

When you execute an SQL query, OrientDB parses the query text and produces an AST. This structure is cached at storage level, so if you execute the query again, OrientDB will avoid to parse it again and will just pick the AST from the cache.

In most cases, your application will execute a very limited number of queries, but with different condition values, eg.

```
SELECT FROM Person WHERE name = 'Joe';
SELECT FROM Person WHERE name = 'Jenny';
SELECT FROM Person WHERE name = 'Frank';
SELECT FROM Person WHERE name = 'Anne';
```

With these queries, OrientDB has to perform parsing operations each time.

Re-writing the query as follows:

```
SELECT FROM Person WHERE name = ?;
```

or

```
SELECT FROM Person WHERE name = :theName;
```

and passing the name as a parameter (see the docs for your language driver, eg TinkerPop API) allows OrientDB to parse the query only once and then cache it. The second time you execute the same query, even with different parameters, there will be no parsing at all.

Parallel queries

Starting from v2.2, the OrientDB SQL executor will decide if execute or not a query in parallel. To tune parallel query execution these are the new settings:

- `query.parallelAuto` enable automatic parallel query, if requirements are met. By default is true if your system has more than 2 CPUs/Cores.
- `query.parallelMinimumRecords` is the minimum number of records to activate parallel query automatically. Default is 300,000.
- `query.parallelResultQueueSize` is the size of the queue that holds results on parallel execution. The queue is blocking, so in case

the queue is full, the query threads will be in a wait state. Default is 20,000 results.

Massive Insertion

Use the Massive Insert intent

Intents suggest to OrientDB what you're going to do. In this case you're telling to OrientDB that you're executing a massive insertion. OrientDB auto-reconfigure itself to obtain the best performance. When done you can remove the intent just setting it to null.

Example:

```
db.declareIntent( new OIntentMassiveInsert() );

// YOUR MASSIVE INSERTION

db.declareIntent( null );
```

Disable Journal

In case of massive insertion, specially when this operation is made just once, you could disable the journal (WAL) to improve insertion speed:

```
-storage.useWAL=false
```

By default WAL (Write Ahead Log) is enabled.

Disable sync on flush of pages

This setting avoids to execute a sync at OS level when a page is flushed. Disabling this setting will improve throughput on writes:

```
-Dstorage.wal.syncOnPageFlush=false
```

Massive Updates

Updates generates "holes" at Storage level because rarely the new record fits perfectly the size of the previous one. Holes are free spaces between data. Holes are recycled but an excessive number of small holes it's the same as having a highly defragmented File System: space is wasted (because small holes can't be easily recycled) and performance degrades when the database growth.

Oversize

If you know you will update certain type of records, create a class for them and set the Oversize (default is 0) to 2 or more.

By default the OGraphVertex class has an oversize value setted at 2. If you define your own classes set this value at least at 2.

OClass myClass = getMetadata().getSchema().createClass("Car"); myClass.setOverSize(2);

Wise use of transactions

To obtain real linear performance with OrientDB you should avoid to use Transactions as far as you can. In facts OrientDB keeps in memory all the changes until you flush it with a commit. So the bottleneck is your Heap space and the management of local transaction cache (implemented as a Map).

Transactions slow down massive inserts unless you're using a "remote" connection. In that case it speeds up all the insertion because the client/server communication happens only at commit time.

Disable Transaction Log

If you need to group operations to speed up remote execution in a logical transaction but renouncing to the Transaction Log, just disable it by setting the property **tx.useLog** to false.

Via JVM configuration:

```
java ... -Dtx.useLog=false ...
```

or via API:

```
OGlobalConfiguration.TX_USE_LOG.setValue(false);
```

NOTE: Please note that in case of crash of the JVM the pending transaction OrientDB could not be able to rollback it.

Use the schema

Starting from OrientDB 2.0, if fields are declared in the schema, field names are not stored in document/vertex/edge themselves. This improves performance and saves a lot of space on disk.

Configuration

To tune OrientDB look at the Configuration settings.

Platforms

- Performance analysis on ZFS

Global Configuration

OrientDB can be configured in several ways. To know the current settings use the console with the config command.

Change settings

By command line

You can pass settings via command line when the JVM is launched. This is typically stored inside server.sh (or server.bat on Windows):

```
java -Dcache.size=10000 -Dstorage.keepOpen=true ...
```

By server configuration

Put in the `<properties>` section of the file **orientdb-server-config.xml** (or orientdb-dserver-config.xml) the entries to configure. Example:

```
  ...
  <properties>
    <entry name="cache.size" value="10000" />
    <entry name="storage.keepOpen" value="true" />
  </properties>
  ...
```

At run-time

```
OGlobalConfiguration.MVRBTREE_NODE_PAGE_SIZE.setValue(2048);
```

Dump the configuration

To dump the OrientDB configuration you can set a parameter at JVM launch:

```
java -Denvironment.dumpCfgAtStartup=true ...
```

Or via API at any time:

```
OGlobalConfiguration.dumpConfiguration(System.out);
```

Parameters

To know more look at the Java enumeration: OGlobalConfiguration.java. NOTE: *The documentation below is auto-generated by parsing the OGlobalConfiguration.java class. Please do not edit the parameters here, but rather edit the documentation in the java class.*

Environment

environment.dumpCfgAtStartup

Dumps the configuration during application startup.

```
Setting name...: environment.dumpCfgAtStartup
Default value..: false
Set at run-time: false
```

environment.concurrent

Specifies if running in multi-thread environment. Setting this to false turns off the internal lock management.

```
Setting name...: environment.concurrent
Default value..: true
Set at run-time: false
```

environment.lockManager.concurrency.level

Concurrency level of lock manager.

```
Setting name...: environment.lockManager.concurrency.level
Default value..: 64
Set at run-time: false
```

environment.allowJVMShutdown

Allows the shutdown of the JVM, if needed/requested.

```
Setting name...: environment.allowJVMShutdown
Default value..: true
Set at run-time: true
```

Script

script.pool.maxSize

Maximum number of instances in the pool of script engines.

```
Setting name...: script.pool.maxSize
Default value..: 20
Set at run-time: false
```

Memory

memory.useUnsafe

Indicates whether Unsafe will be used, if it is present.

```
Setting name...: memory.useUnsafe
Default value..: true
Set at run-time: false
```

memory.chunk.size

Size of single memory chunk (in bytes) which will be preallocated by OrientDB.

```
Setting name...: memory.chunk.size
Default value..: 2147483647
Set at run-time: false
```

memory.directMemory.safeMode

Indicates whether to perform a range check before each direct memory update. It is true by default, but usually it can be safely set to false. It should only be to true after dramatic changes have been made in the storage structures.

```
Setting name...: memory.directMemory.safeMode
Default value..: true
Set at run-time: false
```

memory.directMemory.trackMode

Activates the direct memory pool leak detector. This detector causes a large overhead and should be used for debugging purposes only. It's also a good idea to pass the -Djava.util.logging.manager=com.orientechnologies.common.log.OLogManager$DebugLogManager switch to the JVM, if you use this mode, this will enable the logging from JVM shutdown hooks..

```
Setting name...: memory.directMemory.trackMode
Default value..: false
Set at run-time: false
```

memory.directMemory.onlyAlignedMemoryAccess

Some architectures do not allow unaligned memory access or may suffer from speed degradation. For such platforms, this flag should be set to true.

```
Setting name...: memory.directMemory.onlyAlignedMemoryAccess
Default value..: true
Set at run-time: false
```

Jvm

jvm.gc.delayForOptimize

Minimal amount of time (in seconds), since the last System.gc(), when called after tree optimization.

```
Setting name...: jvm.gc.delayForOptimize
Default value..: 600
Set at run-time: false
```

Storage

storage.openFiles.limit

Limit of amount of files which may be open simultaneously.

```
Setting name...: storage.openFiles.limit
Default value..: 512
Set at run-time: false
```

storage.componentsLock.cache

Amount of cached locks is used for component lock to avoid constant creation of new lock instances.

```
Setting name...: storage.componentsLock.cache
Default value..: 10000
Set at run-time: false
```

storage.diskCache.pinnedPages

Maximum amount of pinned pages which may be contained in cache, if this percent is reached next pages will be left in unpinned state. You can not set value more than 50.

```
Setting name...: storage.diskCache.pinnedPages
Default value..: 20
Set at run-time: false
```

storage.diskCache.bufferSize

Size of disk buffer in megabytes, disk size may be changed at runtime, but if does not enough to contain all pinned pages exception will be thrown.

```
Setting name...: storage.diskCache.bufferSize
Default value..: 4096
Set at run-time: true
```

storage.diskCache.writeCachePart

Percentage of disk cache, which is used as write cache.

```
Setting name...: storage.diskCache.writeCachePart
Default value..: 15
Set at run-time: false
```

storage.diskCache.writeCachePageTTL

Max time until a page will be flushed from write cache (in seconds).

```
Setting name...: storage.diskCache.writeCachePageTTL
Default value..: 86400
Set at run-time: false
```

storage.diskCache.writeCachePageFlushInterval

Interval between flushing of pages from write cache (in ms).

```
Setting name...: storage.diskCache.writeCachePageFlushInterval
Default value..: 25
Set at run-time: false
```

storage.diskCache.writeCacheFlushInactivityInterval

Interval between 2 writes to the disk cache, if writes are done with an interval more than provided, all files will be fsynced before the next write, which allows a data restore after a server crash (in ms).

```
Setting name...: storage.diskCache.writeCacheFlushInactivityInterval
Default value..: 60000
Set at run-time: false
```

storage.diskCache.writeCacheFlushLockTimeout

Maximum amount of time the write cache will wait before a page flushes (in ms, -1 to disable).

```
Setting name...: storage.diskCache.writeCacheFlushLockTimeout
Default value..: -1
Set at run-time: false
```

storage.diskCache.diskFreeSpaceCheckInterval

The interval (in seconds), after which the storage periodically checks whether the amount of free disk space is enough to work in write mode.

```
Setting name...: storage.diskCache.diskFreeSpaceCheckInterval
Default value..: 5
Set at run-time: false
```

storage.diskCache.diskFreeSpaceCheckIntervalInPages

The interval (how many new pages should be added before free space will be checked), after which the storage periodically checks whether the amount of free disk space is enough to work in write mode.

```
Setting name...: storage.diskCache.diskFreeSpaceCheckIntervalInPages
Default value..: 2048
Set at run-time: false
```

storage.diskCache.keepState

Keep disk cache state between moment when storage is closed and moment when it is opened again. true by default.

```
Setting name...: storage.diskCache.keepState
Default value..: true
Set at run-time: false
```

storage.configuration.syncOnUpdate

Indicates a force sync should be performed for each update on the storage configuration.

```
Setting name...: storage.configuration.syncOnUpdate
Default value..: true
Set at run-time: false
```

storage.compressionMethod

Record compression method used in storage Possible values : gzip, nothing, snappy, snappy-native. Default is 'nothing' that means no compression.

```
Setting name...: storage.compressionMethod
Default value..: nothing
Set at run-time: false
```

storage.encryptionMethod

Record encryption method used in storage Possible values : 'aes' and 'des'. Default is 'nothing' for no encryption.

```
Setting name...: storage.encryptionMethod
Default value..: nothing
Set at run-time: false
```

storage.encryptionKey

Contains the storage encryption key. This setting is hidden.

```
Setting name...: storage.encryptionKey
Default value..: null
Set at run-time: false
```

storage.makeFullCheckpointAfterCreate

Indicates whether a full checkpoint should be performed, if storage was created.

```
Setting name...: storage.makeFullCheckpointAfterCreate
Default value..: false
Set at run-time: false
```

storage.makeFullCheckpointAfterOpen

Indicates whether a full checkpoint should be performed, if storage was opened. It is needed so fuzzy checkpoints can work properly.

```
Setting name...: storage.makeFullCheckpointAfterOpen
Default value..: true
Set at run-time: false
```

storage.makeFullCheckpointAfterClusterCreate

Indicates whether a full checkpoint should be performed, if storage was opened.

```
Setting name...: storage.makeFullCheckpointAfterClusterCreate
Default value..: true
Set at run-time: false
```

storage.trackChangedRecordsInWAL

If this flag is set metadata which contains rids of changed records is added at the end of each atomic operation.

```
Setting name...: storage.trackChangedRecordsInWAL
Default value..: false
Set at run-time: false
```

storage.useWAL

Whether WAL should be used in paginated storage.

```
Setting name...: storage.useWAL
Default value..: true
Set at run-time: false
```

storage.wal.syncOnPageFlush

Indicates whether a force sync should be performed during WAL page flush.

```
Setting name...: storage.wal.syncOnPageFlush
Default value..: true
Set at run-time: false
```

storage.wal.cacheSize

Maximum size of WAL cache (in amount of WAL pages, each page is 64k) If set to 0, caching will be disabled.

```
Setting name...: storage.wal.cacheSize
Default value..: 3000
Set at run-time: false
```

storage.wal.fileAutoCloseInterval

Interval in seconds after which WAL file will be closed if there is no any IO operations on this file (in seconds), default value is 10.

```
Setting name...: storage.wal.fileAutoCloseInterval
Default value..: 10
Set at run-time: false
```

storage.wal.maxSegmentSize

Maximum size of single WAL segment (in megabytes).

```
Setting name...: storage.wal.maxSegmentSize
Default value..: 128
Set at run-time: false
```

storage.wal.maxSize

Maximum size of WAL on disk (in megabytes).

```
Setting name...: storage.wal.maxSize
Default value..: 4096
Set at run-time: false
```

storage.wal.commitTimeout

Maximum interval between WAL commits (in ms.).

```
Setting name...: storage.wal.commitTimeout
Default value..: 1000
Set at run-time: false
```

storage.wal.shutdownTimeout

Maximum wait interval between events, when the background flush threadreceives a shutdown command and when the background flush will be stopped (in ms.).

```
Setting name...: storage.wal.shutdownTimeout
Default value..: 10000
Set at run-time: false
```

storage.wal.fuzzyCheckpointInterval

Interval between fuzzy checkpoints (in seconds).

```
Setting name...: storage.wal.fuzzyCheckpointInterval
Default value..: 300
Set at run-time: false
```

storage.wal.reportAfterOperationsDuringRestore

Amount of processed log operations, after which status of data restore procedure will be printed (0 or a negative value, disables the logging).

```
Setting name...: storage.wal.reportAfterOperationsDuringRestore
Default value..: 10000
Set at run-time: false
```

storage.wal.restore.batchSize

Amount of WAL records, which are read at once in a single batch during a restore procedure.

```
Setting name...: storage.wal.restore.batchSize
Default value..: 1000
Set at run-time: false
```

storage.wal.readCacheSize

Size of WAL read cache in amount of pages.

```
Setting name...: storage.wal.readCacheSize
Default value..: 1000
Set at run-time: false
```

storage.wal.fuzzyCheckpointShutdownWait

The amount of time the DB should wait until it shuts down (in seconds).

```
Setting name...: storage.wal.fuzzyCheckpointShutdownWait
Default value..: 600
Set at run-time: false
```

storage.wal.fullCheckpointShutdownTimeout

The amount of time the DB will wait, until a checkpoint is finished, during a DB shutdown (in seconds).

```
Setting name...: storage.wal.fullCheckpointShutdownTimeout
Default value..: 600
Set at run-time: false
```

storage.wal.path

Path to the WAL file on the disk. By default, it is placed in the DB directory, but it is highly recommended to use a separate disk to store log operations.

```
Setting name...: storage.wal.path
Default value..: null
Set at run-time: false
```

storage.diskCache.pageSize

Size of page of disk buffer (in kilobytes). !!! NEVER CHANGE THIS VALUE !!!.

```
Setting name...: storage.diskCache.pageSize
Default value..: 64
Set at run-time: false
```

storage.diskCache.diskFreeSpaceLimit

Minimum amount of space on disk, which, when exceeded, will cause the database to switch to read-only mode (in megabytes).

```
Setting name...: storage.diskCache.diskFreeSpaceLimit
Default value..: 256
Set at run-time: false
```

storage.lowestFreeListBound

The least amount of free space (in kb) in a page, which is tracked in paginated storage.

```
Setting name...: storage.lowestFreeListBound
Default value..: 16
Set at run-time: false
```

storage.lockTimeout

Maximum amount of time (in ms) to lock the storage.

```
Setting name...: storage.lockTimeout
Default value..: 0
Set at run-time: false
```

storage.record.lockTimeout

Maximum of time (in ms) to lock a shared record.

```
Setting name...: storage.record.lockTimeout
Default value..: 2000
Set at run-time: false
```

storage.useTombstones

When a record is deleted, the space in the cluster will not be freed, but rather tombstoned.

```
Setting name...: storage.useTombstones
Default value..: false
Set at run-time: false
```

storage.cluster.usecrc32

Indicates whether crc32 should be used for each record to check record integrity.

```
Setting name...: storage.cluster.usecrc32
Default value..: false
Set at run-time: false
```

storage.keepOpen

Deprecated.

```
Setting name...: storage.keepOpen
Default value..: true
Set at run-time: false
```

Record

record.downsizing.enabled

On updates, if the record size is lower than before, this reduces the space taken accordingly. If enabled this could increase defragmentation, but it reduces the used disk space.

```
Setting name...: record.downsizing.enabled
Default value..: true
Set at run-time: false
```

Object

object.saveOnlyDirty

Object Database only! It saves objects bound to dirty records.

```
Setting name...: object.saveOnlyDirty
Default value..: false
Set at run-time: true
```

Db

db.pool.min

Default database pool minimum size.

```
Setting name...: db.pool.min
Default value..: 1
Set at run-time: false
```

db.pool.max

Default database pool maximum size.

```
Setting name...: db.pool.max
Default value..: 100
Set at run-time: false
```

db.pool.idleTimeout

Timeout for checking for free databases in the pool.

```
Setting name...: db.pool.idleTimeout
Default value..: 0
Set at run-time: false
```

db.pool.idleCheckDelay

Delay time on checking for idle databases.

```
Setting name...: db.pool.idleCheckDelay
Default value..: 0
Set at run-time: false
```

db.mvcc.throwfast

Use fast-thrown exceptions for MVCC OConcurrentModificationExceptions. No context information will be available. Set to true, when these exceptions are thrown, but the details are not necessary.

```
Setting name...: db.mvcc.throwfast
Default value..: false
Set at run-time: true
```

db.validation

Enables or disables validation of records.

```
Setting name...: db.validation
Default value..: true
Set at run-time: true
```

db.document.serializer

The default record serializer used by the document database.

```
Setting name...: db.document.serializer
Default value..: ORecordSerializerBinary
Set at run-time: false
```

db.makeFullCheckpointOnIndexChange

When index metadata is changed, a full checkpoint is performed.

```
Setting name...: db.makeFullCheckpointOnIndexChange
Default value..: true
Set at run-time: true
```

db.makeFullCheckpointOnSchemaChange

When index schema is changed, a full checkpoint is performed.

```
Setting name...: db.makeFullCheckpointOnSchemaChange
Default value..: true
Set at run-time: true
```

db.mvcc

Deprecated, MVCC cannot be disabled anymore.

```
Setting name...: db.mvcc
Default value..: true
Set at run-time: false
```

db.use.distributedVersion

Deprecated, distributed version is not used anymore.

```
Setting name...: db.use.distributedVersion
Default value..: false
Set at run-time: false
```

NonTX

nonTX.recordUpdate.synch

Executes a sync against the file-system for every record operation. This slows down record updates, but guarantees reliability on unreliable drives.

```
Setting name...: nonTX.recordUpdate.synch
Default value..: false
Set at run-time: false
```

nonTX.clusters.sync.immediately

List of clusters to sync immediately after update (separated by commas). Can be useful for a manual index.

```
Setting name...: nonTX.clusters.sync.immediately
Default value..: manindex
Set at run-time: false
```

Tx

tx.trackAtomicOperations

This setting is used only for debug purposes. It creates a stack trace of methods, when an atomic operation is started.

```
Setting name...: tx.trackAtomicOperations
Default value..: false
Set at run-time: false
```

tx.commit.synch

Synchronizes the storage after transaction commit.

```
Setting name...: tx.commit.synch
Default value..: false
Set at run-time: false
```

tx.autoRetry

Maximum number of automatic retry if some resource has been locked in the middle of the transaction (Timeout exception).

```
Setting name...: tx.autoRetry
Default value..: 1
Set at run-time: false
```

tx.log.fileType

File type to handle transaction logs: mmap or classic.

```
Setting name...: tx.log.fileType
Default value..: classic
Set at run-time: false
```

tx.log.synch

Executes a synch against the file-system at every log entry. This slows down transactions but guarantee transaction reliability on unreliable drives.

```
Setting name...: tx.log.synch
Default value..: false
Set at run-time: false
```

tx.useLog

Transactions use log file to store temporary data to be rolled back in case of crash.

```
Setting name...: tx.useLog
Default value..: true
Set at run-time: false
```

Index

index.embeddedToSbtreeBonsaiThreshold

Amount of values, after which the index implementation will use an sbtree as a values container. Set to -1, to disable and force using an sbtree.

```
Setting name...: index.embeddedToSbtreeBonsaiThreshold
Default value..: 40
Set at run-time: true
```

index.sbtreeBonsaiToEmbeddedThreshold

Amount of values, after which index implementation will use an embedded values container (disabled by default).

```
Setting name...: index.sbtreeBonsaiToEmbeddedThreshold
Default value..: -1
Set at run-time: true
```

index.auto.synchronousAutoRebuild

Synchronous execution of auto rebuilding of indexes, in case of a DB crash.

```
Setting name...: index.auto.synchronousAutoRebuild
Default value..: true
Set at run-time: false
```

index.auto.lazyUpdates

Configure the TreeMaps for automatic indexes, as buffered or not. -1 means buffered until tx.commit() or db.close() are called.

```
Setting name...: index.auto.lazyUpdates
Default value..: 10000
Set at run-time: false
```

index.flushAfterCreate

Flush storage buffer after index creation.

```
Setting name...: index.flushAfterCreate
Default value..: true
Set at run-time: false
```

index.manual.lazyUpdates

Configure the TreeMaps for manual indexes as buffered or not. -1 means buffered until tx.commit() or db.close() are called.

```
Setting name...: index.manual.lazyUpdates
Default value..: 1
Set at run-time: false
```

index.durableInNonTxMode

Indicates whether index implementation for plocal storage will be durable in non-Tx mode (true by default).

```
Setting name...: index.durableInNonTxMode
Default value..: true
Set at run-time: false
```

index.ignoreNullValuesDefault

Controls whether null values will be ignored by default by newly created indexes or not (false by default).

```
Setting name...: index.ignoreNullValuesDefault
Default value..: false
Set at run-time: false
```

index.txMode

Indicates the index durability level in TX mode. Can be ROLLBACK_ONLY or FULL (ROLLBACK_ONLY by default).

```
Setting name...: index.txMode
Default value..: FULL
Set at run-time: false
```

index.cursor.prefetchSize

Default prefetch size of index cursor.

```
Setting name...: index.cursor.prefetchSize
Default value..: 500000
Set at run-time: false
```

index.auto.rebuildAfterNotSoftClose

Auto rebuild all automatic indexes after upon database open when wasn't closed properly.

```
Setting name...: index.auto.rebuildAfterNotSoftClose
Default value..: true
Set at run-time: false
```

HashTable

hashTable.slitBucketsBuffer.length

Length of buffer (in pages), where buckets that were split, but not flushed to the disk, are kept. This buffer is used to minimize random IO overhead.

```
Setting name...: hashTable.slitBucketsBuffer.length
Default value..: 1500
Set at run-time: false
```

Sbtree

sbtree.maxDepth

Maximum depth of sbtree, which will be traversed during key look up until it will be treated as broken (64 by default).

```
Setting name...: sbtree.maxDepth
Default value..: 64
Set at run-time: false
```

sbtree.maxKeySize

Maximum size of a key, which can be put in the SBTree in bytes (10240 by default).

```
Setting name...: sbtree.maxKeySize
Default value..: 10240
Set at run-time: false
```

sbtree.maxEmbeddedValueSize

Maximum size of value which can be put in an SBTree without creation link to a standalone page in bytes (40960 by default).

```
Setting name...: sbtree.maxEmbeddedValueSize
Default value..: 40960
Set at run-time: false
```

Sbtreebonsai

sbtreebonsai.bucketSize

Size of bucket in OSBTreeBonsai (in kB). Contract: bucketSize < storagePageSize, storagePageSize % bucketSize == 0.

```
Setting name...: sbtreebonsai.bucketSize
Default value..: 2
Set at run-time: false
```

sbtreebonsai.linkBagCache.size

Amount of LINKBAG collections to be cached, to avoid constant reloading of data.

```
Setting name...: sbtreebonsai.linkBagCache.size
Default value..: 100000
Set at run-time: false
```

sbtreebonsai.linkBagCache.evictionSize

The number of cached LINKBAG collections, which will be removed, when the cache limit is reached.

```
Setting name...: sbtreebonsai.linkBagCache.evictionSize
Default value..: 1000
Set at run-time: false
```

sbtreebonsai.freeSpaceReuseTrigger

How much free space should be in an sbtreebonsai file, before it will be reused during the next allocation.

```
Setting name...: sbtreebonsai.freeSpaceReuseTrigger
Default value..: 0.5
Set at run-time: false
```

RidBag

ridBag.embeddedDefaultSize

Size of embedded RidBag array, when created (empty).

```
Setting name...: ridBag.embeddedDefaultSize
Default value..: 4
Set at run-time: false
```

ridBag.embeddedToSbtreeBonsaiThreshold

Amount of values after which a LINKBAG implementation will use sbtree as values container. Set to -1 to always use an sbtree.

```
Setting name...: ridBag.embeddedToSbtreeBonsaiThreshold
Default value..: 40
Set at run-time: true
```

ridBag.sbtreeBonsaiToEmbeddedToThreshold

Amount of values, after which a LINKBAG implementation will use an embedded values container (disabled by default).

```
Setting name...: ridBag.sbtreeBonsaiToEmbeddedToThreshold
Default value..: -1
Set at run-time: true
```

Collections

collections.preferSBTreeSet

This configuration setting is experimental.

```
Setting name...: collections.preferSBTreeSet
Default value..: false
Set at run-time: false
```

File

file.trackFileClose

Log all the cases when files are closed. This is needed only for internal debugging purposes.

```
Setting name...: file.trackFileClose
Default value..: false
Set at run-time: false
```

file.lock

Locks files when used. Default is true.

```
Setting name...: file.lock
Default value..: true
Set at run-time: false
```

file.deleteDelay

Delay time (in ms) to wait for another attempt to delete a locked file.

```
Setting name...: file.deleteDelay
Default value..: 10
Set at run-time: false
```

file.deleteRetry

Number of retries to delete a locked file.

```
Setting name...: file.deleteRetry
Default value..: 50
Set at run-time: false
```

Security

security.userPasswordSaltIterations

Number of iterations to generate the salt or user password. Changing this setting does not affect stored passwords.

```
Setting name...: security.userPasswordSaltIterations
Default value..: 65536
Set at run-time: false
```

security.userPasswordSaltCacheSize

Cache size of hashed salt passwords. The cache works as LRU. Use 0 to disable the cache.

```
Setting name...: security.userPasswordSaltCacheSize
Default value..: 500
Set at run-time: false
```

security.userPasswordDefaultAlgorithm

Default encryption algorithm used for passwords hashing.

```
Setting name...: security.userPasswordDefaultAlgorithm
Default value..: PBKDF2WithHmacSHA256
Set at run-time: false
```

security.createDefaultUsers

Indicates whether default database users should be created.

```
Setting name...: security.createDefaultUsers
Default value..: true
Set at run-time: false
```

Network

network.maxConcurrentSessions

Maximum number of concurrent sessions.

```
Setting name...: network.maxConcurrentSessions
Default value..: 1000
Set at run-time: true
```

network.socketBufferSize

TCP/IP Socket buffer size.

```
Setting name...: network.socketBufferSize
Default value..: 32768
Set at run-time: true
```

network.lockTimeout

Timeout (in ms) to acquire a lock against a channel.

```
Setting name...: network.lockTimeout
Default value..: 15000
Set at run-time: true
```

network.socketTimeout

TCP/IP Socket timeout (in ms).

```
Setting name...: network.socketTimeout
Default value..: 15000
Set at run-time: true
```

network.requestTimeout

Request completion timeout (in ms).

```
Setting name...: network.requestTimeout
Default value..: 3600000
Set at run-time: true
```

network.retry

Number of attempts to connect to the server on failure.

```
Setting name...: network.retry
Default value..: 5
Set at run-time: true
```

network.retryDelay

The time (in ms) the client must wait, before reconnecting to the server on failure.

```
Setting name...: network.retryDelay
Default value..: 500
Set at run-time: true
```

network.binary.loadBalancing.enabled

Asks for DNS TXT record, to determine if load balancing is supported.

```
Setting name...: network.binary.loadBalancing.enabled
Default value..: false
Set at run-time: true
```

network.binary.loadBalancing.timeout

Maximum time (in ms) to wait for the answer from DNS about the TXT record for load balancing.

```
Setting name...: network.binary.loadBalancing.timeout
Default value..: 2000
Set at run-time: true
```

network.binary.maxLength

TCP/IP max content length (in KB) of BINARY requests.

```
Setting name...: network.binary.maxLength
Default value..: 16384
Set at run-time: true
```

network.binary.readResponse.maxTimes

Maximum attempts, until a response can be read. Otherwise, the response will be dropped from the channel.

```
Setting name...: network.binary.readResponse.maxTimes
Default value..: 20
Set at run-time: true
```

network.binary.debug

Debug mode: print all data incoming on the binary channel.

```
Setting name...: network.binary.debug
Default value..: false
Set at run-time: true
```

network.http.installDefaultCommands

Installs the default HTTP commands.

```
Setting name...: network.http.installDefaultCommands
Default value..: true
Set at run-time: true
```

network.http.serverInfo

Server info to send in HTTP responses. Change the default if you want to hide it is a OrientDB Server.

```
Setting name...: network.http.serverInfo
Default value..: OrientDB Server v.2.2.12-SNAPSHOT
Set at run-time: true
```

network.http.maxLength

TCP/IP max content length (in bytes) for HTTP requests.

```
Setting name...: network.http.maxLength
Default value..: 1000000
Set at run-time: true
```

network.http.streaming

Enable Http chunked streaming for json responses.

```
Setting name...: network.http.streaming
Default value..: true
Set at run-time: false
```

network.http.charset

Http response charset.

```
Setting name...: network.http.charset
Default value..: utf-8
Set at run-time: true
```

network.http.jsonResponseError

Http response error in json.

```
Setting name...: network.http.jsonResponseError
Default value..: true
Set at run-time: true
```

network.http.jsonp

Enable the usage of JSONP, if requested by the client. The parameter name to use is 'callback'.

```
Setting name...: network.http.jsonp
Default value..: false
Set at run-time: true
```

network.http.sessionExpireTimeout

Timeout, after which an http session is considered to have expired (in seconds).

```
Setting name...: network.http.sessionExpireTimeout
Default value..: 300
Set at run-time: false
```

network.http.useToken

Enable Token based sessions for http.

```
Setting name...: network.http.useToken
Default value..: false
Set at run-time: false
```

network.token.secretKey

Network token sercret key.

```
Setting name...: network.token.secretKey
Default value..:
Set at run-time: false
```

network.token.encryptionAlgorithm

Network token algorithm.

```
Setting name...: network.token.encryptionAlgorithm
Default value..: HmacSHA256
Set at run-time: false
```

network.token.expireTimeout

Timeout, after which a binary session is considered to have expired (in minutes).

```
Setting name...: network.token.expireTimeout
Default value..: 60
Set at run-time: false
```

Profiler

profiler.enabled

Enables the recording of statistics and counters.

```
Setting name...: profiler.enabled
Default value..: false
Set at run-time: true
```

profiler.config

Configures the profiler as ,,.

```
Setting name...: profiler.config
Default value..: null
Set at run-time: true
```

profiler.autoDump.interval

Dumps the profiler values at regular intervals (in seconds).

```
Setting name...: profiler.autoDump.interval
Default value..: 0
Set at run-time: true
```

profiler.maxValues

Maximum values to store. Values are managed in a LRU.

```
Setting name...: profiler.maxValues
Default value..: 200
Set at run-time: false
```

Sequence

sequence.maxRetry

Maximum number of retries between attempt to change a sequence in concurrent mode.

```
Setting name...: sequence.maxRetry
Default value..: 100
Set at run-time: false
```

sequence.retryDelay

Maximum number of ms to wait between concurrent modification exceptions. The value is computed as random between 1 and this number.

```
Setting name...: sequence.retryDelay
Default value..: 200
Set at run-time: false
```

StorageProfiler

storageProfiler.intervalBetweenSnapshots

Interval between snapshots of profiler state in milliseconds.

```
Setting name...: storageProfiler.intervalBetweenSnapshots
Default value..: 100
Set at run-time: false
```

storageProfiler.cleanUpInterval

Interval between time series in milliseconds.

```
Setting name...: storageProfiler.cleanUpInterval
Default value..: 5000
Set at run-time: false
```

Log

log.console.level

Console logging level.

```
Setting name...: log.console.level
Default value..: info
Set at run-time: true
```

log.file.level

File logging level.

```
Setting name...: log.file.level
Default value..: info
Set at run-time: true
```

log.console.ansi

ANSI Console support. 'auto' means automatic check if it is supported, 'true' to force using ANSI, 'false' to avoid using ANSI.

```
Setting name...: log.console.ansi
Default value..: auto
Set at run-time: false
```

Class

class.minimumClusters

Minimum clusters to create when a new class is created. 0 means Automatic.

```
Setting name...: class.minimumClusters
Default value..: 0
Set at run-time: false
```

Cache

cache.local.impl

Local Record cache implementation.

```
Setting name...: cache.local.impl
Default value..: com.orientechnologies.orient.core.cache.ORecordCacheWeakRefs
Set at run-time: false
```

cache.local.enabled

Deprecated, Level1 cache cannot be disabled anymore.

```
Setting name...: cache.local.enabled
Default value..: true
Set at run-time: false
```

Command

command.timeout

Default timeout for commands (in ms).

```
Setting name...: command.timeout
Default value..: 0
Set at run-time: true
```

command.cache.enabled

Enable command cache.

```
Setting name...: command.cache.enabled
Default value..: false
Set at run-time: false
```

command.cache.evictStrategy

Command cache strategy between: [INVALIDATE_ALL,PER_CLUSTER].

```
Setting name...: command.cache.evictStrategy
Default value..: PER_CLUSTER
Set at run-time: false
```

command.cache.minExecutionTime

Minimum execution time to consider caching the result set.

```
Setting name...: command.cache.minExecutionTime
Default value..: 10
Set at run-time: false
```

command.cache.maxResultsetSize

Maximum resultset time to consider caching result set.

```
Setting name...: command.cache.maxResultsetSize
Default value..: 500
Set at run-time: false
```

Query

query.parallelAuto

Auto enable parallel query, if requirements are met.

```
Setting name...: query.parallelAuto
Default value..: false
Set at run-time: false
```

query.parallelMinimumRecords

Minimum number of records to activate parallel query automatically.

```
Setting name...: query.parallelMinimumRecords
Default value..: 300000
Set at run-time: false
```

query.parallelResultQueueSize

Size of the queue that holds results on parallel execution. The queue is blocking, so in case the queue is full, the query threads will be in a wait state.

```
Setting name...: query.parallelResultQueueSize
Default value..: 20000
Set at run-time: false
```

query.scanPrefetchPages

Pages to prefetch during scan. Setting this value higher makes scans faster, because it reduces the number of I/O operations, though it consumes more memory. (Use 0 to disable).

```
Setting name...: query.scanPrefetchPages
Default value..: 20
Set at run-time: false
```

query.scanBatchSize

Scan clusters in blocks of records. This setting reduces the lock time on the cluster during scans. A high value mean a faster execution, but also a lower concurrency level. Set to 0 to disable batch scanning. Disabling batch scanning is suggested for read-only databases only.

```
Setting name...: query.scanBatchSize
Default value..: 1000
Set at run-time: false
```

query.scanThresholdTip

If the total number of records scanned in a query exceeds this setting, then a warning is given. (Use 0 to disable).

```
Setting name...: query.scanThresholdTip
Default value..: 50000
Set at run-time: false
```

query.limitThresholdTip

If the total number of returned records exceeds this value, then a warning is given. (Use 0 to disable).

```
Setting name...: query.limitThresholdTip
Default value..: 10000
Set at run-time: false
```

query.live.support

Enable/Disable the support of live query. (Use false to disable).

```
Setting name...: query.live.support
Default value..: true
Set at run-time: false
```

Statement

statement.cacheSize

Number of parsed SQL statements kept in cache.

```
Setting name...: statement.cacheSize
Default value..: 100
Set at run-time: false
```

Sql

sql.graphConsistencyMode

Consistency mode for graphs. It can be 'tx' (default), 'notx_sync_repair' and 'notx_async_repair'. 'tx' uses transactions to maintain consistency. Instead both 'notx_sync_repair' and 'notx_async_repair' do not use transactions, and the consistency, in case of JVM crash, is guaranteed by a database repair operation that run at startup. With 'notx_sync_repair' the repair is synchronous, so the database comes online after the repair is ended, while with 'notx_async_repair' the repair is a background process.

```
Setting name...: sql.graphConsistencyMode
Default value..: tx
Set at run-time: false
```

Client

client.channel.maxPool

Maximum size of pool of network channels between client and server. A channel is a TCP/IP connection.

```
Setting name...: client.channel.maxPool
Default value..: 100
Set at run-time: false
```

client.connectionPool.waitTimeout

Maximum time, where the client should wait for a connection from the pool, when all connections busy.

```
Setting name...: client.connectionPool.waitTimeout
Default value..: 5000
Set at run-time: true
```

client.channel.dbReleaseWaitTimeout

Delay (in ms), after which a data modification command will be resent, if the DB was frozen.

```
Setting name...: client.channel.dbReleaseWaitTimeout
Default value..: 10000
Set at run-time: true
```

client.ssl.enabled

Use SSL for client connections.

```
Setting name...: client.ssl.enabled
Default value..: false
Set at run-time: false
```

client.ssl.keyStore

Use SSL for client connections.

```
Setting name...: client.ssl.keyStore
Default value..: null
Set at run-time: false
```

client.ssl.keyStorePass

Use SSL for client connections.

```
Setting name...: client.ssl.keyStorePass
Default value..: null
Set at run-time: false
```

client.ssl.trustStore

Use SSL for client connections.

```
Setting name...: client.ssl.trustStore
Default value..: null
Set at run-time: false
```

client.ssl.trustStorePass

Use SSL for client connections.

```
Setting name...: client.ssl.trustStorePass
Default value..: null
Set at run-time: false
```

client.krb5.config

Location of the Kerberos configuration file.

```
Setting name...: client.krb5.config
Default value..: null
Set at run-time: false
```

client.krb5.ccname

Location of the Kerberos client ticketcache.

```
Setting name...: client.krb5.ccname
Default value..: null
Set at run-time: false
```

client.krb5.ktname

Location of the Kerberos client keytab.

```
Setting name...: client.krb5.ktname
Default value..: null
Set at run-time: false
```

client.credentialinterceptor

The name of the CredentialInterceptor class.

```
Setting name...: client.credentialinterceptor
Default value..: null
Set at run-time: false
```

client.session.tokenBased

Request a token based session to the server.

```
Setting name...: client.session.tokenBased
Default value..: true
Set at run-time: false
```

client.channel.minPool

Minimum pool size.

```
Setting name...: client.channel.minPool
Default value..: 1
Set at run-time: false
```

Server

server.openAllDatabasesAtStartup

If true, the server opens all the available databases at startup. Available since 2.2.

```
Setting name...: server.openAllDatabasesAtStartup
Default value..: false
Set at run-time: false
```

server.channel.cleanDelay

Time in ms of delay to check pending closed connections.

```
Setting name...: server.channel.cleanDelay
Default value..: 5000
Set at run-time: false
```

server.cache.staticFile

Cache static resources upon loading.

```
Setting name...: server.cache.staticFile
Default value..: false
Set at run-time: false
```

server.log.dumpClientExceptionLevel

Logs client exceptions. Use any level supported by Java java.util.logging.Level class: OFF, FINE, CONFIG, INFO, WARNING, SEVERE.

```
Setting name...: server.log.dumpClientExceptionLevel
Default value..: FINE
Set at run-time: false
```

server.log.dumpClientExceptionFullStackTrace

Dumps the full stack trace of the exception sent to the client.

```
Setting name...: server.log.dumpClientExceptionFullStackTrace
Default value..: false
Set at run-time: true
```

server.security.file

Location of the OrientDB security.json configuration file.

```
Setting name...: server.security.file
Default value..: null
Set at run-time: false
```

Distributed

distributed.crudTaskTimeout

Maximum timeout (in ms) to wait for CRUD remote tasks.

```
Setting name...: distributed.crudTaskTimeout
Default value..: 3000
Set at run-time: true
```

distributed.commandTaskTimeout

Maximum timeout (in ms) to wait for command distributed tasks.

```
Setting name...: distributed.commandTaskTimeout
Default value..: 120000
Set at run-time: true
```

distributed.commandQuickTaskTimeout

Maximum timeout (in ms) to wait for quick command distributed tasks.

```
Setting name...: distributed.commandQuickTaskTimeout
Default value..: 5000
Set at run-time: true
```

distributed.commandLongTaskTimeout

Maximum timeout (in ms) to wait for Long-running distributed tasks.

```
Setting name...: distributed.commandLongTaskTimeout
Default value..: 86400000
Set at run-time: true
```

distributed.deployDbTaskTimeout

Maximum timeout (in ms) to wait for database deployment.

```
Setting name...: distributed.deployDbTaskTimeout
Default value..: 1200000
Set at run-time: true
```

distributed.deployChunkTaskTimeout

Maximum timeout (in ms) to wait for database chunk deployment.

```
Setting name...: distributed.deployChunkTaskTimeout
Default value..: 15000
Set at run-time: true
```

distributed.deployDbTaskCompression

Compression level (between 0 and 9) to use in backup for database deployment.

```
Setting name...: distributed.deployDbTaskCompression
Default value..: 7
Set at run-time: true
```

distributed.asynchQueueSize

Queue size to handle distributed asynchronous operations. The bigger is the queue, the more operation are buffered, but also more memory it's consumed. 0 = dynamic allocation, which means up to 2^31-1 entries.

```
Setting name...: distributed.asynchQueueSize
Default value..: 0
Set at run-time: false
```

distributed.asynchResponsesTimeout

Maximum timeout (in ms) to collect all the asynchronous responses from replication. After this time the operation is rolled back (through an UNDO).

```
Setting name...: distributed.asynchResponsesTimeout
Default value..: 15000
Set at run-time: false
```

distributed.purgeResponsesTimerDelay

Maximum timeout (in ms) to collect all the asynchronous responses from replication. This is the delay the purge thread uses to check asynchronous requests in timeout.

```
Setting name...: distributed.purgeResponsesTimerDelay
Default value..: 15000
Set at run-time: false
```

distributed.conflictResolverRepairerChain

Chain of conflict resolver implementation to use.

```
Setting name...: distributed.conflictResolverRepairerChain
Default value..: majority,content,version
Set at run-time: false
```

distributed.conflictResolverRepairerCheckEvery

Time (in ms) when the conflict resolver auto-repairer checks for records/cluster to repair.

```
Setting name...: distributed.conflictResolverRepairerCheckEvery
Default value..: 5000
Set at run-time: true
```

distributed.conflictResolverRepairerBatch

Number of record to repair in batch.

```
Setting name...: distributed.conflictResolverRepairerBatch
Default value..: 100
Set at run-time: true
```

distributed.txAliveTimeout

Maximum timeout (in ms) a distributed transaction can be alive. This timeout is to rollback pending transactions after a while.

```
Setting name...: distributed.txAliveTimeout
Default value..: 30000
Set at run-time: true
```

distributed.requestChannels

Number of network channels used to send requests.

```
Setting name...: distributed.requestChannels
Default value..: 1
Set at run-time: false
```

distributed.responseChannels

Number of network channels used to send responses.

```
Setting name...: distributed.responseChannels
Default value..: 1
Set at run-time: false
```

distributed.heartbeatTimeout

Maximum time in ms to wait for the heartbeat. If the server does not respond in time, it is put offline.

```
Setting name...: distributed.heartbeatTimeout
Default value..: 10000
Set at run-time: false
```

distributed.checkHealthCanOfflineServer

In case a server does not respond to the heartbeat message, it is set offline.

```
Setting name...: distributed.checkHealthCanOfflineServer
Default value..: false
Set at run-time: false
```

distributed.checkHealthEvery

Time in ms to check the cluster health. Set to 0 to disable it.

```
Setting name...: distributed.checkHealthEvery
Default value..: 10000
Set at run-time: false
```

distributed.autoRemoveOfflineServers

This is the amount of time (in ms) the server has to be OFFLINE, before it is automatically removed from the distributed configuration. -1 = never, 0 = immediately, >0 the actual time to wait.

```
Setting name...: distributed.autoRemoveOfflineServers
Default value..: -1
Set at run-time: true
```

distributed.publishNodeStatusEvery

Time in ms to publish the node status on distributed map. Set to 0 to disable such refresh of node configuration.

```
Setting name...: distributed.publishNodeStatusEvery
Default value..: 5000
Set at run-time: true
```

distributed.localQueueSize

Size of the intra-thread queue for distributed messages.

```
Setting name...: distributed.localQueueSize
Default value..: 10000
Set at run-time: false
```

distributed.dbWorkerThreads

Number of parallel worker threads per database that process distributed messages.

```
Setting name...: distributed.dbWorkerThreads
Default value..: 8
Set at run-time: false
```

distributed.queueMaxSize

Maximum queue size to mark a node as stalled. If the number of messages in queue are more than this values, the node is restarted with a remote command (0 = no maximum, which means up to 2^31-1 entries).

```
Setting name...: distributed.queueMaxSize
Default value..: 10000
Set at run-time: false
```

distributed.backupDirectory

Directory where the copy of an existent database is saved, before it is downloaded from the cluster. Leave it empty to avoid the backup..

```
Setting name...: distributed.backupDirectory
Default value..: ../backup/databases
Set at run-time: false
```

distributed.concurrentTxMaxAutoRetry

Maximum attempts the transaction coordinator should execute a transaction automatically, if records are locked. (Minimum is 1 = no attempts).

```
Setting name...: distributed.concurrentTxMaxAutoRetry
Default value..: 10
Set at run-time: true
```

distributed.atomicLockTimeout

Timeout (in ms) to acquire a distributed lock on a record. (0=infinite).

```
Setting name...: distributed.atomicLockTimeout
Default value..: 300
Set at run-time: true
```

distributed.concurrentTxAutoRetryDelay

Delay (in ms) between attempts on executing a distributed transaction, which had failed because of locked records. (0=no delay).

```
Setting name...: distributed.concurrentTxAutoRetryDelay
Default value..: 100
Set at run-time: true
```

distributed.queueTimeout

Maximum timeout (in ms) to wait for the response in replication.

```
Setting name...: distributed.queueTimeout
Default value..: 500000
Set at run-time: true
```

Jna

jna.disable.system.library

This property disables using JNA, should it be installed on your system. (Default true) To use JNA bundled with database.

```
Setting name...: jna.disable.system.library
Default value..: true
Set at run-time: false
```

Oauth2

oauth2.secretkey

Http OAuth2 secret key.

```
Setting name...: oauth2.secretkey
Default value..:
Set at run-time: false
```

Lazyset

lazyset.workOnStream

Deprecated, now BINARY serialization is used in place of CSV.

```
Setting name...: lazyset.workOnStream
Default value..: true
Set at run-time: false
```

Mvrbtree

mvrbtree.timeout

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.timeout
Default value..: 0
Set at run-time: false
```

mvrbtree.nodePageSize

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.nodePageSize
Default value..: 256
Set at run-time: false
```

mvrbtree.loadFactor

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.loadFactor
Default value..: 0.7
Set at run-time: false
```

mvrbtree.optimizeThreshold

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.optimizeThreshold
Default value..: 100000
Set at run-time: false
```

mvrbtree.entryPoints

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.entryPoints
Default value..: 64
Set at run-time: false
```

mvrbtree.optimizeEntryPointsFactor

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.optimizeEntryPointsFactor
Default value..: 1.0
Set at run-time: false
```

mvrbtree.entryKeysInMemory

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.entryKeysInMemory
Default value..: false
Set at run-time: false
```

mvrbtree.entryValuesInMemory

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.entryValuesInMemory
Default value..: false
Set at run-time: false
```

mvrbtree.ridBinaryThreshold

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.ridBinaryThreshold
Default value..: -1
Set at run-time: false
```

mvrbtree.ridNodePageSize

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.ridNodePageSize
Default value..: 64
Set at run-time: false
```

mvrbtree.ridNodeSaveMemory

Deprecated, MVRBTREE IS NOT USED ANYMORE IN FAVOR OF SBTREE AND HASHINDEX.

```
Setting name...: mvrbtree.ridNodeSaveMemory
Default value..: false
Set at run-time: false
```

NOTE: On 64-bit systems you have not the limitation of 32-bit systems with memory.

Logging

Logging is configured in a separate file, look at Logging for more information.

Storage configuration

OrientDB allows modifications to the storage configuration. Even though this will be supported with high level commands, for now it's pretty "internal" using Java API.

To get the storage configuration for the current database:

```
OStorageConfiguration cfg = db.getStorage().getConfiguration();
```

Look at `OStorageConfiguration` to discover all the properties you can change. To change the configuration of a cluster get it by ID;

```
OStoragePhysicalClusterConfigurationLocal clusterCfg = (OStoragePhysicalClusterConfigurationLocal) cfg.clusters.get(3);
```

To change the default settings for new clusters get the file template object. In this example we change the initial file size from the default 500Kb down to 10Kb:

```
OStorageSegmentConfiguration defaultCfg = (OStorageSegmentConfiguration) cfg.fileTemplate;
defaultCfg.fileStartSize = "10Kb";
```

After changes call `OStorageConfiguration.update()` :

```
cfg.update();
```

Tuning the Graph API

This guide is specific for the TinkerPop Blueprints Graph Database. Please be sure to read the generic guide to the Performance-Tuning.

Connect to the database locally

Local connection is much faster than remote. So use "plocal" based on the storage engine used on database creation. If you need to connect to the database from the network you can use the "Embed the server technique".

Avoid putting properties on edges

Even though supports properties on edges, this is much expensive because it creates a new record per edge. So if you need them you've to know that the database will be bigger and insertion time will be much longer.

Set properties all together

It's much lighter to set properties in block than one by one. Look at this paragraph: Setting Multiple Properties.

Set properties on vertex and edge creation

It's even faster if you set properties directly on creation of vertices and edges. Look at this paragraph: Creating Elements and Properties.

Massive Insertion

See Generic improvement on massive insertion. To access to the underlying database use:

```
database.getRawGraph().declareIntent( new OIntentMassiveInsert() );

// YOUR MASSIVE INSERTION

database.getRawGraph().declareIntent( null );
```

Avoid transactions if you can

Use the OrientGraphNoTx implementation that doesn't use transaction for basic operations like creation and deletion of vertices and edges. If you plan to don't use transactions change the consistency level. OrientGraphNoTx is not compatible with OrientBatchGraph so use it plain:

```
OrientGraphNoTx graph = new OrientGraphNoTx("local:/tmp/mydb");
```

Use the schema

Even if you can model your graph with only the entities (V)ertex and (E)dge it's much better to use schema for your types extending Vertex and Edge classes. In this way traversing will be faster and vertices and edges will be split on different files. For more information look at: Graph Schema.

Example:

```
OClass account = graph.createVertexType("Account");
Vertex v = graph.addVertex("class:Account");
```

Use indexes to lookup vertices by an ID

If you've your own ID on vertices and you need to lookup them to create edges then create an index against it:

```
graph.createKeyIndex("id", Vertex.class, new Parameter("class", "Account"));
```

If the ID is unique then create an UNIQUE index that is much faster and lighter:

```
graph.createKeyIndex("id", Vertex.class, new Parameter("type", "UNIQUE"), new Parameter("class", "Account"));
```

To lookup vertices by ID:

```
for( Vertex v : graph.getVertices("Account.id", "23876JS2") ) {
  System.out.println("Found vertex: " + v );
}
```

Disable validation

Every time a graph element is modified, OrientDB executes a validation to assure the graph rules are all respected, that means:

- put edge in out/in collections
- put vertex in edges in/out

Now if you use the Graph API without bypassing graph element manipulation this could be turned off with a huge gain in performance:

```
graph.setValidationEnabled(false);
```

Reduce vertex objects

You can avoid the creation of a new ODocument for each new vertex by reusing it with ODocument.reset() method that clears the instance making it ready for a new insert operation. Bear in mind that you will need to assign the document with the proper class after resetting as it is done in the code below.

NOTE: This trick works ONLY IN NON-TRANSACTIONAL contexts, because during transactions the documents could be kept in memory until commit.

Example:

```
db.declareIntent( new OIntentMassiveInsert() );

ODocument doc = db.createVertex("myVertex");
for( int i = 0; i < 1000000; ++i ){
  doc.reset();
  doc.setClassName("myVertex");
  doc.field("id", i);
  doc.field("name", "Jason");
  doc.save();
}

db.declareIntent( null );
```

Cache management

Graph Database, by default, caches the most used elements. For massive insertion is strongly suggested to disable cache to avoid to keep all the element in memory. Massive Insert Intent automatically sets it to false.

```
graph.setRetainObjects(false);
```

Tuning the Document API

This guide is specific for the Document Database. Please be sure to read the generic guide to the Performance-Tuning.

Massive Insertion

See Generic improvement on massive insertion.

Avoid document creation

You can avoid the creation of a new ODocument for each insertion by using the ODocument.reset() method that clears the instance making it ready for a new insert operation. Bear in mind that you will need to assign the document with the proper class after resetting as it is done in the code below.

NOTE: This trick works ONLY IN NON-TRANSACTIONAL contexts, because during transactions the documents could be kept in memory until commit.

Example:

```
import com.orientechnologies.orient.core.intent.OIntentMassiveInsert;

db.declareIntent( new OIntentMassiveInsert() );

ODocument doc = new ODocument();
for( int i = 0; i < 1000000; ++i ){
  doc.reset();
  doc.setClassName("Customer");
  doc.field("id", i);
  doc.field("name", "Jason");
  doc.save();
}

db.declareIntent( null );
```

Tuning the Object API

This guide is specific for the Object Database. Please be sure to read the generic guide to the Performance-Tuning.

Massive Insertion

See Generic improvement on massive insertion.

Profiler

OrientDB Enterprise Edition comes with a profiler that collects all the metrics about the engine and the system where is running.

Automatic dump

When you incur in problems, the best way to produce information about OrientDB is activating a regular dump of the profiler. Set this configuration variable at start:

```
java ... -Dprofiler.autoDump.reset=true -Dprofiler.autoDump.interval=60 -Dprofiler.enabled=true ...
```

This will dump the profiler in the console every 60 seconds and resets the metrics after the dump. For more information about settings look at Parameters.

Retrieve profiler metrics via HTTP

```
http://<server>[<:port>]/profiler/<command>/[<config>]|[<from>/<to>]
```

Where:

- **server** is the server where OrientDB is running
- **port** is the http port, OrientDB listens at 2480 by default
- **command**, is the command between:
  - **realtime** to retrieve realtime information
  - **last** to retrieve realtime information
  - **archive** to retrieve archived profiling
  - **summary** to retrieve summary of past profiling
  - **start** to start profiling
  - **stop** to stop profiling
  - **reset** to reset the profiler (equals to stop+start)
  - **status** to know the status of profiler
  - **configure** to configure profiling
  - **metadata** to retrieve metadata

Example:

```
http://localhost:2480/profiler/realtime
```

Metric type

Chrono

Chrono are recording of operation. Each Chrono has the following values:

- **last**, as the last time recorded
- **min**, as the minimum time recorded
- **max**, as the maximum time recorded
- **average**, as the average time recorded
- **total**, as the total time recorded
- **entries**, as the number of times the metric has been recorded

Counter

It's a counter as long value that records resources.

HookValues

Are generic values of any type between the supported ones: string, number, boolean or null.

A hook value is not collected in central way, but it's gathered at runtime by calling the hooks as callbacks.

Metric main categories

Follows the main categories of metrics:

- `db.<db-name>` : database related metrics
- `db.<db-name>.cache` : metrics about db's caching
- `db.<db-name>.index` : metrics about db's indexes
- `system` : system metrics like CPU, memory, OS, etc.
- `system.disk` : File system metrics
- `process` : not strictly related to database but to the process (JVM) that is running OrientDB as client, server or embedded
- `process.network` : network metrics
- `process.runtime` : process's runtime information like memory used, etc
- `server` : server related metrics

Example of profiler values extracted from the server after test suite is run (http://localhost:2480/profiler/realtime):

```
{
    "realtime": {
        "from": 1344531312356,
        "to": 9223372036854776000,
        "hookValues": {
            "db.0$db.cache.level1.current": 0,
            "db.0$db.cache.level1.enabled": false,
            "db.0$db.cache.level1.max": -1,
            "db.0$db.cache.level2.current": 0,
            "db.0$db.cache.level2.enabled": true,
            "db.0$db.cache.level2.max": -1,
            "db.0$db.data.holeSize": 0,
            "db.0$db.data.holes": 0,
            "db.0$db.index.dictionary.entryPointSize": 64,
            "db.0$db.index.dictionary.items": 0,
            "db.0$db.index.dictionary.maxUpdateBeforeSave": 5000,
            "db.0$db.index.dictionary.optimizationThreshold": 100000,
            "db.1$db.cache.level1.current": 0,
            "db.1$db.cache.level1.enabled": false,
            "db.1$db.cache.level1.max": -1,
            "db.1$db.cache.level2.current": 0,
            "db.1$db.cache.level2.enabled": true,
            "db.1$db.cache.level2.max": -1,
            "db.1$db.data.holeSize": 0,
            "db.1$db.data.holes": 0,
            "db.1$db.index.dictionary.entryPointSize": 64,
            "db.1$db.index.dictionary.items": 0,
            "db.1$db.index.dictionary.maxUpdateBeforeSave": 5000,
            "db.1$db.index.dictionary.optimizationThreshold": 100000,
            "db.2$db.cache.level1.current": 0,
            "db.2$db.cache.level1.enabled": false,
            "db.2$db.cache.level1.max": -1,
            "db.2$db.cache.level2.current": 0,
            "db.2$db.cache.level2.enabled": true,
            "db.2$db.cache.level2.max": -1,
            "db.2$db.data.holeSize": 0,
            "db.2$db.data.holes": 0,
            "db.2$db.index.dictionary.entryPointSize": 64,
            "db.2$db.index.dictionary.items": 0,
            "db.2$db.index.dictionary.maxUpdateBeforeSave": 5000,
            "db.2$db.index.dictionary.optimizationThreshold": 100000,
            "db.demo.cache.level1.current": 0,
            "db.demo.cache.level1.enabled": false,
            "db.demo.cache.level1.max": -1,
            "db.demo.cache.level2.current": 20520,
            "db.demo.cache.level2.enabled": true,
            "db.demo.cache.level2.max": -1,
            "db.demo.data.holeSize": 47553,
            "db.demo.data.holes": 24,
            "db.demo.index.BaseTestClass.testParentProperty.entryPointSize": 64,
            "db.demo.index.BaseTestClass.testParentProperty.items": 2,
            "db.demo.index.BaseTestClass.testParentProperty.maxUpdateBeforeSave": 5000,
            "db.demo.index.BaseTestClass.testParentProperty.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedList.entryPointSize": 64,
            "db.demo.index.ClassIndexTestCompositeEmbeddedList.items": 0,
            "db.demo.index.ClassIndexTestCompositeEmbeddedList.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedList.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMap.entryPointSize": 64,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMap.items": 0,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMap.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMap.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMapByKey.entryPointSize": 64,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMapByKey.items": 0,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMapByKey.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMapByKey.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMapByValue.entryPointSize": 64,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMapByValue.items": 0,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMapByValue.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedMapByValue.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedSet.entryPointSize": 64,
            "db.demo.index.ClassIndexTestCompositeEmbeddedSet.items": 0,
            "db.demo.index.ClassIndexTestCompositeEmbeddedSet.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestCompositeEmbeddedSet.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestCompositeLinkList.entryPointSize": 64,
            "db.demo.index.ClassIndexTestCompositeLinkList.items": 0,
            "db.demo.index.ClassIndexTestCompositeLinkList.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestCompositeLinkList.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestCompositeLinkMapByValue.entryPointSize": 64,
            "db.demo.index.ClassIndexTestCompositeLinkMapByValue.items": 0,
            "db.demo.index.ClassIndexTestCompositeLinkMapByValue.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestCompositeLinkMapByValue.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestCompositeOne.entryPointSize": 64,
            "db.demo.index.ClassIndexTestCompositeOne.items": 0,
            "db.demo.index.ClassIndexTestCompositeOne.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestCompositeOne.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestCompositeTwo.entryPointSize": 64,
            "db.demo.index.ClassIndexTestCompositeTwo.items": 0,
            "db.demo.index.ClassIndexTestCompositeTwo.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestCompositeTwo.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestDictionaryIndex.entryPointSize": 64,
            "db.demo.index.ClassIndexTestDictionaryIndex.items": 0,
            "db.demo.index.ClassIndexTestDictionaryIndex.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestDictionaryIndex.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestFulltextIndex.entryPointSize": 64,
            "db.demo.index.ClassIndexTestFulltextIndex.items": 0,
            "db.demo.index.ClassIndexTestFulltextIndex.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestFulltextIndex.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestNotUniqueIndex.entryPointSize": 64,
            "db.demo.index.ClassIndexTestNotUniqueIndex.items": 0,
            "db.demo.index.ClassIndexTestNotUniqueIndex.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestNotUniqueIndex.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestParentPropertyNine.entryPointSize": 64,
            "db.demo.index.ClassIndexTestParentPropertyNine.items": 0,
            "db.demo.index.ClassIndexTestParentPropertyNine.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestParentPropertyNine.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestPropertyByKeyEmbeddedMap.entryPointSize": 64,
            "db.demo.index.ClassIndexTestPropertyByKeyEmbeddedMap.items": 0,
            "db.demo.index.ClassIndexTestPropertyByKeyEmbeddedMap.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestPropertyByKeyEmbeddedMap.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestPropertyByValueEmbeddedMap.entryPointSize": 64,
            "db.demo.index.ClassIndexTestPropertyByValueEmbeddedMap.items": 0,
            "db.demo.index.ClassIndexTestPropertyByValueEmbeddedMap.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestPropertyByValueEmbeddedMap.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestPropertyEmbeddedMap.entryPointSize": 64,
            "db.demo.index.ClassIndexTestPropertyEmbeddedMap.items": 0,
            "db.demo.index.ClassIndexTestPropertyEmbeddedMap.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestPropertyEmbeddedMap.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestPropertyLinkedMap.entryPointSize": 64,
            "db.demo.index.ClassIndexTestPropertyLinkedMap.items": 0,
            "db.demo.index.ClassIndexTestPropertyLinkedMap.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestPropertyLinkedMap.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestPropertyLinkedMapByKey.entryPointSize": 64,
            "db.demo.index.ClassIndexTestPropertyLinkedMapByKey.items": 0,
            "db.demo.index.ClassIndexTestPropertyLinkedMapByKey.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestPropertyLinkedMapByKey.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestPropertyLinkedMapByValue.entryPointSize": 64,
            "db.demo.index.ClassIndexTestPropertyLinkedMapByValue.items": 0,
            "db.demo.index.ClassIndexTestPropertyLinkedMapByValue.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestPropertyLinkedMapByValue.optimizationThreshold": 100000,
            "db.demo.index.ClassIndexTestPropertyOne.entryPointSize": 64,
            "db.demo.index.ClassIndexTestPropertyOne.items": 0,
            "db.demo.index.ClassIndexTestPropertyOne.maxUpdateBeforeSave": 5000,
            "db.demo.index.ClassIndexTestPropertyOne.optimizationThreshold": 100000,
            "db.demo.index.Collector.stringCollection.entryPointSize": 64,
            "db.demo.index.Collector.stringCollection.items": 0,
            "db.demo.index.Collector.stringCollection.maxUpdateBeforeSave": 5000,
            "db.demo.index.Collector.stringCollection.optimizationThreshold": 100000,
            "db.demo.index.DropPropertyIndexCompositeIndex.entryPointSize": 64,
            "db.demo.index.DropPropertyIndexCompositeIndex.items": 0,
            "db.demo.index.DropPropertyIndexCompositeIndex.maxUpdateBeforeSave": 5000,
            "db.demo.index.DropPropertyIndexCompositeIndex.optimizationThreshold": 100000,
            "db.demo.index.Fruit.color.entryPointSize": 64,
            "db.demo.index.Fruit.color.items": 0,
            "db.demo.index.Fruit.color.maxUpdateBeforeSave": 5000,
            "db.demo.index.Fruit.color.optimizationThreshold": 100000,
            "db.demo.index.IndexCountPlusCondition.entryPointSize": 64,
            "db.demo.index.IndexCountPlusCondition.items": 5,
            "db.demo.index.IndexCountPlusCondition.maxUpdateBeforeSave": 5000,
            "db.demo.index.IndexCountPlusCondition.optimizationThreshold": 100000,
            "db.demo.index.IndexNotUniqueIndexKeySize.entryPointSize": 64,
            "db.demo.index.IndexNotUniqueIndexKeySize.items": 5,
            "db.demo.index.IndexNotUniqueIndexKeySize.maxUpdateBeforeSave": 5000,
            "db.demo.index.IndexNotUniqueIndexKeySize.optimizationThreshold": 100000,
            "db.demo.index.IndexNotUniqueIndexSize.entryPointSize": 64,
            "db.demo.index.IndexNotUniqueIndexSize.items": 5,
            "db.demo.index.IndexNotUniqueIndexSize.maxUpdateBeforeSave": 5000,
            "db.demo.index.IndexNotUniqueIndexSize.optimizationThreshold": 100000,
            "db.demo.index.MapPoint.x.entryPointSize": 64,
            "db.demo.index.MapPoint.x.items": 9999,
            "db.demo.index.MapPoint.x.maxUpdateBeforeSave": 5000,
            "db.demo.index.MapPoint.x.optimizationThreshold": 100000,
            "db.demo.index.MapPoint.y.entryPointSize": 64,
            "db.demo.index.MapPoint.y.items": 10000,
            "db.demo.index.MapPoint.y.maxUpdateBeforeSave": 5000,
            "db.demo.index.MapPoint.y.optimizationThreshold": 100000,
            "db.demo.index.MyFruit.color.entryPointSize": 64,
            "db.demo.index.MyFruit.color.items": 10,
            "db.demo.index.MyFruit.color.maxUpdateBeforeSave": 5000,
            "db.demo.index.MyFruit.color.optimizationThreshold": 100000,
            "db.demo.index.MyFruit.flavor.entryPointSize": 64,
            "db.demo.index.MyFruit.flavor.items": 0,
            "db.demo.index.MyFruit.flavor.maxUpdateBeforeSave": 5000,
            "db.demo.index.MyFruit.flavor.optimizationThreshold": 100000,
            "db.demo.index.MyFruit.name.entryPointSize": 64,
            "db.demo.index.MyFruit.name.items": 5000,
            "db.demo.index.MyFruit.name.maxUpdateBeforeSave": 5000,
            "db.demo.index.MyFruit.name.optimizationThreshold": 100000,
            "db.demo.index.MyProfile.name.entryPointSize": 64,
            "db.demo.index.MyProfile.name.items": 3,
            "db.demo.index.MyProfile.name.maxUpdateBeforeSave": 5000,
            "db.demo.index.MyProfile.name.optimizationThreshold": 100000,
            "db.demo.index.Profile.hash.entryPointSize": 64,
            "db.demo.index.Profile.hash.items": 5,
            "db.demo.index.Profile.hash.maxUpdateBeforeSave": 5000,
            "db.demo.index.Profile.hash.optimizationThreshold": 100000,
            "db.demo.index.Profile.name.entryPointSize": 64,
            "db.demo.index.Profile.name.items": 20,
            "db.demo.index.Profile.name.maxUpdateBeforeSave": 5000,
            "db.demo.index.Profile.name.optimizationThreshold": 100000,
            "db.demo.index.Profile.nick.entryPointSize": 64,
            "db.demo.index.Profile.nick.items": 38,
            "db.demo.index.Profile.nick.maxUpdateBeforeSave": 5000,
            "db.demo.index.Profile.nick.optimizationThreshold": 100000,
            "db.demo.index.PropertyIndexFirstIndex.entryPointSize": 64,
            "db.demo.index.PropertyIndexFirstIndex.items": 0,
            "db.demo.index.PropertyIndexFirstIndex.maxUpdateBeforeSave": 5000,
            "db.demo.index.PropertyIndexFirstIndex.optimizationThreshold": 100000,
      "db.demo.index.PropertyIndexSecondIndex.entryPointSize": 64,
      "db.demo.index.PropertyIndexSecondIndex.items": 0,
      "db.demo.index.PropertyIndexSecondIndex.maxUpdateBeforeSave": 5000,
      "db.demo.index.PropertyIndexSecondIndex.optimizationThreshold": 100000,
      "db.demo.index.PropertyIndexTestClass.prop1.entryPointSize": 64,
      "db.demo.index.PropertyIndexTestClass.prop1.items": 0,
      "db.demo.index.PropertyIndexTestClass.prop1.maxUpdateBeforeSave": 5000,
      "db.demo.index.PropertyIndexTestClass.prop1.optimizationThreshold": 100000,
      "db.demo.index.SQLDropClassCompositeIndex.entryPointSize": 64,
      "db.demo.index.SQLDropClassCompositeIndex.items": 0,
      "db.demo.index.SQLDropClassCompositeIndex.maxUpdateBeforeSave": 5000,
      "db.demo.index.SQLDropClassCompositeIndex.optimizationThreshold": 100000,
      "db.demo.index.SQLDropIndexCompositeIndex.entryPointSize": 64,
      "db.demo.index.SQLDropIndexCompositeIndex.items": 0,
      "db.demo.index.SQLDropIndexCompositeIndex.maxUpdateBeforeSave": 5000,
      "db.demo.index.SQLDropIndexCompositeIndex.optimizationThreshold": 100000,
      "db.demo.index.SQLDropIndexTestClass.prop1.entryPointSize": 64,
      "db.demo.index.SQLDropIndexTestClass.prop1.items": 0,
      "db.demo.index.SQLDropIndexTestClass.prop1.maxUpdateBeforeSave": 5000,
      "db.demo.index.SQLDropIndexTestClass.prop1.optimizationThreshold": 100000,
      "db.demo.index.SQLDropIndexWithoutClass.entryPointSize": 64,
      "db.demo.index.SQLDropIndexWithoutClass.items": 0,
      "db.demo.index.SQLDropIndexWithoutClass.maxUpdateBeforeSave": 5000,
      "db.demo.index.SQLDropIndexWithoutClass.optimizationThreshold": 100000,
      "db.demo.index.SQLSelectCompositeIndexDirectSearchTestIndex.entryPointSize": 64,
      "db.demo.index.SQLSelectCompositeIndexDirectSearchTestIndex.items": 0,
      "db.demo.index.SQLSelectCompositeIndexDirectSearchTestIndex.maxUpdateBeforeSave": 5000,
      "db.demo.index.SQLSelectCompositeIndexDirectSearchTestIndex.optimizationThreshold": 100000,
      "db.demo.index.SchemaSharedIndexCompositeIndex.entryPointSize": 64,
      "db.demo.index.SchemaSharedIndexCompositeIndex.items": 0,
      "db.demo.index.SchemaSharedIndexCompositeIndex.maxUpdateBeforeSave": 5000,
      "db.demo.index.SchemaSharedIndexCompositeIndex.optimizationThreshold": 100000,
      "db.demo.index.TRPerson.name.entryPointSize": 64,
      "db.demo.index.TRPerson.name.items": 4,
      "db.demo.index.TRPerson.name.maxUpdateBeforeSave": 5000,
      "db.demo.index.TRPerson.name.optimizationThreshold": 100000,
      "db.demo.index.TRPerson.surname.entryPointSize": 64,
      "db.demo.index.TRPerson.surname.items": 3,
      "db.demo.index.TRPerson.surname.maxUpdateBeforeSave": 5000,
      "db.demo.index.TRPerson.surname.optimizationThreshold": 100000,
      "db.demo.index.TestClass.name.entryPointSize": 64,
      "db.demo.index.TestClass.name.items": 2,
      "db.demo.index.TestClass.name.maxUpdateBeforeSave": 5000,
      "db.demo.index.TestClass.name.optimizationThreshold": 100000,
      "db.demo.index.TestClass.testLink.entryPointSize": 64,
      "db.demo.index.TestClass.testLink.items": 2,
      "db.demo.index.TestClass.testLink.maxUpdateBeforeSave": 5000,
      "db.demo.index.TestClass.testLink.optimizationThreshold": 100000,
      "db.demo.index.TransactionUniqueIndexWithDotTest.label.entryPointSize": 64,
      "db.demo.index.TransactionUniqueIndexWithDotTest.label.items": 1,
      "db.demo.index.TransactionUniqueIndexWithDotTest.label.maxUpdateBeforeSave": 5000,
      "db.demo.index.TransactionUniqueIndexWithDotTest.label.optimizationThreshold": 100000,
      "db.demo.index.Whiz.account.entryPointSize": 64,
      "db.demo.index.Whiz.account.items": 1,
      "db.demo.index.Whiz.account.maxUpdateBeforeSave": 5000,
      "db.demo.index.Whiz.account.optimizationThreshold": 100000,
      "db.demo.index.Whiz.text.entryPointSize": 64,
      "db.demo.index.Whiz.text.items": 275,
      "db.demo.index.Whiz.text.maxUpdateBeforeSave": 5000,
      "db.demo.index.Whiz.text.optimizationThreshold": 100000,
      "db.demo.index.a.entryPointSize": 64,
      "db.demo.index.a.items": 0,
      "db.demo.index.a.maxUpdateBeforeSave": 5000,
      "db.demo.index.a.optimizationThreshold": 100000,
      "db.demo.index.anotherproperty.entryPointSize": 64,
      "db.demo.index.anotherproperty.items": 0,
      "db.demo.index.anotherproperty.maxUpdateBeforeSave": 5000,
      "db.demo.index.anotherproperty.optimizationThreshold": 100000,
      "db.demo.index.byte-array-manualIndex-notunique.entryPointSize": 64,
      "db.demo.index.byte-array-manualIndex-notunique.items": 6,
      "db.demo.index.byte-array-manualIndex-notunique.maxUpdateBeforeSave": 5000,
      "db.demo.index.byte-array-manualIndex-notunique.optimizationThreshold": 100000,
      "db.demo.index.byte-array-manualIndex.entryPointSize": 64,
      "db.demo.index.byte-array-manualIndex.items": 11,
      "db.demo.index.byte-array-manualIndex.maxUpdateBeforeSave": 5000,
      "db.demo.index.byte-array-manualIndex.optimizationThreshold": 100000,
            "db.demo.index.byteArrayKeyIndex.entryPointSize": 64,
            "db.demo.index.byteArrayKeyIndex.items": 2,
            "db.demo.index.byteArrayKeyIndex.maxUpdateBeforeSave": 5000,
            "db.demo.index.byteArrayKeyIndex.optimizationThreshold": 100000,
            "db.demo.index.classIndexManagerComposite.entryPointSize": 64,
            "db.demo.index.classIndexManagerComposite.items": 0,
            "db.demo.index.classIndexManagerComposite.maxUpdateBeforeSave": 5000,
            "db.demo.index.classIndexManagerComposite.optimizationThreshold": 100000,
            "db.demo.index.classIndexManagerTestClass.prop1.entryPointSize": 64,
            "db.demo.index.classIndexManagerTestClass.prop1.items": 0,
            "db.demo.index.classIndexManagerTestClass.prop1.maxUpdateBeforeSave": 5000,
            "db.demo.index.classIndexManagerTestClass.prop1.optimizationThreshold": 100000,
            "db.demo.index.classIndexManagerTestClass.prop2.entryPointSize": 64,
            "db.demo.index.classIndexManagerTestClass.prop2.items": 0,
            "db.demo.index.classIndexManagerTestClass.prop2.maxUpdateBeforeSave": 5000,
            "db.demo.index.classIndexManagerTestClass.prop2.optimizationThreshold": 100000,
            "db.demo.index.classIndexManagerTestClass.prop4.entryPointSize": 64,
            "db.demo.index.classIndexManagerTestClass.prop4.items": 0,
            "db.demo.index.classIndexManagerTestClass.prop4.maxUpdateBeforeSave": 5000,
            "db.demo.index.classIndexManagerTestClass.prop4.optimizationThreshold": 100000,
            "db.demo.index.classIndexManagerTestClass.prop6.entryPointSize": 64,
            "db.demo.index.classIndexManagerTestClass.prop6.items": 0,
            "db.demo.index.classIndexManagerTestClass.prop6.maxUpdateBeforeSave": 5000,
            "db.demo.index.classIndexManagerTestClass.prop6.optimizationThreshold": 100000,
            "db.demo.index.classIndexManagerTestIndexByKey.entryPointSize": 64,
            "db.demo.index.classIndexManagerTestIndexByKey.items": 0,
            "db.demo.index.classIndexManagerTestIndexByKey.maxUpdateBeforeSave": 5000,
            "db.demo.index.classIndexManagerTestIndexByKey.optimizationThreshold": 100000,
            "db.demo.index.classIndexManagerTestIndexByValue.entryPointSize": 64,
            "db.demo.index.classIndexManagerTestIndexByValue.items": 0,
            "db.demo.index.classIndexManagerTestIndexByValue.maxUpdateBeforeSave": 5000,
            "db.demo.index.classIndexManagerTestIndexByValue.optimizationThreshold": 100000,
            "db.demo.index.classIndexManagerTestIndexValueAndCollection.entryPointSize": 64,
            "db.demo.index.classIndexManagerTestIndexValueAndCollection.items": 0,
            "db.demo.index.classIndexManagerTestIndexValueAndCollection.maxUpdateBeforeSave": 5000,
            "db.demo.index.classIndexManagerTestIndexValueAndCollection.optimizationThreshold": 100000,
            "db.demo.index.classIndexManagerTestSuperClass.prop0.entryPointSize": 64,
            "db.demo.index.classIndexManagerTestSuperClass.prop0.items": 0,
            "db.demo.index.classIndexManagerTestSuperClass.prop0.maxUpdateBeforeSave": 5000,
            "db.demo.index.classIndexManagerTestSuperClass.prop0.optimizationThreshold": 100000,
            "db.demo.index.compositeByteArrayKey.entryPointSize": 64,
            "db.demo.index.compositeByteArrayKey.items": 4,
            "db.demo.index.compositeByteArrayKey.maxUpdateBeforeSave": 5000,
            "db.demo.index.compositeByteArrayKey.optimizationThreshold": 100000,
            "db.demo.index.compositeIndexWithoutSchema.entryPointSize": 64,
            "db.demo.index.compositeIndexWithoutSchema.items": 0,
            "db.demo.index.compositeIndexWithoutSchema.maxUpdateBeforeSave": 5000,
            "db.demo.index.compositeIndexWithoutSchema.optimizationThreshold": 100000,
            "db.demo.index.compositeone.entryPointSize": 64,
            "db.demo.index.compositeone.items": 0,
            "db.demo.index.compositeone.maxUpdateBeforeSave": 5000,
            "db.demo.index.compositeone.optimizationThreshold": 100000,
            "db.demo.index.compositetwo.entryPointSize": 64,
            "db.demo.index.compositetwo.items": 0,
            "db.demo.index.compositetwo.maxUpdateBeforeSave": 5000,
            "db.demo.index.compositetwo.optimizationThreshold": 100000,
            "db.demo.index.curotorCompositeIndex.entryPointSize": 64,
            "db.demo.index.curotorCompositeIndex.items": 0,
            "db.demo.index.curotorCompositeIndex.maxUpdateBeforeSave": 5000,
            "db.demo.index.curotorCompositeIndex.optimizationThreshold": 100000,
            "db.demo.index.dictionary.entryPointSize": 64,
            "db.demo.index.dictionary.items": 2,
            "db.demo.index.dictionary.maxUpdateBeforeSave": 5000,
            "db.demo.index.dictionary.optimizationThreshold": 100000,
            "db.demo.index.diplomaThesisUnique.entryPointSize": 64,
            "db.demo.index.diplomaThesisUnique.items": 3,
            "db.demo.index.diplomaThesisUnique.maxUpdateBeforeSave": 5000,
            "db.demo.index.diplomaThesisUnique.optimizationThreshold": 100000,
            "db.demo.index.equalityIdx.entryPointSize": 64,
            "db.demo.index.equalityIdx.items": 0,
            "db.demo.index.equalityIdx.maxUpdateBeforeSave": 5000,
            "db.demo.index.equalityIdx.optimizationThreshold": 100000,
            "db.demo.index.idx.entryPointSize": 64,
            "db.demo.index.idx.items": 2,
            "db.demo.index.idx.maxUpdateBeforeSave": 5000,
            "db.demo.index.idx.optimizationThreshold": 100000,
            "db.demo.index.idxTerm.entryPointSize": 64,
            "db.demo.index.idxTerm.items": 1,
            "db.demo.index.idxTerm.maxUpdateBeforeSave": 5000,
            "db.demo.index.idxTerm.optimizationThreshold": 100000,
            "db.demo.index.idxTransactionUniqueIndexTest.entryPointSize": 64,
            "db.demo.index.idxTransactionUniqueIndexTest.items": 1,
            "db.demo.index.idxTransactionUniqueIndexTest.maxUpdateBeforeSave": 5000,
            "db.demo.index.idxTransactionUniqueIndexTest.optimizationThreshold": 100000,
            "db.demo.index.idxTxAwareMultiValueGetEntriesTest.entryPointSize": 64,
            "db.demo.index.idxTxAwareMultiValueGetEntriesTest.items": 0,
            "db.demo.index.idxTxAwareMultiValueGetEntriesTest.maxUpdateBeforeSave": 5000,
            "db.demo.index.idxTxAwareMultiValueGetEntriesTest.optimizationThreshold": 100000,
            "db.demo.index.idxTxAwareMultiValueGetTest.entryPointSize": 64,
            "db.demo.index.idxTxAwareMultiValueGetTest.items": 0,
            "db.demo.index.idxTxAwareMultiValueGetTest.maxUpdateBeforeSave": 5000,
            "db.demo.index.idxTxAwareMultiValueGetTest.optimizationThreshold": 100000,
            "db.demo.index.idxTxAwareMultiValueGetValuesTest.entryPointSize": 64,
            "db.demo.index.idxTxAwareMultiValueGetValuesTest.items": 0,
            "db.demo.index.idxTxAwareMultiValueGetValuesTest.maxUpdateBeforeSave": 5000,
            "db.demo.index.idxTxAwareMultiValueGetValuesTest.optimizationThreshold": 100000,
            "db.demo.index.idxTxAwareOneValueGetEntriesTest.entryPointSize": 64,
            "db.demo.index.idxTxAwareOneValueGetEntriesTest.items": 0,
            "db.demo.index.idxTxAwareOneValueGetEntriesTest.maxUpdateBeforeSave": 5000,
            "db.demo.index.idxTxAwareOneValueGetEntriesTest.optimizationThreshold": 100000,
            "db.demo.index.idxTxAwareOneValueGetTest.entryPointSize": 64,
            "db.demo.index.idxTxAwareOneValueGetTest.items": 0,
            "db.demo.index.idxTxAwareOneValueGetTest.maxUpdateBeforeSave": 5000,
            "db.demo.index.idxTxAwareOneValueGetTest.optimizationThreshold": 100000,
            "db.demo.index.idxTxAwareOneValueGetValuesTest.entryPointSize": 64,
            "db.demo.index.idxTxAwareOneValueGetValuesTest.items": 0,
            "db.demo.index.idxTxAwareOneValueGetValuesTest.maxUpdateBeforeSave": 5000,
            "db.demo.index.idxTxAwareOneValueGetValuesTest.optimizationThreshold": 100000,
            "db.demo.index.inIdx.entryPointSize": 64,
            "db.demo.index.inIdx.items": 0,
            "db.demo.index.inIdx.maxUpdateBeforeSave": 5000,
            "db.demo.index.inIdx.optimizationThreshold": 100000,
            "db.demo.index.indexForMap.entryPointSize": 64,
            "db.demo.index.indexForMap.items": 0,
            "db.demo.index.indexForMap.maxUpdateBeforeSave": 5000,
            "db.demo.index.indexForMap.optimizationThreshold": 100000,
            "db.demo.index.indexWithoutSchema.entryPointSize": 64,
            "db.demo.index.indexWithoutSchema.items": 0,
            "db.demo.index.indexWithoutSchema.maxUpdateBeforeSave": 5000,
            "db.demo.index.indexWithoutSchema.optimizationThreshold": 100000,
            "db.demo.index.indexfive.entryPointSize": 64,
            "db.demo.index.indexfive.items": 0,
            "db.demo.index.indexfive.maxUpdateBeforeSave": 5000,
            "db.demo.index.indexfive.optimizationThreshold": 100000,
            "db.demo.index.indexfour.entryPointSize": 64,
            "db.demo.index.indexfour.items": 0,
            "db.demo.index.indexfour.maxUpdateBeforeSave": 5000,
            "db.demo.index.indexfour.optimizationThreshold": 100000,
            "db.demo.index.indexone.entryPointSize": 64,
            "db.demo.index.indexone.items": 0,
            "db.demo.index.indexone.maxUpdateBeforeSave": 5000,
            "db.demo.index.indexone.optimizationThreshold": 100000,
            "db.demo.index.indexsix.entryPointSize": 64,
            "db.demo.index.indexsix.items": 0,
            "db.demo.index.indexsix.maxUpdateBeforeSave": 5000,
            "db.demo.index.indexsix.optimizationThreshold": 100000,
            "db.demo.index.indexthree.entryPointSize": 64,
            "db.demo.index.indexthree.items": 0,
            "db.demo.index.indexthree.maxUpdateBeforeSave": 5000,
            "db.demo.index.indexthree.optimizationThreshold": 100000,
            "db.demo.index.indextwo.entryPointSize": 64,
            "db.demo.index.indextwo.items": 0,
            "db.demo.index.indextwo.maxUpdateBeforeSave": 5000,
            "db.demo.index.indextwo.optimizationThreshold": 100000,
            "db.demo.index.linkCollectionIndex.entryPointSize": 64,
            "db.demo.index.linkCollectionIndex.items": 0,
            "db.demo.index.linkCollectionIndex.maxUpdateBeforeSave": 5000,
            "db.demo.index.linkCollectionIndex.optimizationThreshold": 100000,
            "db.demo.index.lpirtCurator.name.entryPointSize": 64,
            "db.demo.index.lpirtCurator.name.items": 0,
            "db.demo.index.lpirtCurator.name.maxUpdateBeforeSave": 5000,
            "db.demo.index.lpirtCurator.name.optimizationThreshold": 100000,
              "db.demo.index.lpirtCurator.salary.entryPointSize": 64,
              "db.demo.index.lpirtCurator.salary.items": 0,
              "db.demo.index.lpirtCurator.salary.maxUpdateBeforeSave": 5000,
              "db.demo.index.lpirtCurator.salary.optimizationThreshold": 100000,
              "db.demo.index.lpirtDiploma.GPA.entryPointSize": 64,
              "db.demo.index.lpirtDiploma.GPA.items": 3,
              "db.demo.index.lpirtDiploma.GPA.maxUpdateBeforeSave": 5000,
              "db.demo.index.lpirtDiploma.GPA.optimizationThreshold": 100000,
              "db.demo.index.lpirtDiploma.thesis.entryPointSize": 64,
              "db.demo.index.lpirtDiploma.thesis.items": 54,
              "db.demo.index.lpirtDiploma.thesis.maxUpdateBeforeSave": 5000,
              "db.demo.index.lpirtDiploma.thesis.optimizationThreshold": 100000,
              "db.demo.index.lpirtGroup.curator.entryPointSize": 64,
              "db.demo.index.lpirtGroup.curator.items": 0,
              "db.demo.index.lpirtGroup.curator.maxUpdateBeforeSave": 5000,
              "db.demo.index.lpirtGroup.curator.optimizationThreshold": 100000,
              "db.demo.index.lpirtGroup.name.entryPointSize": 64,
              "db.demo.index.lpirtGroup.name.items": 0,
              "db.demo.index.lpirtGroup.name.maxUpdateBeforeSave": 5000,
              "db.demo.index.lpirtGroup.name.optimizationThreshold": 100000,
              "db.demo.index.lpirtStudent.group.entryPointSize": 64,
              "db.demo.index.lpirtStudent.group.items": 0,
              "db.demo.index.lpirtStudent.group.maxUpdateBeforeSave": 5000,
              "db.demo.index.lpirtStudent.group.optimizationThreshold": 100000,
              "db.demo.index.lpirtStudent.name.entryPointSize": 64,
              "db.demo.index.lpirtStudent.name.items": 0,
              "db.demo.index.lpirtStudent.name.maxUpdateBeforeSave": 5000,
              "db.demo.index.lpirtStudent.name.optimizationThreshold": 100000,
              "db.demo.index.manualTxIndexTest.entryPointSize": 64,
              "db.demo.index.manualTxIndexTest.items": 1,
              "db.demo.index.manualTxIndexTest.maxUpdateBeforeSave": 5000,
              "db.demo.index.manualTxIndexTest.optimizationThreshold": 100000,
              "db.demo.index.mapIndexTestKey.entryPointSize": 64,
              "db.demo.index.mapIndexTestKey.items": 0,
              "db.demo.index.mapIndexTestKey.maxUpdateBeforeSave": 5000,
              "db.demo.index.mapIndexTestKey.optimizationThreshold": 100000,
              "db.demo.index.mapIndexTestValue.entryPointSize": 64,
              "db.demo.index.mapIndexTestValue.items": 0,
              "db.demo.index.mapIndexTestValue.maxUpdateBeforeSave": 5000,
              "db.demo.index.mapIndexTestValue.optimizationThreshold": 100000,
              "db.demo.index.newV.f_int.entryPointSize": 64,
              "db.demo.index.newV.f_int.items": 3,
              "db.demo.index.newV.f_int.maxUpdateBeforeSave": 5000,
              "db.demo.index.newV.f_int.optimizationThreshold": 100000,
              "db.demo.index.nullkey.entryPointSize": 64,
              "db.demo.index.nullkey.items": 0,
              "db.demo.index.nullkey.maxUpdateBeforeSave": 5000,
              "db.demo.index.nullkey.optimizationThreshold": 100000,
              "db.demo.index.nullkeytwo.entryPointSize": 64,
              "db.demo.index.nullkeytwo.items": 0,
              "db.demo.index.nullkeytwo.maxUpdateBeforeSave": 5000,
              "db.demo.index.nullkeytwo.optimizationThreshold": 100000,
              "db.demo.index.propOne1.entryPointSize": 64,
              "db.demo.index.propOne1.items": 0,
              "db.demo.index.propOne1.maxUpdateBeforeSave": 5000,
              "db.demo.index.propOne1.optimizationThreshold": 100000,
              "db.demo.index.propOne2.entryPointSize": 64,
              "db.demo.index.propOne2.items": 0,
              "db.demo.index.propOne2.maxUpdateBeforeSave": 5000,
              "db.demo.index.propOne2.optimizationThreshold": 100000,
              "db.demo.index.propOne3.entryPointSize": 64,
              "db.demo.index.propOne3.items": 0,
              "db.demo.index.propOne3.maxUpdateBeforeSave": 5000,
              "db.demo.index.propOne3.optimizationThreshold": 100000,
              "db.demo.index.propOne4.entryPointSize": 64,
              "db.demo.index.propOne4.items": 0,
              "db.demo.index.propOne4.maxUpdateBeforeSave": 5000,
              "db.demo.index.propOne4.optimizationThreshold": 100000,
              "db.demo.index.propertyone.entryPointSize": 64,
              "db.demo.index.propertyone.items": 0,
              "db.demo.index.propertyone.maxUpdateBeforeSave": 5000,
              "db.demo.index.propertyone.optimizationThreshold": 100000,
              "db.demo.index.simplekey.entryPointSize": 64,
              "db.demo.index.simplekey.items": 0,
              "db.demo.index.simplekey.maxUpdateBeforeSave": 5000,
              "db.demo.index.simplekey.optimizationThreshold": 100000,
              "db.demo.index.simplekeytwo.entryPointSize": 64,
              "db.demo.index.simplekeytwo.items": 0,
              "db.demo.index.simplekeytwo.maxUpdateBeforeSave": 5000,
              "db.demo.index.simplekeytwo.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexCompositeIndex.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexCompositeIndex.items": 0,
              "db.demo.index.sqlCreateIndexCompositeIndex.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexCompositeIndex.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexCompositeIndex2.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexCompositeIndex2.items": 0,
              "db.demo.index.sqlCreateIndexCompositeIndex2.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexCompositeIndex2.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexEmbeddedListIndex.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexEmbeddedListIndex.items": 0,
              "db.demo.index.sqlCreateIndexEmbeddedListIndex.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexEmbeddedListIndex.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexEmbeddedMapByKeyIndex.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexEmbeddedMapByKeyIndex.items": 0,
              "db.demo.index.sqlCreateIndexEmbeddedMapByKeyIndex.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexEmbeddedMapByKeyIndex.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexEmbeddedMapByValueIndex.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexEmbeddedMapByValueIndex.items": 0,
              "db.demo.index.sqlCreateIndexEmbeddedMapByValueIndex.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexEmbeddedMapByValueIndex.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexEmbeddedMapIndex.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexEmbeddedMapIndex.items": 0,
              "db.demo.index.sqlCreateIndexEmbeddedMapIndex.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexEmbeddedMapIndex.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexTestClass.prop1.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexTestClass.prop1.items": 0,
              "db.demo.index.sqlCreateIndexTestClass.prop1.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexTestClass.prop1.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexTestClass.prop3.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexTestClass.prop3.items": 0,
              "db.demo.index.sqlCreateIndexTestClass.prop3.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexTestClass.prop3.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexTestClass.prop5.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexTestClass.prop5.items": 0,
              "db.demo.index.sqlCreateIndexTestClass.prop5.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexTestClass.prop5.optimizationThreshold": 100000,
              "db.demo.index.sqlCreateIndexWithoutClass.entryPointSize": 64,
              "db.demo.index.sqlCreateIndexWithoutClass.items": 0,
              "db.demo.index.sqlCreateIndexWithoutClass.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlCreateIndexWithoutClass.optimizationThreshold": 100000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedList.entryPointSize": 64,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedList.items": 0,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedList.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedList.optimizationThreshold": 100000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedListTwoProp8.entryPointSize": 64,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedListTwoProp8.items": 0,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedListTwoProp8.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedListTwoProp8.optimizationThreshold": 100000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKey.entryPointSize": 64,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKey.items": 0,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKey.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKey.optimizationThreshold": 100000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKeyProp8.entryPointSize": 64,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKeyProp8.items": 0,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKeyProp8.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKeyProp8.optimizationThreshold": 100000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValue.entryPointSize": 64,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValue.items": 0,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValue.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValue.optimizationThreshold": 100000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValueProp8.entryPointSize": 64,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValueProp8.items": 0,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValueProp8.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValueProp8.optimizationThreshold": 100000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedSetProp8.entryPointSize": 64,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedSetProp8.items": 0,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedSetProp8.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlSelectIndexReuseTestEmbeddedSetProp8.optimizationThreshold": 100000,
              "db.demo.index.sqlSelectIndexReuseTestProp9EmbeddedSetProp8.entryPointSize": 64,
              "db.demo.index.sqlSelectIndexReuseTestProp9EmbeddedSetProp8.items": 0,
              "db.demo.index.sqlSelectIndexReuseTestProp9EmbeddedSetProp8.maxUpdateBeforeSave": 5000,
              "db.demo.index.sqlSelectIndexReuseTestProp9EmbeddedSetProp8.optimizationThreshold": 100000,
        "db.demo.index.studentDiplomaAndNameIndex.entryPointSize": 64,
        "db.demo.index.studentDiplomaAndNameIndex.items": 0,
        "db.demo.index.studentDiplomaAndNameIndex.maxUpdateBeforeSave": 5000,
        "db.demo.index.studentDiplomaAndNameIndex.optimizationThreshold": 100000,
        "db.demo.index.testIdx.entryPointSize": 64,
        "db.demo.index.testIdx.items": 1,
        "db.demo.index.testIdx.maxUpdateBeforeSave": 5000,
        "db.demo.index.testIdx.optimizationThreshold": 100000,
        "db.demo.index.test_class_by_data.entryPointSize": 64,
        "db.demo.index.test_class_by_data.items": 0,
        "db.demo.index.test_class_by_data.maxUpdateBeforeSave": 5000,
        "db.demo.index.test_class_by_data.optimizationThreshold": 100000,
        "db.demo.index.twoclassproperty.entryPointSize": 64,
        "db.demo.index.twoclassproperty.items": 0,
        "db.demo.index.twoclassproperty.maxUpdateBeforeSave": 5000,
        "db.demo.index.twoclassproperty.optimizationThreshold": 100000,
        "db.demo.index.vertexA_name_idx.entryPointSize": 64,
        "db.demo.index.vertexA_name_idx.items": 2,
        "db.demo.index.vertexA_name_idx.maxUpdateBeforeSave": 5000,
        "db.demo.index.vertexA_name_idx.optimizationThreshold": 100000,
        "db.demo.index.vertexB_name_idx.entryPointSize": 64,
        "db.demo.index.vertexB_name_idx.items": 2,
        "db.demo.index.vertexB_name_idx.maxUpdateBeforeSave": 5000,
        "db.demo.index.vertexB_name_idx.optimizationThreshold": 100000,
        "db.subTest.cache.level1.current": 0,
        "db.subTest.cache.level1.enabled": false,
        "db.subTest.cache.level1.max": -1,
        "db.subTest.cache.level2.current": 0,
        "db.subTest.cache.level2.enabled": false,
        "db.subTest.cache.level2.max": -1,
        "db.subTest.data.holeSize": 0,
        "db.subTest.data.holes": 0,
        "db.subTest.index.dictionary.entryPointSize": 64,
        "db.subTest.index.dictionary.items": 0,
        "db.subTest.index.dictionary.maxUpdateBeforeSave": 5000,
        "db.subTest.index.dictionary.optimizationThreshold": 100000,
        "db.temp.cache.level1.current": 0,
        "db.temp.cache.level1.enabled": false,
        "db.temp.cache.level1.max": -1,
        "db.temp.cache.level2.current": 3,
        "db.temp.cache.level2.enabled": true,
        "db.temp.cache.level2.max": -1,
        "db.temp.index.dictionary.entryPointSize": 64,
        "db.temp.index.dictionary.items": 0,
        "db.temp.index.dictionary.maxUpdateBeforeSave": 5000,
        "db.temp.index.dictionary.optimizationThreshold": 100000,
        "process.network.channel.binary./0:0:0:0:0:0:0:1:451822480.flushes": 0,
        "process.network.channel.binary./0:0:0:0:0:0:0:1:451822480.receivedBytes": 513,
        "process.network.channel.binary./0:0:0:0:0:0:0:1:451822480.transmittedBytes": 0,
        "process.network.channel.binary./127.0.0.1:451282424.flushes": 3,
        "process.network.channel.binary./127.0.0.1:451282424.receivedBytes": 98,
        "process.network.channel.binary./127.0.0.1:451282424.transmittedBytes": 16,
        "process.network.channel.binary./127.0.0.1:451292424.flushes": 3,
        "process.network.channel.binary./127.0.0.1:451292424.receivedBytes": 72,
        "process.network.channel.binary./127.0.0.1:451292424.transmittedBytes": 17,
        "process.network.channel.binary./127.0.0.1:451352424.flushes": 3,
        "process.network.channel.binary./127.0.0.1:451352424.receivedBytes": 79,
        "process.network.channel.binary./127.0.0.1:451352424.transmittedBytes": 134,
        "process.network.channel.binary./127.0.0.1:451362424.flushes": 3,
        "process.network.channel.binary./127.0.0.1:451362424.receivedBytes": 105,
        "process.network.channel.binary./127.0.0.1:451362424.transmittedBytes": 16,
        "process.network.channel.binary./127.0.0.1:451382424.flushes": 3,
        "process.network.channel.binary./127.0.0.1:451382424.receivedBytes": 79,
        "process.network.channel.binary./127.0.0.1:451382424.transmittedBytes": 16,
        "process.network.channel.binary./127.0.0.1:451392424.flushes": 3,
        "process.network.channel.binary./127.0.0.1:451392424.receivedBytes": 79,
        "process.network.channel.binary./127.0.0.1:451392424.transmittedBytes": 134,
        "process.network.channel.binary./127.0.0.1:451402424.flushes": 3,
        "process.network.channel.binary./127.0.0.1:451402424.receivedBytes": 105,
        "process.network.channel.binary./127.0.0.1:451402424.transmittedBytes": 16,
        "process.network.channel.binary./127.0.0.1:451422424.flushes": 3,
        "process.network.channel.binary./127.0.0.1:451422424.receivedBytes": 79,
        "process.network.channel.binary./127.0.0.1:451422424.transmittedBytes": 16,
        "process.network.channel.binary./127.0.0.1:451432424.flushes": 3,
        "process.network.channel.binary./127.0.0.1:451432424.receivedBytes": 72,
        "process.network.channel.binary./127.0.0.1:451432424.transmittedBytes": 127,
            "process.network.channel.binary./127.0.0.1:451442424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451442424.receivedBytes": 98,
            "process.network.channel.binary./127.0.0.1:451442424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451452424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451452424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451452424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451462424.flushes": 7,
            "process.network.channel.binary./127.0.0.1:451462424.receivedBytes": 194,
            "process.network.channel.binary./127.0.0.1:451462424.transmittedBytes": 2606,
            "process.network.channel.binary./127.0.0.1:451472424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451472424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451472424.transmittedBytes": 127,
            "process.network.channel.binary./127.0.0.1:451482424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451482424.receivedBytes": 98,
            "process.network.channel.binary./127.0.0.1:451482424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451492424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451492424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451492424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451502424.flushes": 7,
            "process.network.channel.binary./127.0.0.1:451502424.receivedBytes": 194,
            "process.network.channel.binary./127.0.0.1:451502424.transmittedBytes": 2606,
            "process.network.channel.binary./127.0.0.1:451512424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451512424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451512424.transmittedBytes": 127,
            "process.network.channel.binary./127.0.0.1:451522424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451522424.receivedBytes": 98,
            "process.network.channel.binary./127.0.0.1:451522424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451532424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451532424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451532424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451542424.flushes": 7,
            "process.network.channel.binary./127.0.0.1:451542424.receivedBytes": 194,
            "process.network.channel.binary./127.0.0.1:451542424.transmittedBytes": 2606,
            "process.network.channel.binary./127.0.0.1:451552424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451552424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451552424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451562424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451562424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451562424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451572424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451572424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451572424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451582424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451582424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451582424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451592424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451592424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451592424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451602424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451602424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451602424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451612424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451612424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451612424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451622424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451622424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451622424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451632424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451632424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451632424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451642424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451642424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451642424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451652424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451652424.receivedBytes": 98,
            "process.network.channel.binary./127.0.0.1:451652424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451672424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451672424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451672424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451682424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451682424.receivedBytes": 98,
            "process.network.channel.binary./127.0.0.1:451682424.transmittedBytes": 16,
            "process.network.channel.binary./127.0.0.1:451692424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451692424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451692424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451702424.flushes": 76545,
            "process.network.channel.binary./127.0.0.1:451702424.receivedBytes": 4937639,
            "process.network.channel.binary./127.0.0.1:451702424.transmittedBytes": 53391585,
            "process.network.channel.binary./127.0.0.1:451712424.flushes": 3,
            "process.network.channel.binary./127.0.0.1:451712424.receivedBytes": 72,
            "process.network.channel.binary./127.0.0.1:451712424.transmittedBytes": 17,
            "process.network.channel.binary./127.0.0.1:451762424.flushes": 16176,
            "process.network.channel.binary./127.0.0.1:451762424.receivedBytes": 435578,
            "process.network.channel.binary./127.0.0.1:451762424.transmittedBytes": 7744941,
            "process.network.channel.binary./127.0.0.1:451772424.flushes": 16181,
            "process.network.channel.binary./127.0.0.1:451772424.receivedBytes": 446949,
            "process.network.channel.binary./127.0.0.1:451772424.transmittedBytes": 7932617,
            "process.network.channel.binary./127.0.0.1:451782424.flushes": 16103,
            "process.network.channel.binary./127.0.0.1:451782424.receivedBytes": 437708,
            "process.network.channel.binary./127.0.0.1:451782424.transmittedBytes": 7192022,
            "process.network.channel.binary./127.0.0.1:451792424.flushes": 15663,
            "process.network.channel.binary./127.0.0.1:451792424.receivedBytes": 422013,
            "process.network.channel.binary./127.0.0.1:451792424.transmittedBytes": 1128841,
            "process.network.channel.binary.flushes": 140851,
            "process.network.channel.binary.receivedBytes": 6687263,
            "process.network.channel.binary.transmittedBytes": 77419866,
            "process.runtime.availableMemory": 311502288,
            "process.runtime.maxMemory": 939524096,
            "process.runtime.totalMemory": 442368000,
            "server.connections.actives": 101,
            "system.config.cpus": 8,
            "system.disk.C.freeSpace": 50445692928,
            "system.disk.C.totalSpace": 127928365056,
            "system.disk.C.usableSpace": 50445692928,
            "system.disk.D.freeSpace": 0,
            "system.disk.D.totalSpace": 0,
            "system.disk.D.usableSpace": 0,
            "system.disk.G.freeSpace": 12820815872,
            "system.disk.G.totalSpace": 500103213056,
            "system.disk.G.usableSpace": 12820815872,
            "system.file.mmap.mappedPages": 177,
            "system.file.mmap.nonPooledBufferUsed": 0,
            "system.file.mmap.pooledBufferCreated": 0,
            "system.file.mmap.pooledBufferUsed": 0,
            "system.file.mmap.reusedPages": 31698774,
            "system.memory.alerts": 0,
            "system.memory.stream.resize": 21154
        },
        "chronos": {
            "db.0$db.close": {
                "entries": 4,
                "last": 16,
                "min": 0,
                "max": 16,
                "average": 4,
                "total": 16
            },
            "db.0$db.create": {
                "entries": 1,
                "last": 13,
                "min": 13,
                "max": 13,
                "average": 13,
                "total": 13
            },
            "db.0$db.createRecord": {
                "entries": 10,
                "last": 1,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 6
            },
            "db.0$db.data.createHole": {
                "entries": 14,
                "last": 2,
                "min": 0,
                "max": 2,
                "average": 0,
                "total": 8
            },
            "db.0$db.data.findClosestHole": {
                "entries": 11,
                "last": 0,
                "min": 0,
                "max": 0,
                "average": 0,
                "total": 0
            },
            "db.0$db.data.move": {
                "entries": 6,
                "last": 1,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 3
            },
            "db.0$db.data.recycled.notFound": {
                "entries": 7,
                "last": 0,
                "min": 0,
                "max": 0,
                "average": 0,
                "total": 0
            },
            "db.0$db.data.recycled.partial": {
                "entries": 11,
                "last": 0,
                "min": 0,
                "max": 0,
                "average": 0,
                "total": 0
            },
            "db.0$db.data.updateHole": {
                "entries": 21,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 2
            },
            "db.0$db.delete": {
                "entries": 1,
                "last": 101,
                "min": 101,
                "max": 101,
                "average": 101,
                "total": 101
            },
            "db.0$db.metadata.load": {
                "entries": 3,
                "last": 0,
                "min": 0,
                "max": 0,
                "average": 0,
                "total": 0
            },
            "db.0$db.open": {
                "entries": 3,
                "last": 0,
                "min": 0,
                "max": 0,
                "average": 0,
                "total": 0
            },
            "db.0$db.readRecord": {
                "entries": 15,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 5
            },
            "db.0$db.updateRecord": {
                "entries": 18,
                "last": 2,
                "min": 0,
                "max": 2,
            "average": 0,
            "total": 9
        },
        "db.1$db.close": {
            "entries": 4,
            "last": 13,
            "min": 0,
            "max": 13,
            "average": 3,
            "total": 13
        },
        "db.1$db.create": {
            "entries": 1,
            "last": 15,
            "min": 15,
            "max": 15,
            "average": 15,
            "total": 15
        },
        "db.1$db.createRecord": {
            "entries": 10,
            "last": 1,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 5
        },
        "db.1$db.data.createHole": {
            "entries": 14,
            "last": 3,
            "min": 0,
            "max": 3,
            "average": 0,
            "total": 8
        },
        "db.1$db.data.findClosestHole": {
            "entries": 11,
            "last": 0,
            "min": 0,
            "max": 0,
            "average": 0,
            "total": 0
        },
        "db.1$db.data.move": {
            "entries": 6,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 3
        },
        "db.1$db.data.recycled.notFound": {
            "entries": 7,
            "last": 0,
            "min": 0,
            "max": 0,
            "average": 0,
            "total": 0
        },
        "db.1$db.data.recycled.partial": {
            "entries": 11,
            "last": 0,
            "min": 0,
            "max": 0,
            "average": 0,
            "total": 0
        },
        "db.1$db.data.updateHole": {
            "entries": 21,
            "last": 1,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 1
        },
        "db.1$db.delete": {
                "entries": 1,
                "last": 115,
                "min": 115,
                "max": 115,
                "average": 115,
                "total": 115
            },
            "db.1$db.metadata.load": {
                "entries": 3,
                "last": 0,
                "min": 0,
                "max": 0,
                "average": 0,
                "total": 0
            },
            "db.1$db.open": {
                "entries": 3,
                "last": 0,
                "min": 0,
                "max": 0,
                "average": 0,
                "total": 0
            },
            "db.1$db.readRecord": {
                "entries": 15,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 4
            },
            "db.1$db.updateRecord": {
                "entries": 18,
                "last": 3,
                "min": 0,
                "max": 3,
                "average": 0,
                "total": 7
            },
            "db.2$db.close": {
                "entries": 4,
                "last": 15,
                "min": 0,
                "max": 15,
                "average": 3,
                "total": 15
            },
            "db.2$db.create": {
                "entries": 1,
                "last": 17,
                "min": 17,
                "max": 17,
                "average": 17,
                "total": 17
            },
            "db.2$db.createRecord": {
                "entries": 10,
                "last": 1,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 5
            },
            "db.2$db.data.createHole": {
                "entries": 14,
                "last": 1,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 5
            },
            "db.2$db.data.findClosestHole": {
                "entries": 11,
                "last": 0,
                "min": 0,
                "max": 0,
        "average": 0,
        "total": 0
    },
    "db.2$db.data.move": {
        "entries": 6,
        "last": 0,
        "min": 0,
        "max": 1,
        "average": 0,
        "total": 1
    },
    "db.2$db.data.recycled.notFound": {
        "entries": 7,
        "last": 0,
        "min": 0,
        "max": 0,
        "average": 0,
        "total": 0
    },
    "db.2$db.data.recycled.partial": {
        "entries": 11,
        "last": 0,
        "min": 0,
        "max": 0,
        "average": 0,
        "total": 0
    },
    "db.2$db.data.updateHole": {
        "entries": 21,
        "last": 0,
        "min": 0,
        "max": 1,
        "average": 0,
        "total": 1
    },
    "db.2$db.delete": {
        "entries": 1,
        "last": 61,
        "min": 61,
        "max": 61,
        "average": 61,
        "total": 61
    },
    "db.2$db.metadata.load": {
        "entries": 3,
        "last": 0,
        "min": 0,
        "max": 0,
        "average": 0,
        "total": 0
    },
    "db.2$db.open": {
        "entries": 3,
        "last": 0,
        "min": 0,
        "max": 0,
        "average": 0,
        "total": 0
    },
    "db.2$db.readRecord": {
        "entries": 15,
        "last": 0,
        "min": 0,
        "max": 1,
        "average": 0,
        "total": 1
    },
    "db.2$db.updateRecord": {
        "entries": 18,
        "last": 1,
        "min": 0,
        "max": 1,
        "average": 0,
        "total": 5
    },
    "db.demo.close": {
            "entries": 1396,
            "last": 0,
            "min": 0,
            "max": 31,
            "average": 0,
            "total": 51
        },
        "db.demo.create": {
            "entries": 3,
            "last": 19,
            "min": 19,
            "max": 40,
            "average": 27,
            "total": 81
        },
        "db.demo.createRecord": {
            "entries": 35716,
            "last": 0,
            "min": 0,
            "max": 12,
            "average": 0,
            "total": 1187
        },
        "db.demo.data.createHole": {
            "entries": 58886,
            "last": 0,
            "min": 0,
            "max": 23,
            "average": 0,
            "total": 9822
        },
        "db.demo.data.findClosestHole": {
            "entries": 51022,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 181
        },
        "db.demo.data.move": {
            "entries": 1327946,
            "last": 0,
            "min": 0,
            "max": 16,
            "average": 0,
            "total": 4091
        },
        "db.demo.data.recycled.complete": {
            "entries": 24,
            "last": 0,
            "min": 0,
            "max": 0,
            "average": 0,
            "total": 0
        },
        "db.demo.data.recycled.notFound": {
            "entries": 16070,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 59
        },
        "db.demo.data.recycled.partial": {
            "entries": 57638,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 102
        },
        "db.demo.data.updateHole": {
            "entries": 108613,
            "last": 0,
            "min": 0,
            "max": 12,
                "average": 0,
                "total": 451
            },
            "db.demo.delete": {
                "entries": 2,
                "last": 61,
                "min": 61,
                "max": 124,
                "average": 92,
                "total": 185
            },
            "db.demo.deleteRecord": {
                "entries": 12362,
                "last": 0,
                "min": 0,
                "max": 24,
                "average": 0,
                "total": 4626
            },
            "db.demo.metadata.load": {
                "entries": 1423,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 49
            },
            "db.demo.open": {
                "entries": 1423,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 6
            },
            "db.demo.readRecord": {
                "entries": 476697,
                "last": 0,
                "min": 0,
                "max": 16,
                "average": 0,
                "total": 3071
            },
            "db.demo.synch": {
                "entries": 484,
                "last": 2,
                "min": 0,
                "max": 34,
                "average": 2,
                "total": 1251
            },
            "db.demo.updateRecord": {
                "entries": 180667,
                "last": 0,
                "min": 0,
                "max": 12,
                "average": 0,
                "total": 2343
            },
            "db.subTest.close": {
                "entries": 10,
                "last": 0,
                "min": 0,
                "max": 16,
                "average": 3,
                "total": 31
            },
            "db.subTest.create": {
                "entries": 2,
                "last": 44,
                "min": 18,
                "max": 44,
                "average": 31,
                "total": 62
            },
            "db.subTest.createRecord": {
```

```
        "entries": 20,
        "last": 1,
        "min": 0,
        "max": 1,
        "average": 0,
        "total": 11
    },
    "db.subTest.data.createHole": {
        "entries": 28,
        "last": 2,
        "min": 0,
        "max": 2,
        "average": 0,
        "total": 12
    },
    "db.subTest.data.findClosestHole": {
        "entries": 22,
        "last": 0,
        "min": 0,
        "max": 1,
        "average": 0,
        "total": 1
    },
    "db.subTest.data.move": {
        "entries": 12,
        "last": 0,
        "min": 0,
        "max": 1,
        "average": 0,
        "total": 4
    },
    "db.subTest.data.recycled.notFound": {
        "entries": 14,
        "last": 0,
        "min": 0,
        "max": 0,
        "average": 0,
        "total": 0
    },
    "db.subTest.data.recycled.partial": {
        "entries": 22,
        "last": 0,
        "min": 0,
        "max": 0,
        "average": 0,
        "total": 0
    },
    "db.subTest.data.updateHole": {
        "entries": 42,
        "last": 0,
        "min": 0,
        "max": 1,
        "average": 0,
        "total": 2
    },
    "db.subTest.delete": {
        "entries": 2,
        "last": 118,
        "min": 76,
        "max": 118,
        "average": 97,
        "total": 194
    },
    "db.subTest.metadata.load": {
        "entries": 6,
        "last": 0,
        "min": 0,
        "max": 1,
        "average": 0,
        "total": 1
    },
    "db.subTest.open": {
        "entries": 6,
        "last": 0,
        "min": 0,
        "max": 0,
                "average": 0,
                "total": 0
            },
            "db.subTest.readRecord": {
                "entries": 30,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 3
            },
            "db.subTest.updateRecord": {
                "entries": 36,
                "last": 2,
                "min": 0,
                "max": 2,
                "average": 0,
                "total": 16
            },
            "db.temp.createRecord": {
                "entries": 10,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 2
            },
            "db.temp.readRecord": {
                "entries": 7,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 1
            },
            "db.temp.updateRecord": {
                "entries": 21,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 2
            },
            "process.file.mmap.commitPages": {
                "entries": 2034,
                "last": 1,
                "min": 0,
                "max": 21,
                "average": 0,
                "total": 1048
            },
            "process.mvrbtree.clear": {
                "entries": 16007,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 141
            },
            "process.mvrbtree.commitChanges": {
                "entries": 165235,
                "last": 0,
                "min": 0,
                "max": 55,
                "average": 0,
                "total": 5730
            },
            "process.mvrbtree.entry.fromStream": {
                "entries": 5408,
                "last": 0,
                "min": 0,
                "max": 1,
                "average": 0,
                "total": 45
            },
            "process.mvrbtree.entry.toStream": {
```

```
            "entries": 60839,
            "last": 0,
            "min": 0,
            "max": 26,
            "average": 0,
            "total": 3013
        },
        "process.mvrbtree.fromStream": {
            "entries": 7424,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 54
        },
        "process.mvrbtree.get": {
            "entries": 97863,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 233
        },
        "process.mvrbtree.put": {
            "entries": 151070,
            "last": 0,
            "min": 0,
            "max": 55,
            "average": 0,
            "total": 5002
        },
        "process.mvrbtree.putAll": {
            "entries": 1847,
            "last": 0,
            "min": 0,
            "max": 8,
            "average": 0,
            "total": 84
        },
        "process.mvrbtree.remove": {
            "entries": 41000,
            "last": 0,
            "min": 0,
            "max": 10,
            "average": 0,
            "total": 2226
        },
        "process.mvrbtree.toStream": {
            "entries": 124870,
            "last": 0,
            "min": 0,
            "max": 6,
            "average": 0,
            "total": 543
        },
        "process.mvrbtree.unload": {
            "entries": 7424,
            "last": 0,
            "min": 0,
            "max": 10,
            "average": 0,
            "total": 519
        },
        "process.serializer.record.string.binary2string": {
            "entries": 1867,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 18
        },
        "process.serializer.record.string.bool2string": {
            "entries": 43,
            "last": 0,
            "min": 0,
            "max": 0,
            "average": 0,
            "total": 0
        },
        "process.serializer.record.string.byte2string": {
            "entries": 1143,
            "last": 0,
            "min": 0,
            "max": 0,
            "average": 0,
            "total": 0
        },
        "process.serializer.record.string.date2string": {
            "entries": 114176,
            "last": 0,
            "min": 0,
            "max": 6,
            "average": 0,
            "total": 464
        },
        "process.serializer.record.string.datetime2string": {
            "entries": 2,
            "last": 0,
            "min": 0,
            "max": 0,
            "average": 0,
            "total": 0
        },
        "process.serializer.record.string.decimal2string": {
            "entries": 2,
            "last": 1,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 1
        },
        "process.serializer.record.string.double2string": {
            "entries": 30237,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 104
        },
        "process.serializer.record.string.embed2string": {
            "entries": 122581,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 117
        },
        "process.serializer.record.string.embedList2string": {
            "entries": 29922,
            "last": 0,
            "min": 0,
            "max": 2,
            "average": 0,
            "total": 87
        },
        "process.serializer.record.string.embedMap2string": {
            "entries": 3160,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 25
        },
        "process.serializer.record.string.embedSet2string": {
            "entries": 32280,
            "last": 1,
            "min": 0,
            "max": 8,
            "average": 0,
            "total": 1430
        },
        "process.serializer.record.string.float2string": {
            "entries": 20640,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 63
        },
        "process.serializer.record.string.fromStream": {
            "entries": 1735665,
            "last": 0,
            "min": 0,
            "max": 82,
            "average": 0,
            "total": 7174
        },
        "process.serializer.record.string.int2string": {
            "entries": 246700,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 101
        },
        "process.serializer.record.string.link2string": {
            "entries": 18664,
            "last": 0,
            "min": 0,
            "max": 6,
            "average": 0,
            "total": 62
        },
        "process.serializer.record.string.linkList2string": {
            "entries": 2648,
            "last": 0,
            "min": 0,
            "max": 2,
            "average": 0,
            "total": 52
        },
        "process.serializer.record.string.linkMap2string": {
            "entries": 28,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 1
        },
        "process.serializer.record.string.linkSet2string": {
            "entries": 1269,
            "last": 0,
            "min": 0,
            "max": 33,
            "average": 0,
            "total": 80
        },
        "process.serializer.record.string.long2string": {
            "entries": 1620,
            "last": 0,
            "min": 0,
            "max": 1,
            "average": 0,
            "total": 6
        },
        "process.serializer.record.string.string2string": {
            "entries": 358585,
            "last": 0,
            "min": 0,
            "max": 3,
            "average": 0,
            "total": 183
        },
        "process.serializer.record.string.toStream": {
            "entries": 183912,
            "last": 0,
            "min": 0,
            "max": 34,
                "average": 0,
                "total": 3149
            },
            "server.http.0:0:0:0:0:0:0:1.request": {
                "entries": 2,
                "last": 2,
                "min": 2,
                "max": 19,
                "average": 10,
                "total": 21
            }
        },
        "statistics": {},
        "counters": {
            "db.0$db.cache.level2.cache.found": 7,
            "db.0$db.cache.level2.cache.notFound": 8,
            "db.0$db.data.update.notReused": 11,
            "db.0$db.data.update.reusedAll": 7,
            "db.1$db.cache.level2.cache.found": 7,
            "db.1$db.cache.level2.cache.notFound": 8,
            "db.1$db.data.update.notReused": 11,
            "db.1$db.data.update.reusedAll": 7,
            "db.2$db.cache.level2.cache.found": 7,
            "db.2$db.cache.level2.cache.notFound": 8,
            "db.2$db.data.update.notReused": 11,
            "db.2$db.data.update.reusedAll": 7,
            "db.demo.cache.level2.cache.found": 364467,
            "db.demo.cache.level2.cache.notFound": 393509,
            "db.demo.data.update.notReused": 38426,
            "db.demo.data.update.reusedAll": 140921,
            "db.demo.data.update.reusedPartial": 100,
            "db.demo.query.compositeIndexUsed": 46,
            "db.demo.query.compositeIndexUsed.2": 42,
            "db.demo.query.compositeIndexUsed.2.1": 20,
            "db.demo.query.compositeIndexUsed.2.2": 18,
            "db.demo.query.compositeIndexUsed.3": 4,
            "db.demo.query.compositeIndexUsed.3.1": 1,
            "db.demo.query.compositeIndexUsed.3.2": 1,
            "db.demo.query.compositeIndexUsed.3.3": 2,
            "db.demo.query.indexUsed": 2784,
            "db.subTest.cache.level2.cache.found": 14,
            "db.subTest.cache.level2.cache.notFound": 16,
            "db.subTest.data.update.notReused": 22,
            "db.subTest.data.update.reusedAll": 14,
            "db.temp.cache.level2.cache.found": 5,
            "db.temp.cache.level2.cache.notFound": 4,
            "process.file.mmap.pagesCommitted": 2034,
            "process.mvrbtree.entry.serializeKey": 4617509,
            "process.mvrbtree.entry.serializeValue": 68620,
            "process.mvrbtree.entry.unserializeKey": 6127,
            "process.mvrbtree.entry.unserializeValue": 225,
            "process.serializer.record.string.linkList2string.cached": 19,
            "server.http.0:0:0:0:0:0:0:1.requests": 3,
            "server.http.0:0:0:0:0:0:0:1.timeout": 1
        }
    }
}
```

Memory Leak Detector

OrientDB uses off-heap memory pool for its file cache allocations. Leaks of such allocations can't be tracked using standard Java techniques like heap dumps, to overcome this problem we developed a specialized leak detector. For sure, OrientDB is designed to avoid any kind of leaks, but sometimes bad things happen.

> Use the leak detector for debugging and troubleshooting purposes only, since it significantly slowdowns OrientDB off-heap memory management infrastructure. Never leave the leak detector turned on in a production setup.

Activating Leak Detector

To activate the leak detection provide the `memory.directMemory.trackMode=true` configuration option to the OrientDB instance in question. For example, you may provide following command line argument to the JVM:

```
java ... -Dmemory.directMemory.trackMode=true ...
```

Leak detector may be turned on or off at startup time only, at runtime changing the `memory.directMemory.trackMode` setting will have no effect.

Activating Debugging Logger

Leak detector uses logging facilities to report detected problems. To be sure you see all produced log entries activate the OrientDB debugging logger. Provide following command line argument to the JVM to activate it:

```
-Djava.util.logging.manager=com.orientechnologies.common.log.OLogManager$DebugLogManager
```

> Make sure `$DebugLogManager` part is not interpreted as a shell variable substitution. To avoid the substitution apply escaping specific to your shell environment. Read more about debugging logger here.

Inspecting for Leaks

After activation of both the leak detector and the debugging logger, information about found problems will be written to the log. Related log entries are marked with the `DIRECT-TRACK` label. Leak detector is able to detect following problems:

1. Attempt to release an off-heap memory buffer more than once. Reported instantly.
2. Presence of suspicious memory buffers released due to the JVM object finalization procedure. Reported during OrientDB lifetime, but not instantly.
3. Presence of unreleased/leaked memory buffers. Reported at OrientDB runtime shutdown.

Summary

The leak detection procedure looks as follows:

1. Shutdown the OrientDB instance in question, if it's running.
2. Activate the leak detector and the debugging logger as described above.
3. Start the OrientDB instance.
4. Start a workload cycle specific to your application.
5. Monitor for leaks using the log.
6. Wait for the workload cycle to finish.
7. Stop the OrientDB instance.
8. Inspect the OrientDB log for leaks detected at shutdown.

Distributed Configuration Tuning

When you run distributed on multiple servers, you could face on a drop of performance you got with single node. While it's normal that replication has a cost, there are many ways to improve performance on distributed configuration:

- Use transactions
- Replication vs Sharding
- Use few MASTER and many REPLICA servers
- Scale up on writes
- Scale up on reads
- Replication vs Sharding

Generic advice

Load Balancing

Active the load balancing to distribute the load across multiple nodes.

Use transactions

Even though when you update graphs you should always work in transactions, OrientDB allows also to work outside of them. Common cases are read-only queries or massive and non concurrent operations can be restored in case of failure. When you run on distributed configuration, using transactions helps to reduce latency. This is because the distributed operation happens only at commit time. Distributing one big operation is much efficient than transferring small multiple operations, because the latency.

Replication vs Sharding

OrientDB distributed configuration is set to full replication. Having multiple nodes with the very same copy of database is important for HA and scale reads. In facts, each server is independent on executing reads and queries. If you have 10 server nodes, the read throughput is 10x.

With writes it's the opposite: having multiple nodes with full replication slows down operations if the replication is synchronous. In this case Sharding the database across multiple nodes allows you to scale up writes, because only a subset of nodes are involved on write. Furthermore you could have a database bigger than one server node HD.

Use few MASTER and many REPLICA servers

Starting from v2.2, the biggest advantage of having many REPLICA servers is that they don't concur in the `writeQuorum`, so if you have 3 MASTER servers and 100 REPLICA servers, every write operation will be replicated across 103 servers, but the majority of the writeQuorum would be just 2, because given N/2+1, N is the number of MASTER servers. In this case after the operation is executed locally, the server coordinator of the write operation has to wait only for one more MASTER server.

For more information look at Server roles.

Scale up on writes

If you have a slow network and you have a synchronous (default) replication, you could pay the cost of latency. In facts when OrientDB runs synchronously, it waits at least for the `writeQuorum`. This means that if the `writeQuorum` is 3 ("majority"), and you have 5 nodes, the coordinator server node (where the distributed operation is started) has to wait for the answer from at least 3 nodes in order to provide the answer to the client.

In order to maintain the consistency, the `writeQuorum` should be set to the majority (the default setting), defined as N/2+1, where N is the number of MASTER servers. If you have 5 nodes the majority is 3. With 4 nodes is still 3. Setting the `writeQuorum` to 3 instead of 4 or 5 allows to reduce the latency cost and still maintain the consistency.

Scale up on reads

If you already set the `writeQuorum` to the majority to the nodes, you can leave the `readQuorum` to 1 (the default). This speeds up all the reads.

Security

OrientDB is the NoSQL implementation with the greatest focus on security.

- To connect to an existing database, you need a user and password. Users and roles are defined inside the database. For more information on this process, see Database Security.
- In the event that you're connecting to the OrientDB Server that is hosting the database, you can access the database using the server's user. For more information on this process, see Server Security.
- Additionally, you can encrypt the database contents on disk. For more information on this process, see Database Encryption.

❗	While OrientDB Server can function as a regular Web Server, it is not recommended that you expose it directly to either the Internet or public networks. Instead, always hide OrientDB server in private networks.

See also:

- New Security Features
- Database security
- Server security
- Database Encryption
- Secure SSL connections
- OrientDB Web Server

Database Security

OrientDB uses a security model based on well-known concepts of users and roles. That is, a database has its own users. Each User has one or more roles. Roles are a combination of the working mode and a set of permissions.

> For more information on security, see:
>
> - Server security
> - Database Encryption
> - Secure SSL connections
> - Record Level Security

Users

A user is an actor on the database. When you open a database, you need to specify the user name and the password to use. Each user has its own credentials and permissions.

By convention, each time you create a new database OrientDB creates three default users. The passwords for these users are the same as the usernames. That is, by default the `admin` user has a password of `admin`.

- `admin` This user has access to all functions on the database without limitation.
- `reader` This user is a read-only user. The `reader` can query any records in the database, but can't modify or delete them. It has no access to internal information, such as the users and roles themselves.
- `writer` This user is the same as the user `reader`, but it can also create, update and delete records.

The users themselves are records stored inside the cluster `ouser`. OrientDB stores passwords in hash. From version 2.2 on, OrientDB uses the PBKDF2 algorithm. Prior releases relied on SHA-256. For more information on passwords, see Password Management.

OrientDB stores the user status in the field `status`. It can either be `SUSPENDED` or `ACTIVE`. Only `ACTIVE` users can log in.

Working with Users

When you are connected to a database, you can query the current users on the database by using `SELECT` queries on the `OUser` class.

```
orientdb> SELECT RID, name, status FROM OUser

---+--------+--------+--------
#  | @CLASS | name   | status
---+--------+--------+--------
0  | null   | admin  | ACTIVE
1  | null   | reader | ACTIVE
2  | null   | writer | ACTIVE
---+--------+--------+--------
3 item(s) found. Query executed in 0.005 sec(s).
```

Creating a New User

To create a new user, use the `INSERT` command. Remember in doing so, that you must set the status to `ACTIVE` and give it a valid role.

```
orientdb> INSERT INTO OUser SET name = 'jay', password = 'JaY', status = 'ACTIVE',
          roles = (SELECT FROM ORole WHERE name = 'reader')
```

Updating Users

You can change the name for the user with the `UPDATE` statement:

```
orientdb> UPDATE OUser SET name = 'jay' WHERE name = 'reader'
```

In the same way, you can also change the password for the user:

```
orientdb> UPDATE OUser SET password = 'hello' WHERE name = 'reader'
```

OrientDB saves the password in a hash format. The trigger `OUserTrigger` encrypts the password transparently before it saves the record.

Disabling Users

To disable a user, use `UPDATE` to switch its status from `ACTIVE` to `SUSPENDED`. For instance, if you wanted to disable all users except for `admin`:

```
orientdb> UPDATE OUser SET status = 'SUSPENDED' WHERE name <> 'admin'
```

> **NOTE**: In the event that, due to accident or database corruption, you lose the user `admin` and need to restore it on the database, see Restoring the admin User`.

Roles

A role determines what operations a user can perform against a resource. Mainly, this decision depends on the working mode and the rules. The rules themselves work differently, depending on the working mode.

Working with Roles

When you are connected to a database, you can query the current roles on the database using `SELECT` queries on the `ORole` class.

```
orientdb> SELECT RID, mode, name, rules FROM ORole

--+------+----+--------+-------------------------------------------------------
# |@CLASS|mode| name   | rules
--+------+----+--------+-------------------------------------------------------
0 | null | 1  | admin  | {database.bypassRestricted=15}
1 | null | 0  | reader | {database.cluster.internal=2, database.cluster.orole=0...
2 | null | 0  | writer | {database.cluster.internal=2, database.cluster.orole=0...
--+------+----+--------+-------------------------------------------------------
3 item(s) found.  Query executed in 0.002 sec(s).
```

Creating New Roles

To create a new role, use the `INSERT` statement.

```
orientdb> INSERT INTO ORole SET name = 'developer', mode = 0
```

Role Inheritance

Roles can inherit permissions from other roles in an object-oriented fashion. To let a role extend another, add the parent role in the `inheritedRole` attribute. For instance, say you want users with the role `appuser` to inherit settings from the role `writer`.

```
orientdb> UPDATE ORole SET inheritedRole = (SELECT FROM ORole WHERE name = 'writer')
          WHERE name = 'appuser'
```

Working with Modes

Where rules determine what users belonging to certain roles can do on the databases, working modes determine how OrientDB interprets these rules. There are two types of working modes, designating by `1` and `0`.

- **Allow All But (Rules)** By default is the super user mode. Specify exceptions to this using the rules. If OrientDB finds no rules for a requested resource, then it allows the user to execute the operation. Use this mode mainly for power users and administrators. The default role `admin` uses this mode by default and has no exception rules. It is written as `1` in the database.
- **Deny All But (Rules)** By default this mode allows nothing. Specify exceptions to this using the rules. If OrientDB finds rules for a requested resource, then it allows the user to execute the operation. Use this mode as the default for all classic users. The default roles `reader` and `writer` use this mode. It is written as `0` in the database.

Operations

The supported operations are the classic CRUD operations. That is, **C**reate, **R**ead, **U**pdate, **D**elete. Roles can have none of these permissions or all of them. OrientDB represents each permission internally by a 4-digit bitmask flag.

```
NONE:   #0000 - 0
CREATE: #0001 - 1
READ:   #0010 - 2
UPDATE: #0100 - 4
DELETE: #1000 - 8
ALL:    #1111 - 15
```

In addition to these base permissions, you can also combine them to create new permissions. For instance, say you want to allow only the Read and Update permissions:

```
READ:               #0010 - 2
UPDATE:             #0100 - 4
Permission to use:  #0110 - 6
```

Resources

Resources are strings bound to OrientDB concepts.

> **NOTE**: Resource entries are case-sensitive.

- `database` , checked on accessing to the database
- `database.class.<class-name>` , checked on accessing on specific class
- `database.cluster.<cluster-name>` , checked on accessing on specific cluster
- `database.query` , checked on query execution
- `database.command` , checked on command execution
- `database.schema` , checked to access to the schema
- `database.function` , checked on function execution
- `database.config` , checked on accessing at database configuration
- `database.hook.record`
- `server.admin` , checked on accessing to remote server administration

For instance, say you have a role `motorcyclist` that you want to have access to all classes except for the class `Car` .

```
orientdb> UPDATE ORole PUT rules = "database.class.*", 15 WHERE name = "motorcyclist"

orientdb> UPDATE ORole PUT rules = "database.class.Car", 0 WHERE name = "motorcyclist"
```

Granting and Revoking Permissions

To grant and revoke permissions from a role, use the GRANT and REVOKE commands.

```
orientdb> GRANT UPDATE ON database.cluster.Car TO motorcyclist
```

Record-level Security

The sections above manage security in a vertical fashion at the schema-level, but in OrientDB you can also manage security in a horizontal fashion, that is: per record. This allows you to completely separate database records as sandboxes, where only authorized users can access restricted records.

To active record-level security, create classes that extend the `ORestricted` super class. In the event that you are working with a Graph Database, set the `V` and `E` classes (that is, the vertex and edge classes) themselves to extend `ORestricted`.

```
orientdb> ALTER CLASS V SUPERCLASS ORestricted

orientdb> ALTER CLASS E SUPERCLASS ORestricted
```

This causes all vertices and edges to inherit the record-level security. Beginning with version 2.1, OrientDB allows you to use multiple inheritances, to cause only certain vertex or edge calsses to be restricted.

```
orientdb> CREATE CLASS Order EXTENDS V, ORestricted
```

Whenever a class extends the class `ORestricted`, OrientDB uses special fields to type-set `_<OIdentifiable>` to store authorization on each record.

- `_allow` Contains the users that have full access to the record, (that is, all CRUD operations).
- `_allowRead` Contains the users that can read the record.
- `_allowUpdate` Contains the users that can update the record.
- `_allowDelete` Contains the users that can delete the record.

To allow full control over a record to a user, add the user's RID to the `_allow` set. To provide only read permissions, use `_allowRead`. In the example below, you allow the user with the RID `#5:10` to read record `#43:22`:

```
orientdb> UPDATE #43:22 ADD _allowRead #5:10
```

If you want to remove read permissions, use the following command:

```
orientdb> UPDATE #43:22 REMOVE _allowRead #5:10
```

Run-time Checks

OrientDB checks record-level security using a hook that injects the check before each CRUD operation:

- **Create Documents**: Sets the current database's user in the `_allow` field. To change this behavior, see Customize on Creation.
- **Read Documents**: Checks if the current user, or its roles, are listed in the `_allow` or `_allowRead` fields. If not, OrientDB skips the record. This allows each query to work per user.
- **Update Documents**: Checks if the current user, or its roles, are listed in the `_allow` or `_allowUpdate` field. If not, OrientDB raises an `OSecurityException` exception.
- **Delete Documents**: Checks if the current user, or its roles, are listed in the `_allow` or `_allowDelete` field. If not, OrientDB raises an `OSecurityException` exception.

The allow fields, (that is, `_allow`, `_allowRead`, `_allowUpdate`, and `_allowDelete`) can contain instances of `OUser` and `ORole` records, as both classes extend `OIdentity`. Use the class `OUser` to allow single users and use the class `ORole` to allow all users that are a part of that role.

Using the API

In addition to managing record-level security features through the OrientDB console, you can also configure it through the Graph and Document API's.

- **Graph API**

  ```
  OrientVertex v = graph.addVertex("class:Invoice");
  v.setProperty("amount", 1234567);
  graph.getRawGraph().getMetadata().getSecurity().allowUser(
        v.getRecord(), ORestrictedOperation.ALLOW_READ, "report");
  v.save();
  ```

- **Document API**

  ```
  ODocument invoice = new ODocument("Invoice").field("amount", 1234567);
  database.getMetadata().getSecurity().allowUser(
        invoice, ORestrictedOperation.ALLOW_READ, "report");
  invoice.save();
  ```

Customize on Creation

By default, whenever you create a restricted record, (that is, create a class that extends the class `ORestricted`), OrientDB inserts the current user into the `_allow` field. You can change this using custom properties in the class schema:

- `onCreate.fields` Specifies the names of the fields it sets. By default, these are `_allow` , but you can also specify `_allowRead` , `_allowUpdate` , `_allowDelete` or a combination of them as an alternative. Use commas to separate multiple fields.
- `onCreate.identityType` Specifies whether to insert the user's object or its role (the first one). By default, it is set to `user` , but you can also set it to use its `role` .

For instance, say you wanted to prevent a user from deleting new posts:

```
orientdb> ALTER CLASS Post CUSTOM onCreate.fields=_allowRead,_allowUpdate
```

Consider another example, where you want to assign a role instead of a user to new instances of `Post` .

```
orientdb> ALTER CLASS Post CUSTOM onCreate.identityType=role
```

Bypassing Security Constraints

On occasion, you may need a role that can bypass restrictions, such as for backup or administrative operations. You can manage this through the special permission `database.bypassRestricted` , by changing its value to `READ` . By default, the role `admin` has this permission.

For security reasons, this permission is not inheritable. In the event that you need to assign it to other roles in your database, you need to set it on each role.

Using Security

Now that you have some familiarity with how security works in OrientDB, consider the use case of OrientDB serving as the database for a blog-like application. The blog is accessible through the web and you need to implement various security features to ensure that it works properly and does not grant its users access to restricted content.

To begin, the administrator connects to the database and creates the document class `Post` , which extends `ORestricted` . This ensures that users can only see their own entries in the blog and entries that are shared with them.

```
orientdb> CONNECT REMOTE:localhost/blog admin admin
orientdb> CREATE CLASS Post EXTENDS ORestricted

Class 'Post' created successfully.
```

The user Luke is registered in `OUser` as `luke`, with an `RID` of `#5:5`. He logs into the database and creates a new blog, which is an instance of the class `Post`.

```
orientdb> CONNECT REMOTE:localhost/blog luke lukepassword
orientdb> INSERT INTO Post SET title = "Yesterday in Italy"

Created document #18:0

orientdb> SELECT FROM Post

-------+--------+--------------------
 RID   | _allow | title
-------+--------+--------------------
 #18:0 | [#5:5] | Yesterday in Italy
-------+--------+--------------------
```

Independent of the users `admin` and `luke`, there is the user Steve. Steve is registers with `OUser` as `steve`, he has an RID of `#5:6`. Steve logs into OrientDB and also creates a new entry on the class `Post`:

```
orientdb> CONNECT REMOTE:localhost/blog steve steve
orientdb> INSERT INTO Post SET title = "My Nutella Cake!"

Created document #18:1

orientdb> SELECT FROM Post

-------+--------+------------------
 RID   | _allow | title
-------+--------+------------------
 #18:1 | [#5:6] | My Nutella Cake!
-------+--------+------------------
```

As you can see, the users Steve and Luke can only see the records that they have access to. Now, after some editorial work, Luke is satisfied with the state of his blog entry `Yesterday in Italy`. He is now ready to share it with others. From the database console, he can do so by adding the user Steve's RID to the `_allow` field.

```
orientdb> UPDATE #18:0 ADD _allow = #5:6
```

Now, when Steve logs in, the same query from before gives him different results, since he can now see the content Luke shared with him.

```
orientdb> SELECT FROM Post

-------+--------+--------------------
 RID   | _allow | title
-------+--------+--------------------
 #18:0 | [#5:5] | Yesterday in Italy
 #18:1 | [#5:6] | My Nutella Cake!
-------+--------+--------------------
```

While this is an effective solution, it does have one minor flaw for Luke. By adding Steve to the `_allow` list, Steve can not only read posts Luke makes, but he can also modify them. While Luke may find Steve a reasonable person, he begins to have second thoughts about this blanket permission and decides to remove Steve from the `_allow` field and instead add him to the `_allowRead` field:

```
orientdb> UPDATE #18:0 REMOVE _allow = 5:6
orientdb> UPDATE #18:0 ADD _allowRead = #5:6
```

For the sake of argument, assume that Luke's misgivings about Steve have some foundation. Steve decides that he does not like Luke's entry `Yesterday in Italy` and would like to remove it from the database. He logs into OrientDB, runs `SELECT` to find its RID, and attempts to `DELETE` the record:

```
orientdb> SELECT FROM Post

-------+--------+--------------------
 RID   | _allow | title
-------+--------+--------------------
 #18:0 | [#5:5] | Yesterday in Italy
 #18:1 | [#5:6] | My Nutella Cake!
-------+--------+--------------------

orientdb> DELETE FROM #18:0

!Error: Cannot delete record #18:0 because the access to the resource is restricted.
```

As you can see, OrientDB blocks the `DELETE` operation, given that the current user, Steve, does not have permission to do so on this resource.

Password Management

OrientDB stores user passwords in the `OUser` records using the PBKDF2 HASH algorithm with a 24-bit length Salt per user for a configurable number of iterations. By default, this number is 65,536 iterations. You can change this account through the `security.userPasswordSaltIterations` global configuration. Note that while a higher iteration count can slow down attacks, it also slows down the authentication process on legitimate OrientDB use.

In order to speed up password hashing, OrientDB uses a password cache, which it implements as an LRU with a maximum of five hundred entries. You can change this setting through the `security.userPasswordSaltCacheSize` global configuration. Giving this global configuration the value of `0` disables the cache.

> **NOTE**: In the event that attackers gain access to the Java virtual machine memory dump, he could access this map, which would give them access to all passwords. You can protect your database from this attack by disabling the in memory password cache.

Server Security

Individual OrientDB servers can manage multiple databases at a time and each database can have its own set of users. When using OrientDB through the HTTP protocol, the OrientDB server uses one realm per database.

❗	While OrientDB can function as a regular Web Server, it is not recommended that you expose it directly to the internet or to public networks. Instead, always hide the OrientDB server within a private network.

Server users are stored in the `config/orientdb-server-config.xml` configuration file, in the `<users>` element.

```
    <users>
        <user name="root" password="{PBKDF2WithHmacSHA256}55F3DEAE:DLJEJFDKY8:65536" resources="*" />
        <user name="guest" password="{PBKDF2WithHmacSHA256}B36E7993C961:C8C8B36F3:65536" resources="connect,server.listDatabases,server.dblist" />
    </users>
```

When the OrientDB server starts for the first time, it creates the user `root` automatically, by asking you to give the password in the terminal. In the event that you do not specify a password, OrientDB generates a random password. Beginning with version 2.2, OrientDB hashes the passwords using `PBKDF2WithHmacSHA256` algorithm if it's running on Java 8 or major, otherwise `PBKDF2WithHmacSHA1` with a configurable SALT.

For more information on security in Orientdb, see:

- Database security
- Database Encryption
- Secure SSL connections

Configuration

While the default users and passwords are fine while you are setting your system up, it would be inadvisable to leave them in production. To help restrict untrusted users from accessing the OrientDB server, add a new user and change the passwords in the `config/orientdb-server-config.xml` server configuration file.

To restrict unauthorized users from giving themselves privileges on the OrientDB server, disable write-access to the configuration file. To help prevent them from viewing passwords, disable read-access as well. Note that even if the passwords are hashed, there are many techniques available to crack the hash or otherwise guess the real password.

❗	It is strongly recommended that you allow read/write access to the entire `config` directory only to the user that starts the OrientDB server.

Managing Users

Beginning with version 2.2, the OrientDB console provides a series of commands for managing users:

- `LIST SERVER USERS`: Displays all users.
- `SET SERVER USER`: Creates or modifies a user.
- `DROP SERVER USER`: Drops a user.

Server Resources

Each user can declare which resources have access. The wildcard `*` grants access to any resource. By default, the user `root` has all privileges (`*`), so it can access all the managed databases and operations.

Resources	Permission to
`database.create`	Create a new database in the server
`database.drop`	Drop a database
`database.exists`	Check the existence of a database
`database.freeze`	Freeze the access to a database
`database.release`	Release a frozen database
`database.passthrough`	Allow access to all managed databases
`server.config.get`	Retrieve server's setting
`server.config.set`	Update server's setting
`server.connect`	Connect to a server
`server.info`	Retrieve server's information and statistics
`server.listDatabases`	Enlist available databases on the server
`server.listDatabases.system`	Include the OSystem database in the list of databases
`server.replication`	Execute a replication command from another server
`server.replication.start`	Start the replication of a database
`server.replication.stop`	Stop the replication of a database
`server.replication.config`	Update the replication configuration
`server.shutdown`	Shutdown a server

For example,

```
<user name="replicator" password="repl" resources="database.passthrough"/>
```

Securing Connections with SSL

Beginning with version 1.7, you can further improve security on your OrientDB server by securing connections with SSL. For more information on implementing this, see Using SSL.

Restoring the User admin

In the event that something happens and you drop the class `OUser` or the user `admin`, you can use the following procedure to restore the user to your database.

1. Ensure that the database is in the OrientDB server database directory, `$ORIENTDB_HOME/database/ folder`.
2. Launch the console or studio and log into the database with the user `root`.

```
$  $ORIENTDB_HOME/bin/console.sh

OrientDB console v.X.X.X (build 0) www.orientdb.com
Type 'HELP' to display all the commands supported.
Installing extensions for GREMLIN language v.X.X.X

orientdb> CONNECT remote:localhost/my_database root rootpassword
```

3. Check that the class `OUser` exists:

   ```
   orientdb> SELECT FROM OUser WHERE name = 'admin'
   ```

   - In the event that this command fails because the class `OUser` doesn't exist, create it:

     ```
     orientdb> CREATE CLASS OUser EXTENDS OIdentity
     ```

   - In the event that this command fails because the class `OIdentity doesn't exist, create it first:

     ```
     orinetdb> CREATE CLASS OIdentity
     ```

     Then repeat the above command, creating the class `OUser`

4. Check that the class `ORole` exists.

   ```
   orientdb> SELECT FROM ORole WHERE name = 'admin'
   ```

   - In the event that the class `ORole` doesn't exist, create it:

     ```
     orientdb> CREATE CLASS ORole EXTENDS OIdentity
     ```

5. In the event that the user or role `admin` doesn't exist, run the following commands:

   - In the event that the role `admin` doesn't exist, create it:

     ```
     orientdb> INSERT INTO ORole SET name = 'admin', mode = 1,
               rules = { "database.bypassrestricted": 15 }
     ```

   - In the event that the user `admin` doesn't exist, create it:

     ```
     orientdb> INSERT INTO OUser SET name = 'admin',
               password = 'my-admin_password', status = 'ACTIVE',
               rules = ( SELECT FROM ORole WHERE name = 'admin' )
     ```

The user `admin` is now active again on your database.

Restoring the Server's User root

1. Open the `config/orientdb-server-config.xml` file and remove the "root" user entry
2. Remove the tag `<isAfterFirstTime>true</isAfterFirstTime>` at the end of the file
3. On next restart of the server, a new root password is asked again

Database Encryption

Beginning with version 2.2, OrientDB can encrypt records on disk. This prevents unauthorized users from accessing database content or even from bypassing OrientDB security. OrientDB does not save the encryption key to the database. You must provide it at run-time. In the event that you lose the encryption key, the database, (or at least the parts of the database you have encrypted), you lose access to its content.

> **NOTE**: Encryption at rest is not supported on remote protocol yet. It can be used only with plocal.

Encryption works through the encryption interface. It acts at the cluster (collection) level. OrientDB supports two algorithms for encryption:

- `aes` algorithm, which uses AES
- `des` algorithm, which uses DES

The AES algorithm is preferable to DES, given that it's stronger.

Encryption in OrientDB operates at the database-level. You can have multiple databases, each with different encryption interfaces, running under the same server, (or, JVM, in the event that you run OrientDB embedded). That said, you can use global configurations to define the same encryption rules for all databases open in the same JVM. For instance, you can define rules through the Java API:

```
OGlobalConfiguration.STORAGE_ENCRYPTION_METHOD.setValue("aes");
OGlobalConfiguration.STORAGE_ENCRYPTION_KEY.setValue("T1JJRU5UREJfSVNfQ09PTA==");
```

You can enable this at startup by passing these settings as JVM arguments:

```
$ java ... -Dstorage.encryptionMethod=aes \
      -Dstorage.encryptionKey="T1JJRU5UREJfSVNfQ09PTA=="
```

For more information on security in OrientDB, see the following pages:

- Database security
- Server security
- Secure SSL connections

Creating Encrypted Databases

You can create an encrypted database using either the console or through the Java API. To create an encrypted database, use the `-encryption` option through the `CREATE DATABASE` command. However, before you do so, you must set the encryption key by defining the `storage.encryptionKey` value through the `CONFIG` command.

```
orientdb> CONFIG SET storage.encryptionKey T1JJRU5UREJfSVNfQ09PTA==
orientdb> CREATE DATABASE plocal:/tmp/db/encrypted-db admin my_admin_password
            plocal document -encryption=aes
```

To create an encrypted database through the Java API, define the encryption algorithm and then set the encryption key as database properties:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/tmp/db/encrypted");
db.setProperty(OGlobalConfiguration.STORAGE_ENCRYPTION_METHOD.getKey(), "aes");
db.setProperty(OGlobalConfiguration.STORAGE_ENCRYPTION_KEY.getKey(), "T1JJRU5UREJfSVNfQ09PTA==");
db.create();
```

Whether you use the console or the Java API, these commands encrypt the entire database on disk. OrientDB does not store the encryption key within the database. You must provide it at run-time.

Encrypting Clusters

In addition to the entire database, you can also only encrypt certain clusters on the database. To do so, set the encryption to the default of `nothing` when you create the database, then configure the encryption per cluster through the `ALTER CLUSTER` command.

To encrypt the cluster through the Java API, create the database, then alter the cluster to use encryption:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/tmp/db/encrypted");
db.setProperty(OGlobalConfiguration.STORAGE_ENCRYPTION_KEY.getKey(), "T1JJRU5UREJfSVNfQ09PTA==");
db.create();
db.command(new OCommandSQL("ALTER CLUSTER Salary encryption aes")).execute();
```

Bear in mind that the key remains the same for the entire database. You cannot use different keys per cluster. If you attempt to apply encryption or an encryption setting on a cluster that is not empty, it raises an error.

To accomplish the same through the console, set the encryption key through `storage.encryptionKey` then define the encryption algorithm for the cluster:

```
orientdb> CONFIG SET storage.encryptionKey T1JJRU5UREJfSVNfQ09PTA==
orientdb> ALTER CLUSTER Salary encryption aes
```

Opening Encrypted Databases

You can access an encrypted database through either the console or the Java API. To do so through the console, set the encryption key with `storage.encryptionKey` then open the database.

```
orientdb> CONFIG SET storage.encryptionKey T1JJRU5UREJfSVNfQ09PTA==
orientdb> CONNECT plocal:/tmp/db/encrypted-db admin my_admin_password
```

When opening through the Java API, given that the encryption settings are stored with the database, you do not need to define the encryption algorithm when you open the database, just the encryption key.

```
db.setProperty(OGlobalConfiguration.STORAGE_ENCRYPTION_KEY.getKey(), "T1JJRU5UREJfSVNfQ09PTA==");
db.open("admin", "my_admin_password");
```

In the event that you pass a null or invalid key when you open the database, OrientDB raises an `OSecurityException` exception.

SSL

Beginning with version 1.7, OrientDB provides support for securing its HTTP and BINARY protocols through SSL. For distributed SSL, see the HazelCast documentation.

For more information on securing OrientDB, see the following pages:

- Database security
- Server security
- Database Encryption

Setting up the Key and Trust Stores

In order to set up and manage certificates, OrientDB uses the Java Keytool. Using certificates signed by a Certificate Authority (CA) is beyond the scope of this tutorial. For more information on using the Java Keytool, see the Documentation.

To create key and trust stores that reference a self-signed certificate, use the following guide:

1. Using Keytool, create a certificate for the server:

   ```
   # keytool -genkey -alias server -keystore orientdb.ks \
        -keyalg RSA -keysize 2048 -validity 3650
   ```

2. Export the server certificate to share it with the client:

   ```
   # keytool -export -alias server -keystore orientdb.ks \
          -file orientdb.cert
   ```

3. Create a certificate/keystore for the console/clients:

   ```
   # keytool -genkey -alias console -keystore orientdb-console.ks \
          -keyalg RSA -keysize 2048 -validity 3650
   ```

4. Create a trust-store for the client, then import the server certificate.

   ```
   # keytool -import -alias server -keystore orientdb-console.ts \
          -file orientdb.cert
   ```

   This establishes that the client trusts the server.

You now have a self-signed certificate to use with OrientDB. Bear in mind that for each remote client JVM you want to connect to the server, you need to repeat steps three and four. Remember to change the alias, keystore, and trust-store filenames accordingly.

Configuring OrientDB for SSL/TLS

Server Configuration

The server configuration file, `$ORIENTDB_HOME/config/orientdb-server-config.xml`, does not use TLS by default. To enable TLS on a protocol listener, you must change the `socket` attribute to the `<listener>` value from `default` to one of your configured `<socket>` definitions.

There are two default definitions available: `ssl` and `https`. (Note that `ssl` is used for the name but TLS is used internally.) For most use cases this is sufficient, however you can define more if you want to secure different listeners with their own certificates or would like to use a custom factory implementation. When using the `ssl` implementation, bear in mind that the default port for

OrientDB SSL is `2434`. You need to change your port range to `2434-2440`.

By default, the OrientDB server looks for its keys and trust-stores in `$ORIENTDB_HOME/config/cert`. You can configure it using the `<socket>` parameters. Be sure that all the key and trust-stores created in the previous setup are in the correct directory and that the passwords used are correct.

> **NOTE**: Paths are relative to `$ORIENTDB_HOME`. OrientDB also supports absolute paths.

```
<sockets>
  <socket implementation="com.orientechnologies.orient.server.network.OServerTLSSocketFactory" name="ssl">
    <parameters>
      <parameter value="false" name="network.ssl.clientAuth"/>
      <parameter value="config/cert/orientdb.ks" name="network.ssl.keyStore"/>
      <parameter value="password" name="network.ssl.keyStorePassword"/>
       <!-- NOTE: We are using the same store for keys and trust.
            This will change if client authentication is enabled. See Configuring Client section -->

      <parameter value="config/cert/orientdb.ks" name="network.ssl.trustStore"/>
      <parameter value="password" name="network.ssl.trustStorePassword"/>
    </parameters>
  </socket>

  ...

  <listener protocol="binary" ip-address="0.0.0.0" port-range="2424-2430" socket="default"/>
  <listener protocol="binary" ip-address="0.0.0.0" port-range="2434-2440" socket="ssl"/>
```

Console Configuration

For remote connections using the console, you need to make a few changes to to `console.sh`, enable SSL:

1. Confirm that your `KEYSTORE`, `TRUSTSTORE` and respective `PASSWORD` variables are correctly set.
2. In the `SSL_OPTS` definition, set `client.ssl.enabled` system property to `true`.

Client Configuration

To configure remote clients, use the standard Java system property patterns:

- `client.ssl.enabled`: Use this to enable/disable SSL. The property accepts `true` or `false`. You only need to define this when using remote binary client connections.
- `javax.net.ssl.keyStore`: Define the path to the keystore.
- `javax.net.ssl.keyStorePassword`: Defines the password to the keystore.
- `javax.net.ssl.trustStore`: Defines the path to the trust-store.
- `javax.net.ssl.trustStorePassword`: Defines the password to the trust-store.

Use the third and fourth steps from Setting up the Key and Trust Stores section above to create the client certificates and server trust. The paths to the stores are client specific but do not need to be the same as the server.

Note, if you would like to use key and/or trust-stores other than that of the default JVM, you need to define the following variables as well:

- `client.ssl.keyStore`: Defines the path to the keystore.
- `client.ssl.keyStorePass`: Defines the keystore password.
- `client.ssl.trustStore`: Defines the path to the trust-store.
- `client.ssl.trustStorePass`: Defines the password to the trust-store.

Consider the following example, configuring TLS from the command-line through Java:

```
$ java -Dclient.ssl.enabled=false \
      -Djavax.net.ssl.keyStore= \
      -Djavax.net.ssl.keyStorePassword= \
      -Djavax.net.ssl.trustStore= \
      -Djavax.net.ssl.trustStorePassword=
```

As an alternative, you can define these variables through the Java API:

```
System.setProperty("client.ssl.enabled", <"true"|"false">); # This will only be needed for remote binary clients
System.setProperty("javax.net.ssl.keyStore", </path/to/keystore>);
System.setProperty("javax.net.ssl.keyStorePassword", <keystorepass>);
System.setProperty("javax.net.ssl.trustStore", </path/to/truststore>);
System.setProperty("javax.net.ssl.trustStorePassword", <truststorepass>);
```

To verify or authenticate client certificates, you need to take a few additional steps on the server:

1. Export the client certificate, so that you can share it with the server:

   ```
   # keytool -export -alias  \
         -keystore  -file client_cert
   ```

   Alternatively, you can do this through the console:

   ```
   # keytool -export -alias console -keystore orientdb-console.ks \
         -file orientdb-console.cert
   ```

2. If you do not have a trust-store for the server, create one and import the client certificate. This establishes that the server trusts the client:

   ```
   # keytool -import -alias  -keystore orientdb.ts \
         -file client_cert
   ```

   Alternatively, you can manage the same through the console:

   ```
   # keytool -import -alias console -keystore orientdb.ts \
         -file orientdb-console.cert
   ```

In the server configuration file, ensure that you have client authentication enabled for the `<socket>` and that the trust-store path and password are correct:

```
  <sockets>
    <socket implementation="com.orientechnologies.orient.server.network.OServerSSLSocketFactory" name="ssl">
      <parameters>
        <parameter value="true" name="network.ssl.clientAuth"/>
        <parameter value="config/cert/orientdb.ks" name="network.ssl.keyStore"/>
        <parameter value="password" name="network.ssl.keyStorePassword"/>

        <!-- NOTE: We are using the trust store with the imported client cert. You can import as many client as you would like
 -->
        <parameter value="config/cert/orientdb.ts" name="network.ssl.trustStore"/>
        <parameter value="password" name="network.ssl.trustStorePassword"/>
      </parameters>
    </socket>
    ...
</sockets>
```

OrientDB Security Configuration

The new OrientDB security system uses a JSON configuration file that's located by default in the *config* directory. The default name of the file is *security.json*, but it can be overridden by setting the "server.security.file" property in *orientdb-server-config.xml* or by setting the global server property, "server.security.file".

The security.json configuration file may contain up to eight global properties: "enabled", "debug", "server", "authentication", "passwordValidator", "ldapImporter", and "auditing".

Here's a description of each top-level property in the security.json file.

Property	Description
"enabled"	If set to true (the default), the OrientDB security module that provides external authentication, authorization, password validation, LDAP import, and advanced auditing will be enabled.
"debug"	When set to true (false is the default), additional context information will be written to the OrientDB server log pertaining to the security system's operation.
"server"	This property is an object with one sub-property, "createDefaultUsers". It is described below.
"authentication"	This property is an object with two sub-properties, "allowDefault" and "authenticators", and it is described in greater detail below. Its purpose defines the available security authenticators.
"passwordValidator"	This property is also an object and defines the parameters required for the security system's password validator when enabled. It is described in detail below.
"ldapImporter"	This property is an object with up to five sub-properties, and it defines the LDAP importer object. See below for more details.
"auditing"	This property is also an object and contains two sub-properties, "class" and "enabled", and is described below.

"server"

The "server" object contains one property called "createDefaultUsers".

Property	Description
"createDefaultUsers"	When set to true (the default), the default OrientDB users will be created for new databases and the default orientdb-server-config.xml users will be created on the first run of the server.

"authentication"

The "authentication" object specifies security authenticators for OrientDB and their order. Here's an example:

```
"authentication" :
{
    "allowDefault" : true,

    "authenticators" :
    [
        {
            "name"        : "Kerberos",
            "class"        : "com.orientechnologies.security.kerberos.OKerberosAuthenticator",
            "enabled"    : true,
            "debug"        : false,

            "krb5_config"    : "/etc/krb5.conf",

            "service" :
            {
                "ktname"     : "/etc/keytab/kerberosuser",
                "principal"    : "kerberosuser/kerberos.domain.com@REALM.COM"
            },

            "spnego" :
            {
                "ktname"     : "/etc/keytab/kerberosuser",
                "principal"    : "HTTP/kerberos.domain.com@REALM.COM"
            },

            "client" :
            {
                "useTicketCache"    : true,
                "principal"         : "kerberosuser@REALM.COM",
                "renewalPeriod"        : 300
            }
        },

        {
            "name"                : "Password",
            "class"                : "com.orientechnologies.orient.server.security.authenticator.ODefaultPasswordAuthenticator
",
            "enabled"            : true,
            "users"             :
            [
                { "username" : "guest", "resources" : "connect,server.listDatabases,server.dblist" }
            ]
        },

        {
            "name"                : "ServerConfig",
            "class"                : "com.orientechnologies.orient.server.security.authenticator.OServerConfigAuthenticator",
            "enabled"            : true
        },

        {
            "name"                : "SystemAuthenticator",
            "class"                : "com.orientechnologies.orient.server.security.authenticator.OSystemUserAuthenticator",
            "enabled"            : true
        }    ]
}
```

Notice that "authentication" has two sub-properties, "allowDefault" and "authenticators". The "allowDefault" property, when set to true, tells the security system that if authentication fails for all the specified authenticators to allow OrientDB's default (built-in) authentication mechanism to also try.

The "authenticators" property is an array of authenticator objects. The order of the authenticators is significant, as the first is the primary authenticator. Subsequent authenticators in the list are called, if the first authenticator fails. This continues until either authentication succeeds or the list of authenticators is exhausted. If the "allowDefault" property is true and authentication is being called for a database, then OrientDB's native database security will then be attempted.

Each authenticator object supports at least three properties: "name", "class", and "enabled".

Property	Description
"name"	The "name" property must be unique among the authenticators and can be used by other security components to reference which authenticator is used to authenticate a service. As an example, the OLDAPImporter component may specify an "authenticator" to use and it must correspond to the "name" property.
"class"	The "class" property defines which authenticator component is instantiated for the security authenticator. The available authenticators are: ODefaultPasswordAuthenticator, OKerberosAuthenticator, OServerConfigAuthenticator, and OSystemUserAuthenticator. All are described below.
"enabled"	When set to true, the authenticator is used as part of the chain of authenticators. If set to false, the authenticator is ignored.

ODefaultPasswordAuthenticator

ODefaultPasswordAuthenticator supports an additional "users" property which contains an array of user objects. Each user object must contain "username" and "resource" properties. An optional "password" property is also permitted if authentication using a password is required.

Each user object can be used for authorization of the specified resources as well as authentication, if a password is present.

The full classpath for the "class" property is "com.orientechnologies.orient.server.security.authenticator.ODefaultPasswordAuthenticator".

Here's an example of the "users" property:

```
    "users"                :
    [
        { "username" : "someuser", "resources" : "*" }
    ]
```

The "resources" property uses the same format as the "resources" property for each `<user>` in the `<users>` section of the orientdb-server-config.xml file.

Additionally, *ODefaultPasswordAuthenticator* supports a "caseSensitive" property. It defaults to true. When set to false, usernames are not case-sensitive when retrieved for password or resources look-up.

OServerConfigAuthenticator

OServerConfigAuthenticator utilizes the element in the orientdb-server-config.xml file and permits its list of server users to be used for authentication and authorization of resources. Beyond "name", "class", and "enabled", *OServerConfigAuthenticator* requires no additional properties.

The full classpath for the "class" property is "com.orientechnologies.orient.server.security.authenticator.OServerConfigAuthenticator".

OServerConfigAuthenticator's "caseSensitive" property is always false, meaning that usernames are not case-sensitive when retrieved for password or resources look-up.

OSystemUserAuthenticator

OSystemUserAuthenticator implements authentication and authorization support for *system users* that reside in the OrientDB system database. Beyond "name", "class", and "enabled", *OSystemUserAuthenticator* requires no additional properties.

The full classpath for the "class" property is "com.orientechnologies.orient.server.security.authenticator.OSystemUserAuthenticator".

OSecuritySymmetricKeyAuth

OSecuritySymmetricKeyAuth implements support for symmetric key authentication for server users.

Here's an example:

```
"authenticators": [
    {
        "name": "SymmetricKey-Key",
        "class": "com.orientechnologies.agent.security.authenticator.OSecuritySymmetricKeyAuth",
        "enabled": true,
        "users": [
            {
                "username": "sysadmin",
                "resources": "*",
                "properties" :
                {
                    "key" : "8BC7LeGkFbmHEYNTz5GwDw==",
                    "keyAlgorithm" : "AES"
                }
            }
        ]
    }
]
```

The authenticator class is "com.orientechnologies.agent.security.authenticator.OSecuritySymmetricKeyAuth".

Each `user` has a "properties" field that can specify an actual key, a path pointing to a file with the key inside, or a KeyStore containing the key. Each is mutually exclusive.

"key"

The `key` property is a Base64-encoded string of a secret key. The `keyAlgorithm` property must also be used with `key` .

"keyFile"

The `keyFile` property is a path to a file containing a Base64-encoded secret key string. The `keyAlgorithm` property must also be used with `keyFile` .

Here's an example:

```
"properties" :
{
    "keyFile" : "${ORIENTDB_HOME}/config/AES.key",
    "keyAlgorithm" : "AES"
}
```

"keyStore"

The `keyStore` property is a JSON object that contains four sub-properties.

Here's an example of the `keyStore` property and its sub-properties.

"properties" : { "keyStore" : { "file" : "${ORIENTDB_HOME}/config/test.jks", "password" : "password", "keyAlias" : "keyAlias", "keyPassword" : "password" } }

"file"

The `file` property is a path to a Java KeyStore file that contains the secret key. Note that the `JCEKS` KeyStore format must be used to hold secret keys.

"password"

The `password` property holds the password for the KeyStore file. It is optional.

"keyAlias"

The `keyAlias` property is the alias name of the secret key in the KeyStore file. It is required.

"keyPassword"

The `keyPassword` property holds the password for the secret key in the KeyStore and is optional.

OSystemSymmetricKeyAuth

OSystemSymmetricKeyAuth implements support for symmetric key authentication for system users.

Here's an example:

```
"authenticators": [
    {
        "name": "OSystemSymmetricKey",
        "class": "com.orientechnologies.agent.security.authenticator.OSystemSymmetricKeyAuth",
        "enabled": true
    }
]
```

The authenticator class is "com.orientechnologies.agent.security.authenticator.OSystemSymmetricKeyAuth".

Each system user will have an embedded-JSON "properties" field that supports the same sub-properties as the `OSecuritySymmetricKeyAuth` authenticator.

OKerberosAuthenticator

OKerberosAuthenticator provides support for Kerberos/SPNEGO authentication. In addition to the usual "name", "class", and "enabled" properties, the *OKerberosAuthenticator* component also supports "debug", "krb5_config", "service", "spnego", and "client" properties. All of these properties are defined in greater detail below.

The full classpath for the "class" property is "com.orientechnologies.security.kerberos.OKerberosAuthenticator".

"debug"

When set to true, all of the Kerberos and SPNEGO authentication are displayed in the server's logfile. This can be very useful when debugging authentication problems.

"krb5_config"

By default, the KRB5_CONFIG environment variable is used to tell the Java Kerberos libraries the location of the krb5.conf file. However, the krb5_config property may be used to override this.

"service"

The "service" property contains two sub-properties, "ktname" and "principal".

Sub-Property	Description
"ktname"	This specifies the location of the keytab file used to obtain the key for decrypting OrientDB client service tickets. By default, the KRB5_KTNAME environment variable is used to determine the location of the server's keytab file. However, the ktname property may be used to override this.
"principal"	This specifies the LDAP/AD user that's associated with the OrientDB SPN. This is used to obtain the correct service key to decrypt service tickets from OrientDB clients. This is a mandatory property.

"spnego"

The "spnego" property contains two sub-properties, "ktname" and "principal".

Sub-Property	Description
"ktname"	This specifies the location of the keytab file used to obtain the key for decrypting OrientDB SPNEGO service tickets. By default (and the same as the "service" property), the KRB5_KTNAME environment variable is used to determine the location of the server's keytab file. However, the "ktname" property may be used to override this.
"principal"	This specifies the LDAP/AD user that's associated with the OrientDB SPN. This is used to obtain the correct service key to decrypt service tickets from OrientDB SPNEGO clients. This is a mandatory property.

"client"

The "client" property is used by OrientDB for configuring the LDAP/AD importer user, and it may contain five different sub-properties: "ccname", "ktname", "principal", "useTicketCache", and "renewalPeriod".

Sub-Property	Description
"ccname"	By default, the KRB5CCNAME environment variable is used to determine the location of the local credential cache file. However, the "ccname" property may be used to override this. This is only used if "useTicketCache" is set to true.
"ktname"	By default, the KRB5_CLIENT_KTNAME environment variable is used to determine the location of the server's client keytab file. However, the "ktname" property may be used to override this. This is only used if "useTicketCache" is set to false.
"principal"	This is the name of the principal used with the client credentials when accessing the LDAP service. It is a required property.
"useTicketCache"	Defaults to false. If true, the local ticket cache will be used to provide the client credentials to the LDAP service. The KRB5CCNAME environment variable is used to determine the location of the ticket cache if the "ccname" property is not specified. If false (or not present) then the specified keytab file (ktname) is used.
"renewalPeriod"	This is an optional property, and it specifies how often the LDAP client service ticket is renewed (in minutes). It defaults to 5 hours.

"passwordValidator"

The "passwordValidator" object specifies an optional password validator that the OrientDB security system uses when applying new passwords. The supported properties of the password validator object depend on the type of object specified but always contain at least "class" and "enabled". The "class" property defines which password validator component is instantiated. The "enabled" property (defaults to true) specifies whether the password validator component is active or not.

OrientDB ships with an *ODefaultPasswordValidator* component. Its properties are defined below, and each is only required if it's included in the "passwordValidator" property.

The full classpath for the "class" property is "com.orientechnologies.security.password.ODefaultPasswordValidator".

ODefaultPasswordValidator

Property	Description
"minimumLength"	This property defines the minimum number of characters required in the password.
"numberRegEx"	This property defines the regular expression for the minimum count of required numbers and what symbols are considered numbers.
"uppercaseRegEx"	This property defines the regular expression for the minimum count of required uppercase characters and what symbols are considered uppercase characters.
"specialRegEx"	This property defines the regular expression for the minimum count of special characters and what symbols are considered special characters.

Here is an example of the *ODefaultPasswordValidator*'s configuration in the security.json file:

```
"passwordValidator" :
{
    "class"               : "com.orientechnologies.security.password.ODefaultPasswordValidator",
    "minimumLength"       : 5,
    "numberRegEx"       : "(?:[0-9].*){2}",
    "uppercaseRegEx"    : "(?:[A-Z].*){3}",
    "specialRegEx"        : "(?:[^a-zA-Z0-9 ].*){2}"
}
```

"ldapImporter"

The "ldapImporter" object defines the properties for the LDAP importer security component. As with the other security components, the LDAP importer object always has a "class" property and optional "enabled" and "debug" properties. The "class" property defines which LDAP importer component is instantiated. The "enabled" property (defaults to true) specifies whether the LDAP importer component is active or not.

OrientDB provides a default OLDAPImporter component, and its properties are defined below. The full classpath for the "class" property is "com.orientechnologies.security.ldap.OLDAPImporter".

OLDAPImporter

Property	Description
"period"	This property is in seconds and determines how often the LDAP synchronization occurs.
"databases"	This property contains an array of objects. Each object represents an OrientDB database along with LDAP domains to import.

Database Object

Each database object contains three properties: "database", "ignoreLocal", and "domains". The "database" property is just the name of the OrientDB database. The "ignoreLocal" property is an optional boolean property that defaults to true. When true it indicates that existing users in the specified database will not be deleted if they are not present in the imported LDAP users list. The "domains" property is described below.

"domains"

The "domains" property contains an array of objects, each with "domain", "servers", and "users" properties, and an optional "authenticator" property.

Property	Description
"domain"	This property must be unique for each database object and is primarily used with the *_OLDAPUser* class within each OrientDB database.
"authenticator"	The "authenticator" property specifies which of the authenticators should be used to communicate with the LDAP server. If none is specified, then the primary authenticator is used.
"servers"	This property is an array of server objects, each specifying the URL of the LDAP server. It is described below.
"users"	This property is an array of user objects, each specifying the baseDN, filter, and roles to use for importing. It is described below.

"servers"

Property	Description
"url"	This property specifies the LDAP server's URL. (ldaps is supported.)
"isAlias"	This is a boolean property. When true, the hostname specified in the URL is treated as an alias, and the real address is queried followed by a reverse DNS lookup. This is useful for a hostname that is an alias for multiple LDAP servers. The "isAlias" property defaults to false and is not a mandatory property.

"users"

LDAP users are imported based on a starting location within the directory and filtered using a standard LDAP filter rule.

Property	Description
"baseDN"	The "basedDN" property specifies the distinguished name of the starting point for the LDAP import, e.g., "CN=Users,DC=ad,DC=domain,DC=com".
"filter"	This property specifies the LDAP filter to use for importing users. Here's a simple example: "(&(objectCategory=person)(objectclass=user)(memberOf=CN=ODBUser,CN=Users,DC=ad,DC=domain,DC=com))".
"roles"	This is an array of strings, specifying the corresponding OrientDB roles that will be assigned to each user that is imported from the current group.

As an alternative to the "users" property in the security configuration file, *OLDAPImporter* also supports using a class in the database, called *_OLDAPUser*, for importing LDAP users based on the same properties. See below for more details.

Example

Here's an example of the "ldapImporter" property:

```
"ldapImporter" :
{
    "class"        : "com.orientechnologies.security.ldap.OLDAPImporter",
    "enabled"    : true,
    "debug"        : false,
    "period"    : 60,
    "databases" :
    [
        {
            "database"        : "MyDB",
            "ignoreLocal"    : true,
            "domains"        :
            [
                {
                    "domain"        : "ad.domain.com",
                    "authenticator" : "Kerberos",

                    "servers"    :
                    [
                        {
                            "url"        : "ldap://alias.ad.domain.com:389",
                            "isAlias"    : true
                        }
                    ],

                    "users" :
                    [
                        {
                            "baseDN"    : "CN=Users,DC=ad,DC=domain,DC=com",
                            "filter"    : "(&(objectCategory=person)(objectclass=user)(memberOf=CN=ODBUser,CN=Users,DC=ad,DC=d
omain,DC=com))",
                            "roles"        : ["reader", "writer"]
                        },
                        {
                            "baseDN"    : "CN=Users,DC=ad,DC=domain,DC=com",
                            "filter"    : "(&(objectCategory=person)(objectclass=user)(memberOf=CN=ODBAdminGroup,CN=Users,DC=a
d,DC=domain,DC=com))",
                            "roles"        : ["admin"]
                        }
                    ]
                }
            ]
        }
    ]
}
```

_*OLDAPUser* Class

The OrientDB _*OLDAPUser* class can be used in place of or in addition to the "users" section of "ldapImporter".

The class has four properties defined: *Domain*, *BaseDN*, *Filter*, and *Roles*.

Property	Description
Domain	The *Domain* property must match the "domain" property of a domain object in the "domains" array of an "ldapImporter" database object.
BaseDN	This property is equivalent to the "baseDN" property of a user object in the "users" array.
Filter	This property is equivalent to the "filter" property of a user object in the "users" array.
Roles	The *Roles* property is equivalent to the "roles" array of a user object in the "users" array. However, the value of *Roles* is a single string, and each role is separated by a comma.

"auditing"

The *auditing* component of the new security system is configured with the "auditing" object. It has four possible properties, the usual "class" and "enabled" properties, and "distributed" and "systemImport" properties.

"distributed"

The *distributed* property is an object and is used to configure what node events are recorded in the auditing log.

The *distributed* object may contain these properties:

Property	Description
"onNodeJoinedEnabled"	If `true` , enables auditing of node joined events. The default is `false` .
"onNodeJoinedMessage"	This is a custom message stored in the `note` field of the auditing record on node joined events. It supports the dynamic binding of values, see *Customing the Message* below.
"onNodeLeftEnabled"	If `true` , enables auditing of node left events. The default is `false` .
"onNodeLeftMessage"	This is a custom message stored in the `note` field of the auditing record on node left events. It supports the dynamic binding of values, see *Customing the Message* below.

Customing the Message

The variable `${node}` will be substituted in the specified message, if node joined or node left auditing is enabled.

Example

Here's an example of a "distributed" section:

```
"auditing": {
  "class": "com.orientechnologies.security.auditing.ODefaultAuditing",
  "enabled": true,
  "distributed": {
    "onNodeJoinedEnabled": true,
    "onNodeJoinedMessage": "Node ${node} has joined...",
    "onNodeLeftEnabled": true,
    "onNodeLeftMessage": "Node ${node} has left..."
  }
}
```

"systemImport"

The *systemImport* property is an object and is used for importing a database's auditing log into the system database, where all new auditing is stored.

Each auditing log record from the specified databases is moved to a cluster in the new system database with the name *databasename*_auditing.

The *systemImport* object may contain these properties:

Property	Description	Default Value
"enabled"	When set to true, the audit log importer will run for the specified databases.	false
"databases"	This property is a list of database names that will have their auditing logs imported.	none
"auditingClass"	This specifies the name of the auditing class used in the database being imported.	*AuditingLog*
"limit"	The *limit* property indicates how many records will be imported during each *iteration* of the importer. To reduce the impact on the system, the importer retrieves a block of auditing log records for each *iteration* and then sleeps before beginning again.	1000
"sleepPeriod"	This represents (in milliseconds) how long the importer sleeps after each *iteration*.	1000

The importer stops for each database once all the auditing log records have been transferred to the system database.

Here's an example of a "systemImport" section:

```
"auditing": {
  "class": "com.orientechnologies.security.auditing.ODefaultAuditing",
  "enabled": true,
  "systemImport": {
    "enabled": true,
    "databases": ["MyDB", "MyOtherDB", "OneMoreDB"],
    "auditingClass": "AuditingLog",
    "limit": 1000,
    "sleepPeriod": 2000
  }
}
```

OrientDB Kerberos Client Examples

The Java API client is dependent on the *KRB5_CONFIG* and *KRB5CCNAME* environment variables being set. Alternatively, you can pass them to the Java program via `-Dclient.krb5.config=…` and `-Dclient.krb5.ccname=…` , respectively. You can also set them in your Java client program by calling `System.setProperty("client.krb5.config", "…./krb5.conf")` .

If a keytab is preferred, the *KRB5_CLIENT_KTNAME* environment variable may be set with the path to the keytab. Alternatively, you may pass the keytab path as the system property *client.krb5.ktname*.

What enables the Kerberos client support is a system property called *client.credentialinterceptor*. This must be set with the full package name to the Kerberos credential interceptor, as such:

`java -Dclient.credentialinterceptor=com.orientechnologies.orient.core.security.kerberos.OKerberosCredentialInterceptor`

or

`System.setProperty("client.credentialinterceptor", "com.orientechnologies.orient.core.security.kerberos.OKerberosCredentialInterceptor");`

In either case, this system property must be set to enable the Java client Kerberos support.

To use the client, you must specify the principal in the *username* field. The *password* field may also specify the SPN to use to retrieve the service ticket, but if left as an empty string, then the SPN will be generated from the host portion of the URL as such: "OrientDB/" + host.

The principal must either exist as a server user or must exist as an OUser record in the database to be accessed.

Java API Examples

The following is an example of how to use the OServerAdmin interface to send commands directly to the server:

```
String url = "remote:server1.ad.somedomain.com:2424";
String pri = "OrientDBClient@AD.SOMEDOMAIN.COM";
String spn = "OrientDB/db1.somedomain.com";

OServerAdmin serverAdmin = new OServerAdmin(url).connect(pri, spn);
serverAdmin.createDatabase("TestDB", "graph", "plocal");
serverAdmin.close();
```

The next example shows how to open an existing OrientDB database from the Java client API:

```
String url = "remote:server1.ad.somedomain.com:2424/TestDB";
String pri = "OrientDBClient@AD.SOMEDOMAIN.COM";
String spn = "OrientDB/db1.somedomain.com";

ODatabaseDocumentTx db = new ODatabaseDocumentTx(url).open(pri, spn);
```

JDBC Client

The JDBC client support is very similar to the native OrientDB Java client. Make sure the *KRB5_CONFIG* and *KRB5CCNAME* environment variables (or system properties) are set accordingly and then set the *client.credentialinterceptor* system property and specify the URL, principal, and SPN appropriately:

```
System.setProperty("client.credentialinterceptor", "com.orientechnologies.orient.core.security.kerberos.OKerberosCredentialInt
erceptor");

String url = "remote:server1.ad.somedomain.com:2424/TestDB";
String pri = "OrientDBClient@AD.SOMEDOMAIN.COM";
String spn = "OrientDB/db1.somedomain.com";

Class.forName("com.orientechnologies.orient.jdbc.OrientJdbcDriver");

Properties info = new Properties();
info.put("user", pri);
info.put("password", spn);

Connection conn = (OrientJdbcConnection) DriverManager.getConnection(url, info);

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("select from MyClass");
```

OrientDB Console

To enable Kerberos in the OrientDB console, you'll need to modify the console.sh (or console.bat) script.

Simply add the credential interceptor system property to ORIENTDB_SETTINGS as such:

```
ORIENTDB_SETTINGS="-Dclient.credentialinterceptor=com.orientechnologies.orient.core.security.kerberos.OKerberosCredentialInter
ceptor"
```

Here's an example of connecting to the previously used URL, principal, and SPN:

```
connect remote:server1.ad.somedomain.com:2424 OrientDBClient@AD.SOMEDOMAIN.COM OrientDB/db1.somedomain.com
```

Here's another example, this time creating a remote plocal database on remote server:

```
create database remote:server1.ad.somedomain.com:2424/NewDB OrientDBClient@AD.SOMEDOMAIN.COM OrientDB/db1.somedomain.com ploca
l
```

Lastly, this final example with the console shows connecting to the NewDB database that we just created:

```
connect remote:server1.ad.somedomain.com:2424/NewDB OrientDBClient@AD.SOMEDOMAIN.COM OrientDB/db1.somedomain.com
```

New Security Features - Code Changes

The motivation behind creating a new security layer for OrientDB was to meet the requirements of several customers who needed external authentication (specifically Kerberos and SPNEGO support), the ability to import users from specific groups via LDAP, password validation, and new auditing capabilities.

The OrientDB *Client*, *Core*, *Server*, and *Enterprise Agent* modules were all affected by the new security system. At the heart is a new interface called `OSecuritySystem` that provides most of the high-level methods required for implementing the main features. The `Orient` instance in the *Core* stores a reference to the `OSecuritySystem` instance, and it can be accessed by calling `Orient.instance().getSecurity().` Note that this only applies when called from the Server. Otherwise, `getSecurity()` will just return null.

Another main interface, called `OServerSecurity`, is derived from `OSecuritySystem`, and it provides some additional methods that are specific to the server. In `OServer`, an implementation of `OServerSecurity`, called `ODefaultServerSecurity`, is created and stored. It can be accessed via a public method, `getSecurity()`, from the server instance. When `ODefaultServerSecurity` is created in `OServer`, its instance is registered with `Orient`.

`ODefaultServerSecurity` handles loading the new JSON security configuration file that is stored under *config* and is called *security.json* by default. Currently, *security.json* has five main sections: "server", "authentication", "passwordValidator", "ldapImporter", and "auditing".

The most important thing to take away from this implementation is the concept of *authenticators* which can be chained together to support multiple kinds of authentication. As an example, the primary *authenticator* could be for Kerberos, followed by an NTLM *authenticator*, followed by a password *authenticator* (which allows usernames, passwords, and resources to be stored in the *security.json* file), and then followed by an `OServer` config *authenticator* (which supports the users section in *orientdb-server-config.xml*). If authentication fails for the first *authenticator* then the next one in the chain is attempted. This continues until all *authenticators* are tried.

Authorization works in a similar way. The primary *authenticator* implementation is tried first to see if the specified user has the required resource permissions. If not successful, the second *authenticator* is then tried, and so forth.

Lastly, for the type of HTTP authentication that each *authenticator* supports, an authentication header will be created using each *authenticator*, starting with the primary. As an example, this allows for a Kerberos "Negotiate" header to be returned as well as a "Basic" authentication header for fallback purposes.

On the *Client* side, there's a new interface, called `OCredentialInterceptor`, which allows for providing single sign-on security implementations (such as for Kerberos, NTLM) while still working with the OrientDB authentication framework.

The other significant change is in the *Core* and happens in `OMetadataDefault` in its `init()` method. Previously, an instance of `OSecurityShared` was created. Now, an `OSecurity` instance is created by calling `OSecurityManager.instance().newSecurity()`. The reason for this is to support different OSecurity implementations depending on the authentication requirements. As an example, for external authentication, an `OSecurityExternal` instance will be created. `OSecurityExternal` derives from `OSecurityShared` but overrides `authenticate()`, calling the security system's `authenticate()` method instead (`Orient.instance().getSecurity().authenticate()`).

Enterpise auditing has some new capabilities as well. Tracking of creating and dropping classes has been added as well as when nodes join and leave the cluster in a distributed setup. Writing to Syslog as part of the auditing log has been added too.

Client

OServerAdmin

Modified the `connect()` method to create a new `OCredentialInterceptor` from the security manager and, if successful, calls its `intercept()` method, and then retrieves the username and password from the interceptor.

```
OCredentialInterceptor ci = OSecurityManager.instance().newCredentialInterceptor();

if(ci != null)
{
  ci.intercept(storage.getURL(), iUserName, iUserPassword);
  username = ci.getUsername();
  password = ci.getPassword();
}
```

OStorageRemote

Modified the `open()` method to create a new `OCredentialInterceptor` from the security manager and, if successful, calls its `intercept()` method, and then retrieves the username and password from the interceptor.

```
OCredentialInterceptor ci = OSecurityManager.instance().newCredentialInterceptor();

if(ci != null)
{
  ci.intercept(getURL(), iUserName, iUserPassword);
  connectionUserName = ci.getUsername();
  connectionUserPassword = ci.getPassword();
}
else // Do Nothing
{
  connectionUserName = iUserName;
  connectionUserPassword = iUserPassword;
}
```

Core

Orient

Added getter and setter methods for the OSecuritySystem variable.

```
private volatile OSecuritySystem security;

public OSecuritySystem getSecurity() { return this.security; }
public void setSecurity(final OSecuritySystem security) {
  this.security = security;

}
```

OGlobalConfiguration

Added several new properties to support Kerberos on the client and other server-related security properties.

These properties are OrientDB client-specific only:

Property	Description
"client.krb5.config"	Specifies the location of the Kerberos configuration file (typically krb5.conf).
"client.krb5.ccname"	Specifies the location of the Kerberos credential cache file.
"client.krb5.ktname"	Specifies the location of the Kerberos keytab file.
"client.credentialinterceptor"	Specifies the name of the credential interceptor class to use.

These properties are OrientDB server-specific only:

Property	Description
"security.createDefaultUsers"	Indicates if "default users" should be created. When set to true and the server instance is new, the default users in orientdb-server-config.xml will be created. Also, if set to true, when a new database is created the default database users will be created.
"server.security.file"	Specifies the location (and name) of the JSON security configuration file. By default, the file is named *security.json* and is located in the *config* directory.

OMetadataDefault

In the `init()` method, notice that `final OSecurityShared instance = new OSecurityShared();`

```
new Callable<OSecurityShared>() {
 public OSecurityShared call() {
   final OSecurityShared instance = new OSecurityShared();
   if (iLoad) {
     security = instance;
     instance.load();
   }
   return instance;
 }
}), database);
```

is changed to `final OSecurity instance = OSecurityManager.instance().newSecurity();`

```
new Callable<OSecurity>() {
 public OSecurity call() {
   final OSecurity instance = OSecurityManager.instance().newSecurity();
   if (iLoad) {
     security = instance;
     instance.load();
   }
   return instance;
 }
}), database);
```

OSchemaShared

In the `dropClassInternal()` method, added calls to `onDropClass()` for all the *ODatabaseLifecycleListener* listeners.

```
// WAKE UP DB LIFECYCLE LISTENER
for (Iterator<ODatabaseLifecycleListener> it = Orient.instance().getDbLifecycleListeners(); it.hasNext(); )
  it.next().onDropClass(getDatabase(), cls);
```

OSecurityManager

Added four new methods to support the credential interceptor, security factory, and new OSecurity instances. Created a default OSecurityFactory instance.

```
private OSecurityFactory securityFactory = new OSecuritySharedFactory();

public OCredentialInterceptor newCredentialInterceptor()
    // Dynamically creates an OCredentialInterceptor instance using the OGlobalConfiguration.CLIENT_CREDENTIAL_INTERCEPTOR pro
perty.

public OSecurityFactory getSecurityFactory() { return securityFactory; }

public void setSecurityFactory(OSecurityFactory factory) { securityFactory = factory; }

public OSecurity newSecurity()
{
  if(securityFactory != null) return securityFactory.newSecurity();

  return null;
}
```

OSecurityShared

In the `isAllowed()` method, added a fix to check for if `iAllowAll` is null. Also added a check for `areDefaultUsersCreated()` when calling `createUser()` .

```
if(Orient.instance().getSecurity() == null || Orient.instance().getSecurity().areDefaultUsersCreated())
    createUser(…)
```

Also changed the protection of the `getDatabase()` method from private to protected.

OUserTrigger

In the `encodePassword()` method, added a call to the security system's `validatePassword()` method.

```
if(Orient.instance().getSecurity() != null)
{
  Orient.instance().getSecurity().validatePassword(password);
}
```

New Files - Added to orient/core/security

File	Description
OCredentialInterceptor.java	Provides a basic credential interceptor interface with three methods: getUsername(), getPassword(), and intercept().
OInvalidPasswordException.java	Provides a specific exception used by password validator implementations for failed validation. Derives from OSecurityException.
OPasswordValidator.java	Provides a simple interface for validating passwords. Contains one method: validatePassword().
OSecurityExternal.java	Extends OSecurityShared, providing external authentication by calling Orient.instance().getSecurity().authenticate().
OSecurityFactory.java	Provides an interface for creating new OSecurity instances. Contains one method: newSecurity().
OSecuritySharedFactory.java	Implements the OSecurityFactory interface for OSecurityShared instances. Its newSecurity() method returns a new OSecurityShared instance.
OSecuritySystem.java	Provides a basic interface for a modular security system supporting external authenticators.
OSecuritySystemException.java	Provides a specific exception used by OSecuritySystem implementations.

OSecuritySystem

OSecuritySystem is a base interface and requires the following methods:

Method	Description
authenticate()	Takes a username and password and returns the authenticated *principal* (on success) or null (on failure).
areDefaultUsersCreated()	Returns a boolean indicating if the "createDefaultUsers" security property is set to true or false. When set to true and the server instance is new, the default users in orientdb-server-config.xml will be created. Also, if set to true, when a new database is created the default database users will be created.
getAuthenticationHeader()	Returns the current HTTP authentication header. Depending on which *authenticator(s)* is installed, multiple authentication headers may be returned to support primary, secondary, and fallback browser authentication types.
isAuthorized()	Takes a username and resource list (as a String). Walks through the installed *authenticators* and returns true if a matching username is found that has the required authorization for the specified resource list.
isDefaultAllowed()	Returns a boolean indicating if the "authentication"."allowDefault" property is set to true or false. When set to true and authentication fails using the installed *authenticators*, the default database authentication may also be used.
isEnabled()	Returns a boolean indicating if the security system is enabled.
isSingleSignOnSupported()	Returns a boolean that indicates if the primary authenticator supports single sign-on.
validatePassword()	Takes a password and, if a password validator is installed, throws an OInvalidPasswordException if the provided password fails to meet the validation criteria.

New Directory (kerberos) - Added to orient/core/security

Added a new kerberos directory under orient/core/security.

New Files - Added to orient/core/security/kerberos

File	Description
OKerberosCredentialInterceptor.java	Provides an implementation of OCredentialInterceptor, specific to Kerberos.
OKrb5ClientLoginModuleConfig.java	Provides a custom Kerberos client login `Configuration` implementation.

Server

OServer

Added a new variable for an `OServerSecurity` instance along with getter/setter methods.

Created a new method called `authenticateUser()`. If the `OServerSecurity` instance is enabled, it calls its `authenticate()` method; and if a valid *principal* is returned, then its `isAuthorized()` method is called. If `OServerSecurity` is not enabled, then the default `OServer` authentication and authorization methods are used.

Modified the `activate()` method, adding the creation of a new `ODefaultServerSecurity` instance, assigning it to the `serverSecurity` variable, and calling `Orient.instance.setSecurity(serverSecurity)`.

Changed the `serverLogin()` method, calling `authenticateUser()` instead.

Modified the `authenticate()` method, also calling `authenticateUser()` instead.

Updated the `isAllowed()` method, calling the server security's `isAuthorized()` method, if the security module is enabled. Otherwise, the default implementation is used.

Changed the `getUser()` method, calling the server security's `getUser()` method, if the security module is enabled. Otherwise, the default implementation is used.

Modified the `openDatabase()` method, checking and using the returned value of `serverLogin()`. This is done because in some security implementations the user is embedded inside a ticket of some kind that must be decrypted in order to retrieve the actual user identity. If serverLogin() is successful that user identity is returned.

Updated the `loadUsers()` method, adding a check to `serverSecurity.areDefaultUsersCreated()` before calling `createDefaultServerUsers()`.

Changed `createDefaultServerUsers()`, immediately returning if `serverSecurity.arePasswordsStored()` returns false.

Added a try/catch block in `registerPlugins()` around the handler loading. This prevents a bad plugin from keeping the entire system from starting.

OHttpUtils

Added SPNEGO negotiate header support.

```
public static final String HEADER_AUTHENTICATE_NEGOTIATE = "WWW-Authenticate: Negotiate";
public static final String AUTHORIZATION_NEGOTIATE       = "Negotiate";
```

ONetworkProtocolHttpAbstract

Changed the `handlerError()` method, using `server.getSecurity().getAuthenticationHeader()` to set the `responseHeaders` variable.

Modified `readAllContent()`, adding "Negotiate" support.

Added two new commands, `OServerCommandGetSSO()` and `OServerCommandGetPing()`, in `registerStatelessCommands()`.

OServerCommandAuthenticatedDbAbstract

Replaced in `sendAuthorizationRequest()` the assignment to `header` of "Basic" authentication with the result of calling `server.getSecurity().getAuthenticationHeader(iDatabaseName)`.

OServerCommandAuthenticatedServerAbstract

Replaced in `sendAuthorizationRequest()` the assignment to `header` of "Basic" authentication with the result of calling `server.getSecurity().getAuthenticationHeader(iDatabaseName)`.

OServerCommandPostAuthToken

Replaced in `sendAuthorizationRequest()` the assignment to `header` of "Basic" authentication with the result of calling `server.getSecurity().getAuthenticationHeader(iDatabaseName)`.

OServerConfigurationManager

In the `setUser()` method, removed the check for `iServerUserPasswd` being empty, as some security implementations do not require a password.

New File - Added to orient/server/network

Added one new file, `OServerTLSSocketFactory.java`.

OServerTLSSocketFactory

OServerTLSSocketFactory simply extends OServerSSLSocketFactory to make it clear that OrientDB does support TLS encryption by default.

New Files - Added to orient/server/network/protocol/http/command/get

Added two new commands, `OServerCommandGetPing` and `OServerCommandGetSSO`.

OServerCommandGetPing

Added a very simple Get *Ping* command, used by Studio, to test if the HTTP server is still alive. As opposed to *listDatabases*, this command requires no authentication and no special authorization. This is important for heavyweight *authenticators*, such as SPNEGO/Kerberos.

OServerCommandGetSSO

This Get command is also used by Studio to determine if the primary *authenticator* supports single sign-on. If single sign-on is supported, then the *username* and *password* credentials in the login dialog are not required.

New File - Added to orient/server/network/protocol/http/command/post

Added one new command, `OServerCommandPostSecurityReload`.

OServerCommandPostSecurityReload

This command is designed to reload the current `OServerSecurity` module and accepts a path to the security JSON file to use.

New Files - Added to orient/server/security

The following files were added to the security directory:

- OAuditingService.java
- ODefaultPasswordAuthenticator.java
- ODefaultServerSecurity.java
- OSecurityAuthenticator.java
- OSecurityAuthenticatorAbstract.java
- OSecurityAuthenticatorException.java
- OSecurityComponent.java
- OServerConfigAuthenticator.java
- OServerSecurity.java
- OSyslog.java

OAuditingService

Provides a simple auditing service interface, and specifies three `log()` methods. Extends the `OSecurityComponent` interface.

ODefaultPasswordAuthenticator

Extends the `OSecurityAuthenticatorAbstract` class, providing a default *authenticator* that supports a username, password, and resource list.

ODefaultServerSecurity

Provides the default server security implementation, supporting these interfaces: `OSecurityFactory`, `OServerLifecycleListener`, and `OServerSecurity`.

OSecurityAuthenticator

Provides an interface for implementing a security *authenticator*. Extends the `OSecurityComponent` interface.

Requires the following methods.

Method	Description
authenticate()	Authenticates the specifed username and password. The authenticated *principal* is returned if successful, otherwise null.
getAuthenticationHeader()	Returns the HTTP authentication header supported by this *authenticator*.
getClientSubject()	If supported by this *authenticator*, returns a `Subject` object with this *authenticator's* credentials.
getName()	Returns the name of this `OSecurityAuthenticator` .
getUser()	Returns an `OServerUserConfiguration` object if the matching username is found for this *authenticator*. Returns null otherwise.
isAuthorized()	Takes a username and resource list (as a String). Returns true if a matching username is found for this *authenticator* that has the required authorization for the specified resource list.
isSingleSignOnSupported()	Returns a boolean that indicates if this *authenticator* supports single sign-on.

OSecurityAuthenticatorAbstract

Provides an abstract implementation of `OSecurityAuthenticator` upon which most *authenticators* are derived.

OSecurityAuthenticatorException

Provides a custom OException that security *authenticators* can throw.

OSecurityComponent

Provides an interface for creating security components that all modules in the security JSON configuration implement.

`OSecurityComponent` consists of four methods: `active()` , `config()` , `dispose()` , and `isEnabled()` .

OServerConfigAuthenticator

Provides an `OSecurityAuthenticator` implementation that supports the users listed in orientdb-server-config.xml. It extends `OSecurityAuthenticatorAbstract` .

OServerSecurity

Provides an interface for server-specific security features. It extends `OSecuritySystem` .

Requires the following methods.

Method	Description
getAuthenticator()	Returns the `authenticator` based on name, if one exists.
getPrimaryAuthenticator()	Returns the first `authenticator` in the chain of `authenticators` , which is the primary authenticator.
getUser()	Some `authenticators` support maintaining a list of users and associated resources (and sometimes passwords). Returns the associated `OServerUserConfiguration` if one exists for the specified *username*.
openDatabase()	Opens the specified *dbName*, if it exists, as the *superuser*.
registerAuditingService()	Supports a pluggable, external `OAuditingService` instance.
registerSecurityClass()	Supports multiple, pluggable, external security components.
reload()	Reloads the server security module using the specified *cfgPath*.

OSyslog

Provides an interface to syslog (and other such event logging systems) and specifies three `log()` methods. Extends the `OSecurityComponent` interface.

graphdb/config

orientdb-server-config.xml

Replaced OServerSSLSocketFactory with OServerTLSSocketFactory.

security.json

Added a default security.json implementation file.

Security Plugin

To support a single location for all the new security component implementations, a new project, *orientdb-security*, has been created.

The following security components now reside here:

- ODefaultAuditing
- ODefaultPasswordAuthenticator
- ODefaultPasswordValidator
- ODefaultSyslog
- OKerberosAuthenticator
- OLDAPImporter
- OServerConfigAuthenticator

Enterprise Agent

OAuditingHook

[This was moved into *orientdb-security*.]

Added support for auditing of CREATECLASS and DROPCLASS events.

Added optional support for *syslog* audit logging as well.

OAuditingListener

[This was moved into *orientdb-security* and renamed `ODefaultAuditing` .]

Implemented two new interfaces, `OAuditingService` and `ODistributedLifecycleListener` .

Also moved `Orient.instance().addDbLifecycleListener(this)` to the `active()` method.

OEnterpriseAgent

[This was removed from the agent and added to `OSecurityPlugin` in *orientdb-security*.]

Added a new method, `registerSecurityComponents()` , for registering Enterprise-only security components in the security system.

Added a call to `registerSecurityComponents()` in the `startup()` method.

agent/kerberos

[This was moved into *orientdb-security*.]

Added a new *kerberos* directory and the following new files.

File	Description
OKerberosAuthenticator.java	Implements a Kerberos-specific security *authenticator* that extends `OSecurityAuthenticatorAbstract` .
OKerberosLibrary.java	Provides a collection of static methods used by the Kerberos *authenticator*.
OKrb5LoginModuleConfig.java	Provides a custom Kerberos login `Configuration` implementation.

agent/ldap

[This was moved into *orientdb-security*.]

Added a new *ldap* directory and the following new files.

File	Description
OLDAPImporter.java	Provides LDAP user/group importer into a database. Implements the `OSecurityComponent` interface.
OLDAPLibrary.java	Provides a collection of static methods used by the LDAP importer.
OLDAPServer.java	Provides a simple class used by OLDAPImporter and OLDAPLibrary for specifying LDAP servers.

agent/security

[This was moved into *orientdb-security*.]

Added a new *security* directory and the following new files.

File	Description
ODefaultPasswordValidator.java	Provides a default implementation for validating passwords. Implements `OPasswordValidator` and `OSecurityComponent` .
ODefaultSyslog.java	Provides a default implementation of the `OSyslog` interface.

resources

[This was moved into *orientdb-security*.]

Modified the default-auditing-config.json file, adding class support for "onCreateClassEnabled", "onCreateClassMessage", "onDropClassEnabled", and "onDropClassMessage".

pom.xml

[This was moved into *orientdb-security*.]

Added the dependency for CloudBees *syslog* client support:

```
<dependency>
   <groupId>com.cloudbees</groupId>
   <artifactId>syslog-java-client</artifactId>
   <version>1.0.9-SNAPSHOT</version>
</dependency>
```

New Security Features - OrientDB

Overview

The new security features in OrientDB (introduced in release 2.2) provide an extensible framework for adding external authenticators, password validation, LDAP import of database roles and users, advanced auditing capabilities, and syslog support.

The new security system uses a JSON configuration file, located in the *config* directory. The default name of the file is *security.json*, but it can be overridden by setting the "server.security.file" property in *orientdb-server-config.xml* or by setting the global server property, "server.security.file".

To see the complete configuration options, click here: Security Configuration.

Authenticators

In addition to the built-in authentication of users that happens inside the database engine, the new OrientDB security system supports multiple types of external *authenticators*. Each *authenticator* implements various methods to support authentication and authorization, among other things.

All *authenticators* can be configured in the "authentication" section of the security configuration.

Current Implementations

Currently, OrientDB provides a Kerberos authenticator, a password authenticator for authenticating users in the *security.json* file, a server config authenticator for authenticating users in the *orientdb-server-config.xml* file, and a symmetric key authenticator (Enterprise-only). Additional *authenticators* may be provided in the future, and it's very easy to build new ones.

OKerberosAuthenticator

This *authenticator* provides support for Kerberos authentication and full browser SPNEGO support. See the security configuration page for full details on configuring this *authenticator* for Kerberos.

Also, see Kerberos client examples to see how to use the OrientDB clients with Kerberos.

ODefaultPasswordAuthenticator

The `ODefaultPasswordAuthenticator` supports adding server users with passwords and resources to the *security.json* configuration file. The main purpose of this is to allow having server users in a single file (along with all the other security settings) without having to maintain them in the separate *orientdb-server-config.xml* file. See the example in the security configuration page.

OServerConfigAuthenticator

The `OServerConfigAuthenticator` is similar to `ODefaultPasswordAuthenticator` in that it supports server users with passwords and resources, but it's designed to be used with the users in the *orientdb-server-config.xml* configuration file instead.

OSystemUserAuthenticator

The `OSystemUserAuthenticator` supports the new *system user* type that's stored in the system database.

Symmetric Key Authenticator

The `Symmetric Key Authenticator` provides support for authenticating users via a shared symmetric key.

There are two versions of the `Symmetric Key Authenticator`, one that enables authenticating server users and one that enables authenticating system users.

Chaining *Authenticators*

What's important to note is that the *authenticators* can be thought of as being chained together so that if the first *authenticator* in the list fails to authenticate a user, then the next *authenticator* in the chain is tried. This continues until either a user is successfully authenticated or all *authenticators* are tried. This "chaining of *authenticators*" is used for authentication, authorization, retrieving HTTP authentication headers, and several other security features.

Default Authentication

As mentioned previously, OrientDB has a built-in authentication system for each database, and it is enabled by default. To disable it, set the "allowDefault" property in the "authentication" section of the *security.json* configuration to false. When enabled, the built-in authentication acts as a "fallback" if the external *authenticators* cannot authenticate or authorize a user.

Password Validator

Another new feature of the security system is a customizable password validator. The provided validator is called `ODefaultPasswordValidator` that, when enabled, validates user-chosen passwords based on minimum length and regular expressions. Which numbers, uppercase letters, and special characters are permitted along with their required count is specified by a regular expression for each. Like all the security components, the password validator can be extended or completely replaced with a custom password validator component.

See the security configuration page for details on how to configure the password validator.

LDAP Import

Another often-requested feature, importing of LDAP users, is now available as part of OrientDB. Each database can be configured to import from multiple LDAP domains, each domain may specify multiple LDAP servers, and the users and their corresponding roles can be specified per-domain using a standard LDAP filter.

The security configuration page explains in great details all the options for the default LDAP importer.

Auditing/Syslog

Enhancements to the auditing component have also been made. The audit log now supports monitoring of a class being created and dropped as well as when distributed nodes join and leave the cluster. Additionally, for operating systems that support *syslog*, a new *syslog* plug-in has been added for recording auditing events.

See the security configuration page for details on auditing properties.

Reloading

The new security system supports dynamic reloading by an HTTP POST request. The path to the configuration file to use is provided as JSON content. The provided credentials must have sufficient privileges to reload the security module.

Here's an example using *curl* with basic authentication:

```
curl -u root:password -H "Content-Type: application/json" -X POST -d '{ "configFile" : "${ORIENTDB_HOME}/config/security.json"
 }'  servername:2480/security/reload
```

Here's another example using *curl* using Kerberos/SPNEGO authentication:

```
curl --negotiate -u : -H "Content-Type: application/json" -X POST -d '{ "configFile" : "${ORIENTDB_HOME}/config/security.json"
 }'  servername:2480/security/reload
```

Notice the passed-in JSON "configFile" property. Any valid security configuration file may be specified here, making testing multiple configurations possible.

Reloading Individual Components

Instead of reloading the entire security system, it's also possible to reload individual security components.

Here's an example using *curl* with basic authentication:

```
curl -u root:password -H "Content-Type: application/json" -X POST -d '{ "module" : "auditing", "config" : "{ACTUAL JSON CONTEN
T}" }'  servername:2480/security/reload
```

Notice, instead of specifying "configFile" you use "config" and the "module" property.

Currently, you can reload the following security components:

- server
- authentication
- passwordValidator
- ldapImporter
- auditing

Symmetric Key Authentication

Symmetric key authentication enables the use of a shared secret key to authenticate an OrientDB user. Support for this was added to OrientDB Enterprise Edition in 2.2.14.

Two new server-side authenticators were added: OSecuritySymmetricKeyAuth and OSystemSymmetricKeyAuth. See Security Configuration.

Overview

Symmetric key authentication works by both the server and the client sharing a secret key for an OrientDB user that's used to authenticate a client.

When a Java client (or via the OrientDB console) uses a symmetric key and tries to connect to an OrientDB server or open a database, the *username* is encrypted using the specified secret key. On the server-side, the user's corresponding secret key will be used to decrypt the username. If both usernames match, then the client is successfully authenticated.

The normal *password* field is used to encode the encrypted username as well as information specific to the key type. A Java client can use the OSymmetricKey class to encrypt the username as part of the password field or can use the OSymmetricKeyCI credential interceptor.

Note that the symmetric key is *never* sent from the client to the server as part of the authentication process.

OSymmetricKey Client Example

The OSymmetricKey class is used to create symmetric secret keys, load them from various sources, and to encrypt/decrypt data.

Here's an example using a Base64 key string with an OSymmetricKey object to encrypt the username, "Bob", and send that as the password field of the OServerAdmin connect() method:

```
String key = "8BC7LeGkFbmHEYNTz5GwDw==";
OSymmetricKey sk = new OSymmetricKey("AES", "AES/CBC/PKCS5Padding", key);

String url = "remote:localhost";
String username = "Bob";
String password = sk.encrypt(username);

OServerAdmin serverAdmin = new OServerAdmin(url).connect(username, password);
serverAdmin.createDatabase("TestDB", "graph", "plocal");
serverAdmin.close();
```

OSymmetricKeyCI Console Example

To use the symmetric key authentication from the OrientDB console, you need to set the credential interceptor in the console.sh (or console.bat for Windows) script.

Add this property to the `ORIENTDB_SETTINGS` variable:

```
ORIENTDB_SETTINGS="-Dclient.credentialinterceptor=com.orientechnologies.orient.core.security.symmetrickey.OSymmetricKeyCI"
```

The symmetric key credential interceptor is called `OSymmetricKeyCI` .

The password field is used to specify a JSON document that contains the properties for using a secret key string, a file containing a secret key string, or a Java KeyStore file.

One of the required JSON properties is 'transform', which indicates the cipher transformation. If 'transform' is not specified as part of the JSON document, the global property, `client.ci.ciphertransform` can be set in `ORIENTDB_SETTINGS` instead.

Another property that can be set is the key's algorithm type, set by specifying 'algorithm' in the JSON document. If 'algorithm' is not specified then the algorith is determined from the cipher transformation. The key's algorithm can also be set as part of the `ORIENTDB_SETTINGS` using the global property `client.ci.keyalgorithm`.

Example:

```
ORIENTDB_SETTINGS="-Dclient.credentialinterceptor=com.orientechnologies.orient.core.security.symmetrickey.OSymmetricKeyCI -Dcl
ient.ci.keyalgorithm=AES -Dclient.ci.ciphertransform=AES/CBC/PKCS5Padding"
```

key

Here's a console example using an AES 128-bit key:

```
connect remote:localhost:2424 serveruser "{'transform':'AES/CBC/PKCS5Padding','key':'8BC7LeGkFbmHEYNTz5GwDw=='}"
```

Note that the cipher transformation is specified via 'transform'. If 'algorithm' is not specified (the key's algorithm), then the algorith is determined from the cipher transformation.

keyFile

Here's a console example using an AES key file:

```
connect remote:localhost:2424 serveruser "{'transform':'AES/CBC/PKCS5Padding','keyFile':'config/user.key'}"
```

keyStore

Using a Java KeyStore requires a few more properties. Here's an example:

```
connect remote:localhost:2424 serveruser "{'keyStore':{ 'file':'config/server.ks', 'password':'password', 'keyAlias':'useralia
s', 'keyPassword':'password' } }"
```

In the above example, it's assumed the cipher transformation is set as part of the `ORIENTDB_SETTINGS`.

The 'keyStore' property is an object with the following properties:

'file'

The 'file' property is a path to a Java KeyStore file that contains the secret key. Note that the JCEKS KeyStore format must be used to hold secret keys.

'password'

The 'password' property holds the password for the KeyStore file. It is optional.

'keyAlias'

The 'keyAlias' property is the alias name of the secret key in the KeyStore file. It is required.

'keyPassword'

The 'keyPassword' property holds the password for the secret key in the KeyStore and is optional.

Manage a remote Server instance

Introduction

A remote server can be managed via API using the OServerAdmin class. Create it using the URL of the remote server as first parameter of the constructor.

```
OServerAdmin serverAdmin = new OServerAdmin("remote:localhost:2480");
```

You can also use the URL of the remote database:

```
OServerAdmin serverAdmin = new OServerAdmin("remote:localhost:2480/GratefulDeadConcerts");
```

Connect to a remote server

```
OServerAdmin serverAdmin = new OServerAdmin("remote:localhost:2480").connect("admin", "admin");
```

User and password are not the database accounts but the server users configured in orientdb-server-config.xml file.

When finished call the `OServerAdmin.close()` method to release the network connection.

Create a database

To create a new database in a remote server you can use the console's create database command or via API using the `OServerAdmin.createDatabase()` method.

```
// ANY VERSION: CREATE A SERVER ADMIN CLIENT AGAINST A REMOTE SERVER
OServerAdmin serverAdmin = new OServerAdmin("remote:localhost/GratefulDeadConcerts").connect("admin", "admin");
serverAdmin.createDatabase("graph", "local");
```

```
// VERSION >= 1.4: CREATE A SERVER ADMIN CLIENT AGAINST A REMOTE SERVER
OServerAdmin serverAdmin = new OServerAdmin("remote:localhost").connect("admin", "admin");
serverAdmin.createDatabase("GratefulDeadConcerts", "graph", "local");
```

The iStorageMode can be memory or plocal.

Drop a database

To drop a database from a server you can use the console's drop database command or via API using the `OServerAdmin.dropDatabase()` method.

```
// CREATE A SERVER ADMIN CLIENT AGAINST A REMOTE SERVER
OServerAdmin serverAdmin = new OServerAdmin("remote:localhost/GratefulDeadConcerts").connect("admin", "admin");
serverAdmin.dropDatabase("GratefulDeadConcerts");
```

Check if a database exists

To check if a database exists in a server via API use the `OServerAdmin.existsDatabase()` method.

```
// CREATE A SERVER ADMIN CLIENT AGAINST A REMOTE SERVER
OServerAdmin serverAdmin = new OServerAdmin("remote:localhost/GratefulDeadConcerts").connect("admin", "admin");
serverAdmin.existsDatabase("local");
```

Install as Service on Unix/Linux

Following the installation guide above, whether you choose to download binaries or build from source, does not install OrientDB at a system-level. There are a few additional steps you need to take in order to manage the database system as a service.

OrientDB ships with a script, which allows you to manage the database server as a system-level daemon. You can find it in the `bin/` path of your installation directory, (that is, at `$ORIENTDB_HOME/bin/orientdb.sh`.

The script supports three parameters:

- `start`
- `stop`
- `status`

Configuring the Script

In order to use the script on your system, you need to edit the file to define two variables: the path to the installation directory and the user you want to run the database server.

```
$ vi $ORIENTDB_HOME/bin/orientdb.sh

#!/bin/sh
# OrientDB service script
#
# Copyright (c) Orient Technologies LTD (http://www.orientechnologies.com)

# chkconfig: 2345 20 80
# description: OrientDb init script
# processname: orientdb.sh

# You have to SET the OrientDB installation directory here
ORIENTDB_DIR="YOUR_ORIENTDB_INSTALLATION_PATH"
ORIENTDB_USER="USER_YOU_WANT_ORIENTDB_RUN_WITH"
```

Edit the `ORIENTDB_DIR` variable to indicate the installation directory. Edit the `ORIENTDB_USER` variable to indicate the user you want to run the database server, (for instance, `orientdb`).

Installing the Script

Different operating systems and Linux distributions have different procedures when it comes to managing system daemons, as well as the procedure for starting and stopping them during boot up and shutdown. Below are generic guides for init and systemd based unix systems as well Mac OS X. For more information, check the documentation for your particular system.

Installing for init

Many Unix-like operating systems such as FreeBSD, most older distributions of Linux, as well as current releases of Debian, Ubuntu and their derivatives use variations on SysV-style init for these processes. These are typically the systems that manage such processes using the `service` command.

To install OrientDB as a service on an init-based unix or Linux system, copy the modified `orientdb.sh` file from `$ORIENTDB_HOME/bin` into `/etc/init.d/`:

```
# cp $ORIENTDB_HOME/bin/orientdb.sh /etc/init.d/orientdb
```

Once this is done, you can start and stop OrientDB using the `service` command:

```
# service orientdb start
Starting OrientDB server daemon...
```

Installing for systemd

Most newer releases of Linux, especially among the RPM-based distributions like Red Hat, Fedora, and CentOS, as well as future releases of Debian and Ubuntu use systemd for these processes. These are the systems that manage such processes using the `systemctl` command.

The OrientDB's package contains a service descriptor file for systemd based distros. The `orientdb.service` is placed in the `bin` directory. To install OrientDB copy the `orientdb.service` to `/etc/systemd/system` directory (check this, may depend on distro). Edit the file:

```
# vi /etc/systemd/system/orientdb.service

#
# Copyright (c) OrientDB LTD (http://http://orientdb.com/)
#

[Unit]
Description=OrientDB Server
After=network.target
After=syslog.target

[Install]
WantedBy=multi-user.target

[Service]
User=ORIENTDB_USER
Group=ORIENTDB_GROUP
ExecStart=$ORIENTDB_HOME/bin/server.sh
```

Set the right user and group. You may want to use the absolute path instead of the environment variable `$ORIENTDB_HOME` . Once this file is saved, you can start and stop the OrientDB server using the `systemctl` command:

```
# systemctl start orientdb.service
```

Additionally, with the `orientdb.service` file saved, you can set systemd to start the database server automatically during boot by issuing the `enable` command:

```
# systemctl enable orientdb.service
Synchronizing state of orientdb.service with SysV init with /usr/lib/systemd/systemd-sysv-install...
Executing /usr/lib/systemd/systemd-sysv-install enable orientdb
Created symlink from /etc/systemd/system/multi-user.target.wants/orientdb.service to /etc/systemd/system/orientdb.service.
```

Installing for Mac OS X

Manual install

For Mac OS X, create an alias to the OrientDB system daemon script and the console.

```
$ alias orientdb-server=/path/to/$ORIENTDB_HOME/bin/orientdb.sh
$ alias orientdb-console=/path/to/$ORIENTDB_HOME/bin/console.sh
```

You can now start the OrientDB database server using the following command:

```
$ orientdb-server start
```

Once the database starts, it is accessible through the console script.

```
$ orientdb-console

OrientDB console v.1.6 www.orientechnologies.com
Type 'HELP' to display all the commands supported.

orientdb>
```

Brew

OrientDB is available through brew.

```
$ brew install orientdb
```

The installation process gives an output similar to the following one:

```
...
==> Downloading https://orientdb.com/download.php?file=orientdb-community-<ORIENTDB_VERSION>.tar.gz
==> /usr/bin/nohup  /usr/local/Cellar/orientdb/<ORIENTDB_VERSION>/libexec/bin/server.sh &
==> /usr/local/Cellar/orientdb/<ORIENTDB_VERSION>/libexec/bin/shutdown.sh
==> OrientDB installed, server's root user password is 'changeme'
==> Please, follow the instruction on the link below to reset it
==> http://orientdb.com/docs/2.2/Server-Security.html#restoring-the-servers-user-root
...
```

The installation process setups a default server's root user password that **must** be changed. The `orientdb-server-config.xml` file is installed in `/usr/local/Cellar/orientdb/<ORIENTDB_VERSION>/libexec/config/`. Open the file and remove the "root" user entry. Remove the tag true at the end of the file. Start the server on interactive console:

```
/usr/local/Cellar/orientdb/<ORIENTDB_VERSION>/libexec/bin/server.sh
```

The script asks for a new password for the database's root user.

Other resources

To learn more about how to install OrientDB on specific environment please follow the guide below:

- Install on Linux Ubuntu
- Install on JBoss AS
- Install on GlassFish
- Install on Ubuntu 12.04 VPS (DigitalOcean)
- Install as service on Unix, Linux and MacOSX
- Install as service on Windows

Install as a Service on Windows

OrientDB is a Java server application. As most server applications, they have to perform several tasks, before being able to shut down the Virtual Machine process, hence they need a portable way to be notified of the imminent Virtual Machine shutdown. At the moment, the only way to properly shut down an OrientDB server instance (not embedded) is to execute the *shutdown.bat* (or *shutdown.sh*) script shipped with the OrientDB distribution, but it's up to the user to take care of this. This implies that the server instance isn't stopped correctly, when the computer on which it is deployed, is shut down without executing the above script.

Apache Commons Daemon

Apache Commons Daemon is a set of applications and API enabling Java server application to run as native non interactive server applications under Unix and Windows. In Unix, server applications running in the background are called *daemons* and are controlled by the operating system with a set of specified *signals*. Under Windows, such programs are called services and are controlled by appropriate calls to specific functions defined in the application binary. Although the ways of dealing with running daemons or services are different, in both cases the operating system can notify a server application of its imminent shutdown, and the underlying application has the ability to perform certain tasks, before its process of execution is destroyed. Wrapping OrientDB as a *Unix daemon* or as a *Windows service* enables the management of this server application lifecycle through the mechanisms provided natively by both Unix and Windows operating systems.

Installation

This tutorial is focused on Windows, so you have to download *procrun*. Procrun is a set of applications, which allow Windows users to wrap (mostly) Java applications (e.g. Tomcat) as a Windows service. The service can be set to automatically start, when the machine boots and will continue to run with no user logged onto the machine.

1. Point you browser to the Apache Commons Daemon download page.
2. Click on **browse download area**: you will see the index **commons/daemon/**.
3. Click on **binaries/**: you will see the index **commons/daemon/binaries/**.
4. Click on **windows**. Now you can see the index of **commons/daemon/binaries/windows**.
5. Click on **commons-daemon-1.0.15-bin-windows.zip**. The download starts.
6. Unzip the file in a directory of your choice. The content of the archive is depicted below:

```
commons-daemon-1.0.15-bin-windows
|
\---amd64
    |
    \---prunsrv.exe
|
\---ia64
    |
    \---prunsrv.exe
|
\---LICENCE.txt
|
\---NOTICE.txt
|
\---prunmgr.exe
|
\---prunsrv.exe
|
\---RELEASE-NOTES.txt
```

If version 1.0.15 is not listed, check Apache Commons Daemon Windows download page directly.

prunmgr is a GUI application for monitoring and configuring Windows services wrapped with procrun. **prunsrv** is a service application for running applications as services. It can convert any application (not just Java applications) to run as a service. The directory **amd64** contains a version of **prunsrv** for x86-64 machines while the directory **ia64** contains a version of **prunsrv** for Itanium 64 machines.

Once you downloaded the applications, you have to put them in a folder under the OrientDB installation folder.

1. Go to the OrientDB folder, in the following referred as *%ORIENTDB_HOME%*
2. Create a new directory and name it **service**
3. Copy there the appropriate versions of **prunsrv** according to the architecture of your machine and **prunmgr**.

Configuration

In this section, we will show how to wrap OrientDB as a Windows Service. In order to wrap OrientDB as a service, you have to execute a short script that uses the prunsrv application to configure a Windows Service.

Before defining the Windows Service, you have to rename **prunsrv** and **prunmgr** according to the name of the service. Both applications require the name of the service to manage and monitor as parameter but you can avoid it by naming them with the name of the service. In this case, rename them respectively **OrientDBGraph** and **OrientDBGraphw** as *OrientDBGraph* is the name of the service that you are going to configure with the script below. If you want to use a difference service name, you have to rename both application respectively **myservicename** and **myservicenamew** (for example, if you are wrapping OrientDB and the name of the service is *OrientDB*, you could rename *prunsrv* as *OrientDB* and *prunmgr* as *OrientDBw*). After that, create the file **%ORIENTDB_HOME%\service\installService.bat** with the content depicted below:

```
:: OrientDB Windows Service Installation
@echo off
rem Remove surrounding quotes from the first parameter
set str=%~1
rem Check JVM DLL location parameter
if "%str%" == "" goto missingJVM
set JVM_DLL=%str%
rem Remove surrounding quotes from the second parameter
set str=%~2
rem Check OrientDB Home location parameter
if "%str%" == "" goto missingOrientDBHome
set ORIENTDB_HOME=%str%

set CONFIG_FILE=%ORIENTDB_HOME%/config/orientdb-server-config.xml
set LOG_FILE=%ORIENTDB_HOME%/config/orientdb-server-log.properties
set LOG_CONSOLE_LEVEL=info
set LOG_FILE_LEVEL=fine
set WWW_PATH=%ORIENTDB_HOME%/www
set ORIENTDB_ENCODING=UTF8
set ORIENTDB_SETTINGS=-Dprofiler.enabled=true -Dcache.level1.enabled=false -Dcache.level2.strategy=1
set JAVA_OPTS_SCRIPT=-XX:+HeapDumpOnOutOfMemoryError

rem Install service
OrientDBGraph.exe //IS --DisplayName="OrientDB GraphEd X.X.X" ^
--Description="OrientDB Graph Edition, aka GraphEd, contains OrientDB server integrated with the latest release of the TinkerP
op Open Source technology stack supporting property graph data model." ^
--StartClass=com.orientechnologies.orient.server.OServerMain --StopClass=com.orientechnologies.orient.server.OServerShutdownMa
in ^
--Classpath="%ORIENTDB_HOME%\lib\*;%ORIENTDB_HOME%\plugins\*" --JvmOptions=-Dfile.encoding=%ORIENTDB_ENCODING%;-Djava.util.log
ging.config.file="%LOG_FILE%";-Dorientdb.config.file="%CONFIG_FILE%";-Dorientdb.www.path="%WWW_PATH%";-Dlog.console.level=%LOG
_CONSOLE_LEVEL%;-Dlog.file.level=%LOG_FILE_LEVEL%;-Dorientdb.build.number="@BUILD@";-DORIENTDB_HOME="%ORIENTDB_HOME%" ^
--StartMode=jvm --StartPath="%ORIENTDB_HOME%\bin" --StopMode=jvm --StopPath="%ORIENTDB_HOME%\bin" --Jvm="%JVM_DLL%" --LogPath=
"%ORIENTDB_HOME%\log" --Startup=auto

EXIT /B

:missingJVM
echo Insert the JVM DLL location
goto printUsage

:missingOrientDBHome
echo Insert the OrientDB Home
goto printUsage

:printUsage
echo usage:
echo     installService JVM_DLL_location OrientDB_Home
EXIT /B
```

The script requires two input parameters:

1. The location of jvm.dll, for example *C:\Program Files\Java\jdk1.6.0_26\jre\bin\server\jvm.dll*
2. The location of the OrientDB installation folder, for example *D:\orientdb-graphed-1.0rc5*

The service is actually installed when executing **OrientDBGraph.exe** (originally prunsrv) with the appropriate set of command line arguments and parameters. The command line argument **//IS** states that the execution of that application will result in a service installation. Below there is the table with the command line parameters used in the above script.

Parameter name	Description	Source
--DisplayName	The name displayed in the Windows Services Management Console	Custom
--Description	The description displayed in the Windows Services Management Console	Custom
--StartClass	Class that contains the startup method (= the method to be called to start the application). The default method to be called is the `main` method	The class invoked in the */bin/server.bat* script
--StopClass	Class that will be used when receiving a Stop service signal. The default method to be called is the `main` method	The class invoked in the */bin/shutdown.bat* script
--Classpath	Set the Java classpath	The value of the `-cp` parameter specified in the _%ORIENTDB_HOME%\bin\server.bat_ script
--JvmOptions	List of options to be passed to the JVM separated using either # or ; characters	The list of options in the form of -D or -X specified in the _%ORIENTDB_HOME%\bin\server.bat_ script and the definition of the ORIENTDB_HOME system property
--StartMode	Specify how to start the process. In this case, it will start Java in-process and not as a separate image	Based on Apache Tomcat configuration
--StartPath	Working path for the StartClass	_%ORIENTDB_HOME%\bin_
--StopMode	The same as --StartMode	Based on Apache Tomcat configuration
--StopPath	Working path for the StopClass	_%ORIENTDB_HOME%\bin_
--Jvm	Which *jvm.dll* to use: the default one or the one located in the specified full path	The first input parameter of this script. Ensure that you insert the location of the Java HotSpot Server VM as a full path. We will use the server version for both start and stop.
--LogPath	Path used by prunsrv for logging	The default location of the Apache Commons Daemon log
--Startup	States if the service should start at machine start up or manually	auto

For a complete reference to all available parameters and arguments for prunsrv and prunmgr, visit the Procrun page.

In order to install the service:

1. Open the Windows command shell
2. Go to *%ORIENTDB_HOME%\service*, for example typing in the shell `> cd D:\orientdb-graphed-1.0rc5\service`
3. Execute the *installService.bat* specifying the *jvm.dll* location and the OrientDB Home as full paths, for example typing in the shell `> installService.bat "C:\Program Files\Java\jdk1.6.0_26\jre\bin\server\jvm.dll" D:\orientdb-graphed-1.0rc5`
4. Open the Windows Services Management Console - from the taskbar, click on *Start*, *Control Panel*, *Administrative Tools* and then *Service* - and check the existance of a service with the same name specified as value of the `--DisplayName` parameter (in this case **OrientDB GraphEd X.X.X**). You can also use *%ORIENTDB_HOME%\service\OrientDBGraphw.exe* to manage and monitor the *OrientDBGraph* service.

Example (in the `service` directroy) start and stop the service:

```
:start
OrientDBGraph.exe //ES

:stop
OrientDBGraph.exe //SS
```

Note that you need to start the OrientDB server once manually via `server.bat` in %ORIENTDB_HOME%\bin once, before starting the service.

Uninstallation

Create the file **%ORIENTDB_HOME%\service\uninstallService.bat** with the content depicted below and run this file:

```
:: OrientDB Windows Service Uninstallation
@echo off
rem Uninstall service
OrientDBGraph.exe //DS
```

Other resources

To learn more about how to install OrientDB on specific environment please follow the guide below:

- Install on Linux Ubuntu
- Install on JBoss AS
- Install on GlassFish
- Install on Ubuntu 12.04 VPS (DigitalOcean)
- Install as service on Unix, Linux and MacOSX
- Install as service on Windows

Installing in a Docker Container

If you have Docker installed in your computer, this is the easiest way to run OrientDB. From the command line type:

```
$ docker run -d --name orientdb -p 2424:2424 -p 2480:2480
   -e ORIENTDB_ROOT_PASSWORD=root orientdb:latest
```

Where instead of "root", type the root's password you want to use.

Building the image on your own

Dockerfiles are available on a dedicated repository. The repository has a folder for each maintained version of OrientDB. Dockerfiles are approved by Docker's team.. This allows to build images on your own or even customize them for your special purpose.

1. Clone this project to a local folder:

   ```
   git clone https://github.com/orientechnologies/orientdb-docker.git
   ```

2. Build the image for 2.2.x:

   ```
   cd 2.2
   docker build -t <YOUR_DOCKER_HUB_USER>/orientdb:2.2.11 .
   ```

3. Push it to your Docker Hub repository (it will ask for your login credentials):

   ```
   docker push <YOUR_DOCKER_HUB_USER>/orientdb:2.2.11
   ```

OrientDB Stress Test Tool

The OrientDB Stress Test Tool is an utility for very basic benchmarking of OrientDB.

Syntax

```
StressTester
    -m [plocal|memory|remote] (mandatory)
    -w <workload-name>:<workload-params>*
    -c <concurrency level> (number of parallel threads)
    -tx <operationsPerTransaction>
    -o <resultOutputFile>
    -d <plocalDirectory>
    -chk true|false
    -k true|false
--root-password <rootPassword>
--remote-ip <remoteIpOrHost>
--remote-port <remotePort>
--ha-metrics true|false
```

- the **m** parameter sets the type of database to be stressed.
- the **c** parameter sets the concurrency level, as the number of threads that will be launched. Every thread will execute the complete operationSe. If not present, it defaults to 4.
- the **tx** parameter sets the number of operations to be included in a transaction. This value must be lesser than the number of creates divided by the threads number and the iterations number. If the **tx** parameter is not present, all the operations will be executed outside transactions.
- the **w** parameter defines the workloads. To specify multipel workloads, use the comma (`,`) to separate them, but do not use any space. Workloads are pluggable, the available ones are:
    - **CRUD**, executes, in order, (C)reate, (R)ead, (U)pdate and (D)elete operations. The `<workload-params>` must follow the format `C#R#U#D#`, where the '#' is a number:
        - C1000 defines 1000 Create operations
        - R1000 defines 1000 Read operations
        - U1000 defines 1000 Update operations
        - D1000 defines 1000 Delete operations

        So a valid set is C1000R1000U1000D1000. There is only one constraint: the number of reads, updates and deletes cannot be greater than the number of creates. If not present, it defaults to C5000R5000U5000D5000.
    - **GINSERT**, Insert a graph where all the nodes are connected with each others. The `<workload-params>` must follow the format `V#F#`, where the '#' is a number:
        - V1000 creates 1000 vertices
        - F10 Each vertex has 10 edges
    - **GSP**, Executes a shortest path between all the vertices against all the other vertices. The `<workload-params>` must follow the format `L#`, where the '#' is a number::
        - L1000 set the limit to 1000 vertiecs only. Optional.
- the **o** parameter sets the filename where the results are written in JSON format.
- the **d** parameter sets the base directory for writing the plocal database
- the **k** keeps the database at the end of workload. By default is `false`, so the database is dropped.
- the **chk** Checks the database created by the workload at the end of the test. Default is `false`.
- the **remote-ip** parameter defines the remote host (or IP) of the server to connect to. The StressTester will fail if this parameter is not specified and mode is **remote**.
- the **remote-port** parameter defines the port of the server to connect to. If not specified, it defaults to 2424.
- the **root-password** parameter sets the root password of the server to connect to. If not specified and if mode is **remote**, the root password will be asked at the start of the test.
- the **ha-metrics** (since v2.2.7) setting dumps the HA metrics at the end of each workload

If the StressTester is launched without parameters, it fails because the **-m** parameter is mandatory.

How it works

The stress tester tool creates a temporary database where needed (on memory / plocal / remote) and then creates a pool of N threads (where N is the threadsNumber parameter) that - all together - execute the number of operations defined in the OperationSet. So, if the number of Creates is 1000 and the thread number is 4, every single thread will execute 250 Creates (1000/4). After the execution of the test (or any error) the temporary database is dropped.

Example with CRUD workload

Executing a CRUD workload by inserting, reading, updating and deleting 100,000 records by using a connection of type "plocal" and 8 parallel threads:

```
cd bin
./stresstester -m plocal -c 8 -w crud:C100000R100000U100000D100000
```

This is the result:

```
OrientDB Stress Tool v.2.2.4-SNAPSHOT - Copyrights (c) 2016 OrientDB LTD
WARNING: 'tx' option not found. Defaulting to 0.
Created database [plocal:/var/folders/zc/y34429014c3bt_x1587qblth0000gn/T/stress-test-db-20160701_165901].

Starting workload CRUD (concurrencyLevel=8)...
- Workload in progress 100% [Creates: 100% - Reads: 100% - Updates: 100% - Deletes: 100%]
- Total execution time: 27.602 secs
- Created 1000000 records in 26.464 secs
  - Throughput: 37787.184/sec - Avg: 0.026ms/op (0th percentile) - 99th Perc: 0.799ms - 99.9th Perc: 5.769ms
- Read 1000 records in 0.096 secs
  - Throughput: 10416.667/sec - Avg: 0.096ms/op (0th percentile) - 99th Perc: 2.597ms - 99.9th Perc: 6.860ms
- Updated 1000 records in 0.043 secs
  - Throughput: 23255.814/sec - Avg: 0.043ms/op (0th percentile) - 99th Perc: 1.404ms - 99.9th Perc: 3.505ms
- Deleted 1000 records in 0.107 secs
  - Throughput: 9345.794/sec - Avg: 0.107ms/op (0th percentile) - 99th Perc: 3.175ms - 99.9th Perc: 5.664ms

Dropped database [plocal:/var/folders/zc/y34429014c3bt_x1587qblth0000gn/T/stress-test-db-20160701_165901].
```

The first part of the result is updated as long as the test is running, to give the user an idea of how long it will last. It will be deleted as soon as the test successfully terminates. The second part shows the results of the test:

- The total time of execution
- The times of execution of every operation type, their percentiles and the throughput.

The time is computed by summing up the times of execution of all the threads and dividing it by their number; the percentile value shows where the average result is located compared to all other results: if the average is a lot higher than 50%, it means that there are a few executions with higher times that lifted up the average (and you can expect better performance in general); a high percentile can happen when, for example, the OS or another process is doing something else (either CPU or I/O intensive) during the execution of the test.

If you plan to use the results of the StressTester the **o** option is available for writing the results in JSON format on disk.

Example with Graph workloads

Insert 100 vertices with 10 edges each (all connected), then execute a shortest path between all of them and checks the database integrity at the end.

```
cd bin
./stresstester -m plocal -c 16 -w GINSERT:V100F10,GSP -chk true
```

This is the result:

```
OrientDB Stress Tool v.2.2.4-SNAPSHOT - Copyrights (c) 2016 OrientDB LTD
WARNING: 'tx' option not found. Defaulting to 0.
Created database [plocal:/var/folders/zc/y34429014c3bt_x1587qblth0000gn/T/stress-test-db-20160701_170356].

Starting workload GINSERT (concurrencyLevel=16)...
- Workload in progress 100% [Vertices: 100 - Edges: 84]
- Total execution time: 0.068 secs
- Created 100 vertices and 84 edges in 0.052 secs
- Throughput: 1923.077/sec - Avg: 0.520ms/op (0th percentile) - 99th Perc: 15.900ms - 99.9th Perc: 15.900ms
- Checking database...
   - Repair of graph 'plocal:/var/folders/zc/y34429014c3bt_x1587qblth0000gn/T/stress-test-db-20160701_170356' is started ...
   - Scanning 99 edges...
   - Scanning edges completed
   - Scanning 100 vertices...
   - Scanning vertices completed
   - Repair of graph 'plocal:/var/folders/zc/y34429014c3bt_x1587qblth0000gn/T/stress-test-db-20160701_170356' completed in 0 s
ecs
   -  scannedEdges.....: 99
   -  removedEdges.....: 0
   -  scannedVertices..: 100
   -  scannedLinks.....: 198
   -  removedLinks.....: 0
   -  repairedVertices.: 0
- Check completed

Starting workload GSP (concurrencyLevel=16)...
- Workload in progress 100% [Shortest paths blocks (block size=100) executed: 100/100]
- Total execution time: 2.065 secs
- Executed 100 shortest paths in 2.060 secs
- Path depth: maximum 18, average 8.135, not connected 0
- Throughput: 48.544/sec - Avg: 20.600ms/op (0th percentile) - 99th Perc: 650.284ms - 99.9th Perc: 650.284ms
- Checking database...
   - Repair of graph 'plocal:/var/folders/zc/y34429014c3bt_x1587qblth0000gn/T/stress-test-db-20160701_170356' is started ...
   - Scanning 99 edges...
   - Scanning edges completed
   - Scanning 100 vertices...
   - Scanning vertices completed
   - Repair of graph 'plocal:/var/folders/zc/y34429014c3bt_x1587qblth0000gn/T/stress-test-db-20160701_170356' completed in 0 s
ecs
   -  scannedEdges.....: 99
   -  removedEdges.....: 0
   -  scannedVertices..: 100
   -  scannedLinks.....: 198
   -  removedLinks.....: 0
   -  repairedVertices.: 0
- Check completed

Dropped database [plocal:/var/folders/zc/y34429014c3bt_x1587qblth0000gn/T/stress-test-db-20160701_170356].
```

API

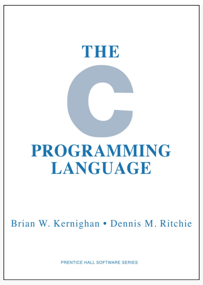
OrientDB supports 3 kinds of drivers:

- **Native Binary Remote**, that talks directly against the TCP/IP socket using the binary protocol
- **HTTP REST/JSON**, that talks directly against the TCP/IP socket using the HTTP protocol
- **Java-wrapped**, as a layer that links in some way the native Java driver. This is pretty easy for languages that run into the JVM like Scala, Groovy and JRuby

Look also at the available integration with Plugins and Frameworks.

This is the list of the known drivers to use OrientDB through different languages:

Language	Name	Type	Description
Java	Java (native) API	Native	Native implementation.
	JDBC driver	Native	For legacy and reporting/Business Intelligence applications and JCA integration for J2EE containers
	OrientDB Spring Data	Native	Official Spring Data Plugin for both Graph and Document APIs
node	OrientJS	Native	Binary protocol, new branch that has been updated with the latest functionality. Tested on 1.7.0, 2.0.x and 2.1-rc*.
	node-orientdb-http	HTTP	RESTful HTTP protocol. Tested on 1.6.1
	Gremlin-Node		To execute Gremlin queries against a remote OrientDB server
php	PhpOrient	Binary	**Official Driver**
	OrientDB-PHP	Binary	This was the first PHP driver for OrientDB, but doesn't support all OrientDB features and it's slow to support new versions of driver protocol.
	Doctrine ODM	Uses OrientDB-PHP	High level framework to use OrientDB from PHP
	.NET driver for OrientDB	Binary	**Official Driver**
python	PyOrient	Binary	Community driver for Python, compatible with OrientDB 1.7 and further.
	Bulbflow project	HTTP	Uses Rexter Graph HTTP Server to access to OrientDB database Configure Rexster for OrientDB
	Compass	HTTP	
golang	OrientGO	Binary	OrientGo is a Go client for the OrientDB database.
	OrientDB-C	Binary	Binary protocol compatibles with C++ and other languages that supports C calls

	LibOrient	Binary	As another Binary protocol driver
	Javascript Driver	HTTP	This driver is the simpler way to use OrientDB from JS
	Javascript Graph Driver	HTTP	This driver mimics the [Blueprints](https://github.com/orientechnologies/orientdb/wiki/Graph-Database-Tinkerpop) interface. Use this driver if you're working against graphs.
	Active-Orient	HTTP	Use OrientDB to persistently store dynamic Ruby-Objects and use database queries to manage even very large datasets. The gem is rails 5 compatible.
	OrientDB-JRuby	Native	Through Java driver
	OrientDB Client	Binary	
	OrientDB4R	HTTP	
	OrientDB Groovy	Java wrapper	This project contains Groovy AST Transformations trying to mimic grails-entity style. All useful information you can find in Spock tests dir. Document API and Graph API with gremlin are supported. Built with OrientDB 2.1.0 and Apache Groovy 2.4.4.
	Any Java driver	Native	Scala runs on top of JVM and it's fully compatible with Java applications like OrientDB
	Scala Page	Native	Offers suggestions and examples to use it without pains
	Scala utilities and tests	Native	To help Scala developers using OrientDB
	R driver	HTTP	R Bridge to execute queries against OrientDB Server
	MarcoPolo Elixir driver	Binary	This driver allows Elixir application to interact with OrientDB. Elixir language leverages the Erlang VM, known for running low-latency, distributed and fault-tolerant systems, while also being successfully used in web development and the embedded software domain.
	Clojure binding	Native	Through Java driver
	Clojure binding of Blueprints API		
	OrientDB Android	Porting	OrientDB-Android is a port/fork of OrientDB for the Android platform by David Wu
	OrientDB Perl driver	Binary	PlOrient is a Perl binary interface for OrientDB

Supported standards

This is the list of the library to use OrientDB by using such standard:

TinkerPop Blueprints

TinkerPop Blueprints, the standard for Graph Databases. OrientDB is 100% compliant with the latest version.

All the trademarks are property of their legal owners.

Functions

This feature allows you to define custom executable units of code that takes parameters from the database or query and return a modified result-set.

> **NOTE**: This guide refers to the last available release of OrientDB. For past revisions look at Compatibility.

Understanding Functions

There are times when you may need to perform some operation or use some feature that is simply not available in OrientDB. While you can develop workarounds at the application layer, you'll generally see better performance if you can move this logic to teh database layer. Relational databases solve this issue with Stored Procedures, which allow users to define custom code in what is often a vendor-specific programming language. OrientDB solves this issue with Functions.

OrientDB Functions are executable units of code. They allow you to use the Functional programming paradigm to develop custom features to better support your applications and infrastructure.

- **Persistent** Functions are persistent. They are stored on the database and can be called by any client.
- **Multiple Language Support** Functions support multiple languages. Currently, you can write them in OrientDB SQL or JavaScript. Support for Ruby, Scala, Java and other languages is currently in development.
- **Multiple Execution Support** OrientDB can execute Functions through SQL, Java, REST and Studio.
- **Recursion** Functions support
- **Mapping** Functions automatically map parameters by position and name.
- **Extensibilty** OrientDB Plugins can inject new objects for Functions to use.

> The OrientDB SQL dialect supports many functions written in the native language. To get better performance, you can write your own native functions in the Java language and register them to the engine.
>
> For more information, see Custom Functions in Java.

Compatibility

1.5.0 and before

OrientDB binds the following variables:

- `db`, that is the current document database instance
- `gdb`, that is the current graph database instance

OrientDB Function Creation

OrientDB provides a number of functions by default. In the event that these are not sufficient for your needs, you can create custom functions using OrientDB SQL or JavaScript. You can then execute them as SQL, HTTP or Java.

Creating Functions

When you create a new function on your database, OrientDB saves it using the `OFunction` class. You can query records of this class as you would any other in the database. The class has the following properties:

Property	Description
`name`	Defines the name of the function.
`code`	Defines the code the function executes.
`parameters`	Defines an optional `EMBEDDEDLIST` of strings, containing the parameter names, if any.
`idempotent`	Defines whether the function is idempotent, that is if it changes the database. Read-only functions are idempotent. This is needed to avoid calling non-idempotent functions using the HTTP GET method.

> Given that OrientDB uses one record per function, the MVCC mechanism is used to protect it against concurrent record updates.

There are two ways to create a function on the database: Studio and SQL

Using OrientDB Studio

Whether you are new to OrientDB and unfamiliar with the command syntax or you simply prefer graphical- to text-based interfaces, OrientDB ships with a convenient web interface for accessing and manipulating databases. Using Studio, you can create and manage custom Functions.

When you log into a database in Studio, there are a series of links running along the top of the page. The **Functions** tab takes you to a page where you can create, edit and otherwise manage custom functions in the database.

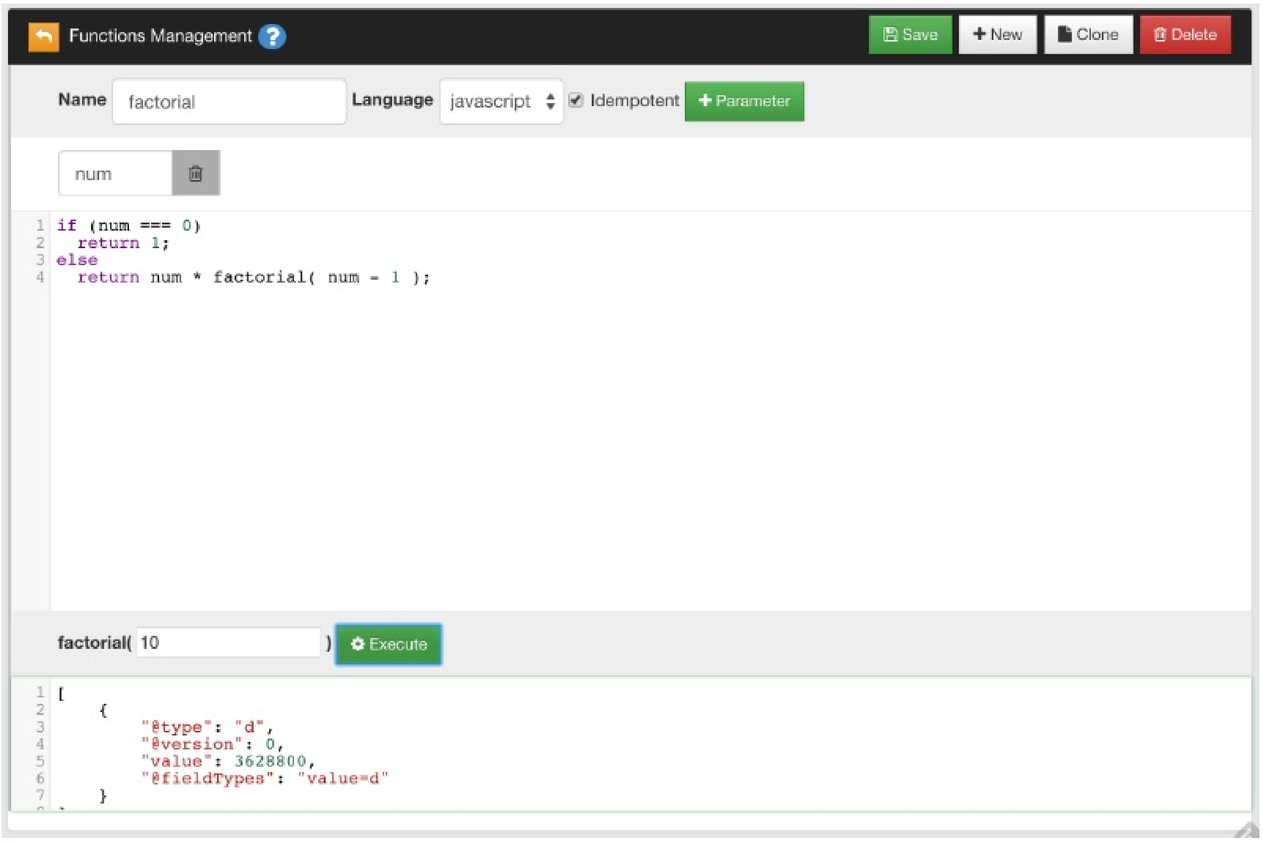
Example

Imagine a situation where you need to caculate factorials on a regular basis. While you could pull this information into your application, it would save time and network traffic if you could have OrientDB process the math and return the results. OrientDB Functions support recursion, so this is a fairly straightforward process.

To manage this, you would navigate to the Functions tab and create a new function, naming it `factorial` or whatever name you find most approriate. This function is written in JavaScript and takes one argument, called `num`. Then, provide the following code:

```
if (num === 0)
    return 1;
else
    return num * factorial(num - 1);
```

When this is done, click Save to update the database with the new function. As you can see, OrientDB Functions support recursion. When `factorial()` is called, it calls itself in calculating the results.



Name: factorial
Language: javascript
[✓ Idempotent]

num

```
if (num === 0)
  return 1;
else
  return num * factorial( num - 1 );
```

factorial(10)

```
[
    {
        "@type": "d",
        "@version": 0,
        "value": 3628800,
        "@fieldTypes": "value=d"
    }
```

Once you've saved a function, it's accessible to the database. You can also test functions below the text block by providing them with arguments and then clicking the Execute buttom. OrientDB then displays the return value, which here is 3648800.0.

Alternatively, you can test it from the Console:

```
orientdb> SELECT factorial(10)
3628800
```

Using OrientDB SQL

In the event that you prefer working from shell environments, OrientDB also provides the `CREATE FUNCTION` SQL command. For instance, take the above example creating a `factorial()` function.

```
orientdb> CREATE FUNCTION factorial "if (num === 0) return 1;
          else return num * factorial(num - 1)"
          PARAMETERS [num]
          LANGUAGE javascript
```

Managing Functions

Using API's like PyOrient or OrientJS in combination with OrientDB SQL commands, you can manage and update Functions using basic scripts to synchronize a repository with the database.

For instance, imagine you have a Git repository that contains a series of JavaScript files as well as a manifest file written in JSON, containing the metadata OrientDB needs to configure functions on the database. Using a simple Python function in a script, you could automatically synchronize all OrientDB Functions with your repository code.

```
################## Update Functions ###################
def update_orientdb_functions(client, funcs):
    """ Recieves PyOrient client with opened database
    and a dictionary containing metadata for each
    function

    funcs = {
       "func-name": {
          "arguments": ["arg1", "arg2"],
          "code": "/path/to/code.js"}
    }

    Connects to OrientDB Database and creates or updates
    functions with the given code."""

    # Loop over Functions Dict
    for name, data in funcs.items():


        # Read Code from File
        with open(data["code"], "r") as f:
            code = f.read()

        # Create Function when it does not Exist
        query_func = "SELECT FROM OFunction WHERE name = '%s'" % name
        if len(client.command(query_func)) == 0:

            # Create Function Command
            osql = "CREATE FUNCTION %s '%s'" % (name, code)

            # Add Arguments
            if "arguments" in data:
                osql += " PARAMETERS %s" % str(data["arguments"])

            # Run Command
            client.command(osql)

        # Update Function when Exists
        else:

            # Create Update Command
            osql = "UPDATE Ofunction SET code = '%s'" % code

            # Add Arguments
            if "arguments" in data:
                osql += ", parameters = '%s'" % str(data["arguments"])

            # Define Condition
            osql += " WHERE name = '%s'" % name

            # Run Command
            client.command(oqsl)
```

Using Functions

Once you have created a function, whether using Studio or the OrientDB SQL `CREATE FUNCTION` command, you can begin to use it to operate on the database.

Calling Functions in SQL

Calling custom functions works the same as calling the standard SQL functions that OrientDB ships by default. For instance, consider the earlier example of a `factorial()` function.

```
orientdb> SELECT factorial(5,3)
```

This works well if you want to pass specific values to the function. You can also pass values from the database to the function. For instance, using `sum()` you might want to combine salary fields with bonuses.

```
orientdb> SELECT sum(salary, bonus) AS total FROM Employee
```

Calling Functions from Java

When working in Java you can use functions from OrientDB, both as general functions in your application and working with content from the database. In order to use functions in your application, you first need to retrieve it from the database.

1. Get the reference to the Function Manager
2. Get the function you want to use.
3. Execute the function.

For instance, imagine you want to retrieve the `factorial()` function created in the example. You would need to open a database and retrieve an `OFunction` instance from your database.

```
// Open Database
ODatabaseDocument db = new ODatabaseDocumentTx("plocal:/tmp/db");
db.open("admin", "admin");

// Retrieve Function
OFunction factorial = db.getMetadata().getFunctionLibrary().getFunction("factorial");

// Use Factorial Function
Number result = factorial.execute(24);
```

Alternatively, you can retrieve the function using the Blueprints Graph API:

```
// Retrieve Function
OFunction factorial = graph.getRawGraph().getMetadata().getFunctionLibrary().getFunction("factorial");

// Use Factorial Function
Number result = factorial.execute(24);
```

Whichever approach you use to retrieve a function from OrientDB, you can define the arguments passed to the function by their position or by mapping the argument names to values in a map. You might find this latter method more useful to avoid confusion when working with complex functions that take several arguments, but it works just as well in cases where the function takes a single argument alone.

```
// Report Factorials from List
public void reportFactorials(List<Number> inputValues) {

    // Retrieve Factorial Function
    ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/tmp/db");
    db.open("admin", "admin");
    OFunction factorial = db.getMetadata().getFunctionLibrary().getFunction("factorial");

    // Loop over Input Values
    for (int i = 0; i < inputValues.size(); i++) {

        // Fetch Number
        Number number = list.get(i);
        Map<String, Object> params = new HashMap<String, Object>();
        params.put("num", number);

        // Calculate Factorial
        Number result = factorial(number);

        // Report Results
        System.out.println(String.format("Factorial of %d: %d", number, result));

    }
}
```

Calling Functions from the HTTP REST API

Functions in OrientDB are accessible to REST services. They receive parameters by position through the URL. Beginning in version 2.1, OrientDB also supports defining parameters in JSON through the request payload.

For instance, imagine you created the `factorial()` function used in the previous examples on the `OpenBeer` example database. Using cURL, you can execute the function and retrieve the result of say the factorial of 10:

```
$ curl --user admin:admin \
      http://localhost:2480/function/OpenBeer/factorial/10
{"result":[{"@type":"d","@version":0,"value":3628800.0,"@fieldTypes":"value=d"}]}
```

Similarly, you can perform the same operation passing the arguments in a JSON payload:

```
$ curl --user admin:admin --data '{"num": 10}' \
      http://localhost:2480/function/OpenBeer/factorial
{"result":[{"@type":"d","@version":0,"value":3628800.0,"@fieldTypes":"value=d"}]
```

Whichever method you use, the OrientDB REST API returns an HTTP 202 OK with an envelope containing the results of the operation.

Note that you can only call idempotent functions using the HTTP GET method. When the function is non-idempotent, you need to use the HTTP POST method. When making requests using HTTP POST, encode the content and set the HTTP request header to `"Content-Typpe: application/json"`.

> For more information, see
>
> - HTTP REST Protocol.
> - Server-side Functions

HTTP Return Values

When calling a function through a REST service, OrientDB returns the results as JSON to the client through HTTP. There may be differences in the results, depending on the return value of function.

For instance,

- Function that returns a number:

```
return 31;
```

Would return the result:

```
{"result":[{"@type":"d","@version":0,"value":31}]}
```

- Function that returns a JavaScript object:

```
return {"a":1, "b":"foo"}
```

Would return the result:

```
{"result":[{"@type":"d","@version":0,"value":{"a":1,"b":"foo"}}]}
```

- Function that returns an array:

```
return [1, 2, 3]
```

Would return the result:

```
{"result":[{"@type":"d","@version":0,"value":[1,2,3]}]}
```

- Function that returns a query result:

```
return db.query("SELECT FROM OUser")
```

Would return the result:

```
{
  "result": [
      {
          "@type": "d",
          "@rid": "#6:0",
          "@version": 1,
          "@class": "OUser",
          "name": "admin",
          "password": "...",
          "status": "ACTIVE",
          "roles": [
              "#4:0"
          ],
          "@fieldTypes": "roles=n"
      },
      {
          "@type": "d",
          "@rid": "#6:1",
          "@version": 1,
          "@class": "OUser",
          "name": "reader",
          "password": "...",
          "status": "ACTIVE",
          "roles": [
              "#4:1"
          ],
          "@fieldTypes": "roles=n"
      }
  ]
}
```

Accessing the Database from a Function

When you create a function for OrientDB, it always binds the special variable `orient` to allow you to use OrientDB services from within the function. The most important methods are:

Function	Description
`orient.getGraph()`	Returns the current transactional graph database instance.
`orient.getGraphNoTx()`	Returns the current non-transactional graph database instance.
`orient.getDatabase()`	Returns the current document database instance.

Executing Queries

Queries are idempotent commands. To execute a query from within a function, use the `query()` method. For nstance,

```
return orient.getDatabase().query("SELECT name FROM OUser");
```

Queries with External Parameters

Create a new function with the name `getUserRoles` with the parameter `user`. Then use this code:

```
return orient.getDatabase().query("SELECT roles FROM OUser WHERE name = ?", name );
```

Here, the function binds the `name` parameter as a variable in JavaScript. You can use this variable to build your query.

Executing Commands

OrientDB accepts commands written in any language that the JVM supports. By default, however, OrientDB only supports SQL and JavaScript.

SQL Commands

Execute an SQL command within the function:

```
var gdb = orient.getGraph();
var results = gdb.command( "sql", "SELECT FROM Employee WHERE company = ?", [ "Orient Technologies" ] );
```

The command returns an array of objects:

- When it returns vertices: The result is an `OrientVertex` instance.
- When it returns edges: The result is an `OrientEdge` instance.
- When it returns records: The result is an `OIdentifiable` (or, any subclass of it) instance.

Creating Repository Classes

Functions provide an ideal place for developing the logic your application uses to access the database. You can adopt a Domain-driven design approach, allowing the function to work as a repository, or as a Data Access Object.

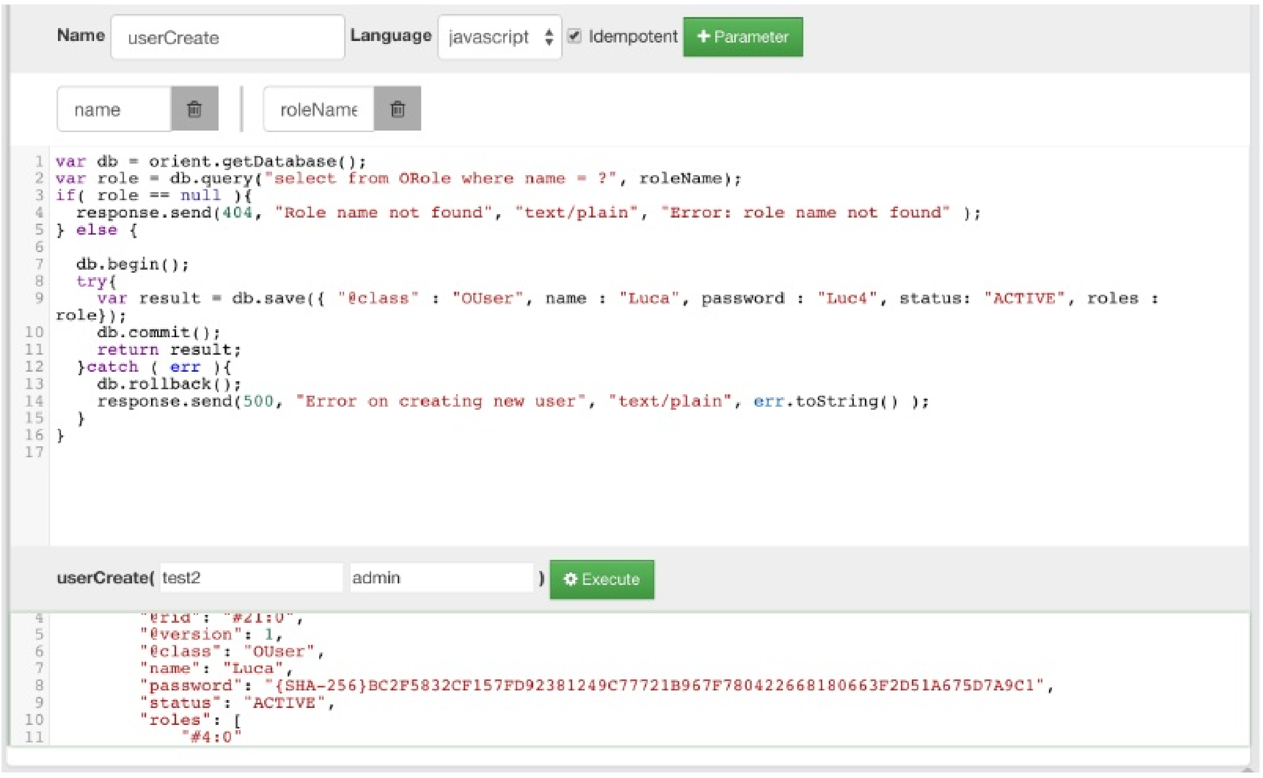
This provides a thin (or thick, if you prefer) layer of encapsulation which may protect you from database changes.

Furthermore, each function is published an dreachable via the HTTP REST protocol, allowing the automatic creation of a RESTful service.

Examples

Below are some examples of functions to build a repository for `OUser` records:

```
function user_getAll() {
   return orient.getDatabase().query("SELECT FROM OUser");
}

function user_getByName( name ){
   return orient.getDatabase().query("SELECT FROM OUser WHERE name = ?", name );
}

function user_getAdmin(){
   return user_getByName("admin");
}

function user_create( name, role ){
   var db = orient.getDatabase();
   var role = db.query("SELECT FROM ORole WHERE name = ?", roleName);
   if( role == null ){
      response.send(404, "Role name not found", "text/plain", "Error: role name not found" );
   } else {

      db.begin();
      try{
         var result = db.save({ "@class" : "OUser", name : "Luca", password : "Luc4", status: "ACTIVE", roles : role});
         db.commit();
         return result;
      }catch ( err ){
         db.rollback();
         response.send(500, "Error on creating new user", "text/plain", err.toString() );
      }
   }
}
```


Name userCreate Language javascript ✓ Idempotent

name roleName

```
var db = orient.getDatabase();
var role = db.query("select from ORole where name = ?", roleName);
if( role == null ){
  response.send(404, "Role name not found", "text/plain", "Error: role name not found" );
} else {

  db.begin();
  try{
    var result = db.save({ "@class" : "OUser", name : "Luca", password : "Luc4", status: "ACTIVE", roles :
role});
    db.commit();
    return result;
  }catch ( err ){
    db.rollback();
    response.send(500, "Error on creating new user", "text/plain", err.toString() );
  }
}
```

userCreate(test2 admin)

```
        "@rid": "#21:0",
        "@version": 1,
        "@class": "OUser",
        "name": "Luca",
        "password": "{SHA-256}BC2F5832CF157FD92381249C77721B967F780422668180663F2D51A675D7A9C1",
        "status": "ACTIVE",
        "roles": [
            "#4:0"
```

Server-side Functions

In OrientDB, you can replace the use of Servlets with server-side functions. For more information on how to call server-side functions, see Functions using the HTTP REST API

When the HTTP REST protocol calls server-side functions, OrientDB embeds a few additional variables:

- **Request Object** The HTTP request, implemented by the `OHttpRequestWrapper` class.
- **Response Object** The HTTP request response, implemented by the `OHttpResponseWrapper` class.
- **Util Object** The utility class with helper functions, to use inside the functions, implemented by the `OFunctionUtilWrapper` class.

Request Object

OrientDB references this object as `request`. For instance,

```
var params = request.getParameters();
```

Method signature	Description	Return type
`getContent()`	Returns the request content.	String
`getUser()`	Gets the request user name.	String
`getContentType()`	Returns the request content type.	String
`getHttpVersion()`	Return the request HTTP version.	String
`getHttpMethod()`	Return the request HTTP method called.	String
`getIfMatch()`	Return the request IF-MATCH header.	String
`isMultipart()`	Returns if the request is multi-part.	boolean
`getArguments()`	Returns the request arguments passed in REST form. Example: `/2012/10/26`.	String[]
`getArgument(<position>)`	Returns the request argument by position, or `null` if not found.	String
`getParameters()`	Returns the request parameters.	String
`getParameter(<name>)`	Returns the request parameter by name or null if not found.	String
`hasParameters(<name>, ...)`	Returns the number of parameters found between those passed.	Integer
`getSessionId()`	Returns the session-id.	String
`getURL()`	Returns the request URL.	String

Response Object

OrientDB references this object as `response`. For instance,

```
var db = orient.getDatabase();
var roles = db.query("select from ORole where name = ?", roleName);
if( roles == null || roles.length == 0 ){
  response.send(404, "Role name not found", "text/plain", "Error: role name not found" );
} else {

  db.begin();
  try{
    var result = db.save({ "@class" : "OUser", name : "Luca", password : "Luc4", "roles" : roles});
    db.commit();
    return result;
  }catch ( err ){
    db.rollback();
    response.send(500, "Error on creating new user", "text/plain", err.toString() );
  }
}
```

Method signature	Description	Return type
`getHeader()`	Returns the response additional headers.	String
`setHeader(String header)`	Sets the response additional headers to send back. To specify multiple headers use the line breaks.	Request object
`getContentType()`	Returns the response content type. If null will be automatically detected.	String
`setContentType(String contentType)`	Sets the response content type. If null will be automatically detected.	Request object
`getCharacterSet()`	Returns the response character set used.	String
`setCharacterSet(String characterSet)`	Sets the response character set.	Request object
`getHttpVersion()`	Returns the HTTP version.	String
`writeStatus(int httpCode, String reason)`	Sets the response status as HTTP code and reason.	Request object
`writeHeaders(String contentType, boolean keepAlive)`	Sets the response headers specifying when using the keep-alive or not.	Request object
`writeLine(String content)`	Writes a line in the response. A line feed will be appended at the end of the content.	Request object
`writeContent(String content)`	Writes content directly to the response.	Request object
`writeRecords(List<OIdentifiable> records)`	Writes records as response. The records are serialized in JSON format.	Request object
`writeRecords(List<OIdentifiable> records, String fetchPlan)`	Writes records as response specifying a fetch-plan to serialize nested records. The records are serialized in JSON format.	Request object
`writeRecord(ORecord record)`	Writes a record as response. The record is serialized in JSON format.	Request object
`writeRecord(ORecord record, String fetchPlan)`	Writes a record as response. The record is serialized in JSON format.	Request object
`send(int code, String reason, String contentType, Object content)`	Sends the complete HTTP response in one call.	Request object
`send(int code, String reason, String contentType, Object content, String headers)`	Sends the complete HTTP response in one call specifying additional headers. Keep-alive is set.	Request object
`send(int code, String reason, String contentType, Object content, String headers, boolean keepAlive)`	Sends the complete HTTP response in one call specifying additional headers.	Request object
`sendStream(int code, String reason, String contentType, InputStream content, long size)`	Sends the complete HTTP response in one call specifying a stream as content.	Request object
`flush()`	Flushes the content to the TCP/IP socket.	Request object

Util Object

OrientDB references this object as `util`. For instance,

```
if( util.exists(year) ){
  print("\nYes, the year was passed!");
}
```

Method signature	Description	Return type
`exists(<variable>)`	Returns trues if any of the passed variables are defined. In JS, for example, a variable is defined if it's not null and not equal to `undefined`.	Boolean

Functions - exists()

This method is called from the `util` object. It returns `true` if the given variable has been defined.

Checking Variables

When developing an application that interacts with OrientDB through functions and the HTTP protocol, you may find this useful in cases where the application does not always define values passed into the function. Using this method, you can check whether the given variable is defined.

Syntax

```
var isDefined = util.exists(<variable>)
```

- **`<variable>`** Defines the variable you want to check.

Return Value

This method returns a boolean value. A value of `true` indicates that the variable is defined. A value of `false` indicates that the variable is `null` or `undefined`.

Functions - flush()

This method is called from the `response` object. It flushes the content to the TCP/IP socket.

Flushing the Response

When your HTTP response is correctly configured, you can use this method to send it to the TCP/IP socket.

Syntax

```
var request = response.flush()
```

Functions - getArgument()

This method is called from the `request` object. It returns the HTTP request argument for the given position.

Retrieving Arguments

Rather than passing data to your function through HTTP request content, you can also pass in arguments through the URL. Using this method you can retrieve a specific argument from the URL determined by its position. To retrieve all URL arguments, use `getArguments()` . To retrieve HTTP request parameters, see `getParameter()` and `getParameters()` .

Syntax

```
var arg = request.getArgument(<position>)
```

- **`<position>`** Defines the position of the argument you want to retrieve.

Return Value

This method returns a string of the argument passed to the function through the HTTP request URL. It determines which argument to return by the given position. If it doesn't find an argument for the given position, it returns `null` .

Functions - getArguments()

This method is called from the `request` object. It returns the HTTP request arguments in REST form.

Retrieving Arguments

Rather than passing data to your function through HTTP request content, you can also pass in arguments through the URL. Using this method you can retrieve the URL arguments into an array for further operations. To retrieve specific arguments by position, use `getArgument()`. To retrieve HTTP request parameters, see `getParameter()` and `getParameters()`.

Syntax

```
var args = request.getArguments()
```

Return Value

This method returns an array of strings that contain each argument passed to the function through the HTTP request.

Functions - getCharacterSet()

This method is called from the `response` object. It returns the character set for the HTTP response.

Retrieving Character Set

When developing server-side functions for applications that interact with OrientDB through the HTTP protocol, you may find it convenient to retrieve or set the character set value on responses, especially if you are using a non-standard character set. Using this method, you can retrieve the character set from the response. To set the character set, see `setCharacterSet()` .

Syntax

```
var char_set = response.getCharacterSet()
```

Return Value

This method returns a string value that provides the HTTP response character set.

Functions - getContent()

This method can be called from the `request` object. It retrieves the content of the request.

Retrieving Content

In developing server-side functions, you may occasionally encounter situations where you need to operate on the content of an HTTP request. Using this method, you can retrieve the content as a string into your function.

Syntax

```
var content = request.getContent()
```

Return Value

This method returns a string of the request content.

Functions - getContentType()

This method is called from both the `request` and `response` objects. It returns the request or response content types.

Retrieving Content Types

When developing applications that interact with OrientDB through HTTP, you may find it useful to create a series of server-side functions to manage the incoming data. Using this function you can determine the content type of the incoming HTTP request or outgoing response. You can use then use this information, for instance, to determine how to parse the content, ensuring that incoming strings of JSON or XML are sent to the appropriate parsers.

To set the content type in HTTP responses, see `setContentType()` .

Syntax

```
var content_type = request.getContentType()

var content_type = response.getContentType()
```

Return Value

This method returns a string containing the content type of the HTTP request or response.

Functions - getIfMatch()

This method is called from the `request` object. It returns the `If-Match` value of the HTTP request.

Retrieving If-Match Values

When developing applications that interact with OrientDB through the HTTP protocol, you may sometimes find it useful to use `If-Match` HTTP requests, such as in cases where you want to use conditions to determine how your function handles the incoming request beyond content type and method.

Syntax

```
var ifmatch = request.getIfMatch()
```

Return Value

This method returns a string containing the `If-Match` value from the HTTP request.

Functions - getHeader()

This method is called from the `response` object. It returns a string of the additional response headers.

Retrieving Headers

With server-side functions, you may occasionally want to operate on the HTTP responses your function returns. Using this method, you can retrieve the response headers. You may find this useful in situations where you need to log information from the response header or where you want to modify the value the function returns. To set the response to additional headers, see `setHeader()`.

Syntax

```
var header = response.getHeader()
```

Return Value

This method returns a string containing the additional headers in the response.

Functions - getHttpMethod()

This method is called from the `request` object. It returns the HTTP protocol method used by the request.

Retrieving HTTP Methods

When developing applications that interact with OrientDB through the HTTP protocol, you may find it useful to retrieve the HTTP method from requests. For instance, the same function might use the request content to run a series of `SELECT` statements when set with a `GET` method or use it to run `UPDATE` statements in responding to a `POST` method. Using this method, you can retrieve the HTTP method sent with the request.

Syntax

```
var method = request.getHttpMethod()
```

Return Value

This method returns a string that provides the HTTP method of the request.

Function - getHttpVersion()

This method is called from either the `request` or `response` objects. It returns the HTTP protocol version of the request or response.

Retrieving HTTP Version

When developing an application that interacts with OrientDB through the HTTP protocol, you may find it useful to check the HTTP version used in requests or responses. Especially on cases where you're relying on features introduced in more recent versions.

Syntax

```
var version = request.getHttpVersion()

var version = response.getHttpVersion()
```

Return Value

This method returns a string that provides the HTTP protocol version of the request or response.

Functions - getParameters)

This method is called from the `request` object. It returns the value of the given HTTP request parameter.

Retrieving Parameters

When developing applications that interact with OrientDB through the HTTP protocol, you may want to retrieve parameters from HTTP requests to operate on in your functions. Using this method, you can retrieve the value of a specific parameter by name. If you would like to operate on the parameters collectively, use `getParameters()`. To operate on arguments, use `getArguments()` or `getArgument()`.

Syntax

```
var value = request.getParameter(<name>)
```

Return Value

This method returns a string value of the given HTTP request parameter.

Functions - getParameters()

This method is called from the `request` object. It returns the HTTP request parameters.

Retrieving Parameters

When developing applications that interact with OrientDB through the HTTP protocol, you may want to retrieve parameters from HTTP requests to operate on in your functions. Using this method, you can retrieve all the parameters sent in the request. If you would like to operate on a specific parameter by name, use `getParameter()`. To operate on arguments, use `getArguments()` or `getArgument()`.

Syntax

```
var params = request.getParameters()
```

Return Value

This method returns a string of the HTTP request parameters.

Functions - getSessionId()

This method is called by the `request` object. It returns the Session ID of the session executing the function.

Retrieving Session ID's

Using this method you can retrieve the current Session ID into your function.

Syntax

```
var id = request.getSessionId()
```

Return Value

This method returns a string containing the current Session ID.

Functions - getURL()

This function is called from the `request` object. It returns the URL of the HTTP request.

Retrieving URL's

When developing applications that interact with OrientDB through HTTP, you may find it useful to retrieve the URL of the HTTP request, either for logging purposes or to control what further operations the function calls.

Syntax

```
var url = request.getURL()
```

Return Value

This method returns a string of the URL in the HTTP request.

Functions - getUser()

This function is called from the `request` object. It returns a string of the requesting user's name.

Retrieving Users

Functions execute on OrientDB under specific users. Using this method you can determine which user executes the function. You may find this useful in logging operations or when building responses for functions that fail due to the user being unprivileged.

Syntax

```
var username = request.getUser()
```

Return Value

This method returns a string that provides the username of the requesting user.

Functions - hasParameters()

This method is called from the `request` object. It returns the number of given parameters found in the HTTP request.

Checking Parameters

When developing applications that interact with OrientDB through the HTTP protocol, you may find it useful to check in your function whether certain parameters were sent. This method takes a series of parameters as arguments, it then counts the number of parameters for which it finds values set in the HTTP request.

Syntax

```
var count = request.hasParameters(<name>, ...)
```

- **`<name>`** Defines a parameter you want to check the HTTP request for, a string value. Include as many parameters as you want to check.

Return Value

This method returns an integer value indicated the number of the given parameters that it found in the HTTP request.

Functions - isMultipart()

This method is called from the `request` object. It determines whether or not an HTTP request is multi-part.

Multi-part HTTP Requests

In developing applications that interact with OrientDB through the HTTP protocol, you may find it useful to check whether the incoming request in multi-part. If your applications sometimes sends multi-part requests, you may find it useful to check, ensuring that you handle the incoming data appropriately.

Syntax

```
var isMulti = request.isMultiaprt()
```

Return Value

This method returns a boolean value, where a value of `true` indicates that the HTTP request is multi-part.

Functions - send()

This method is called from the `response` object. It sends the HTTP response to your application.

Sending HTTP Responses

When developing an application that interacts with OrientDB through functions and the HTTP protocol, eventually you'll need to send the response back to the requesting application. Using this method, you can trigger OrientDB to issue the response.

Syntax

```
var request = response.send(<code>, <reason>,
    <content-type>, <content>)
var request = response.send(<code>, <reason>,
    <content-type>, <content>, <headers>)
var request = response.send(<code>, <reason>,
    <content-type>, <content>, <headers>,
    <keep-alive>)
```

- **`<code>`** Defines the HTTP response status code, as an integer.
- **`<reason>`** Defines the HTTP response status reason, as a string.
- **`<content-type>`** Defines the HTTP response content type, as a string.
- **`<content>`** Defines the HTTP response content, as a string.
- **`<headers>`** Defines the HTTP response headers, as a string.
- **`<keep-alive>`** Defines whether you want to use a keep alive with the response, as a boolean.

Return Value

This method returns the HTTP request object.

Functions - sendStream()

This method is called from the `response` object. It sends the HTTP response to your application as a stream.

Sending HTTP Responses

When developing an application that interacts with OrientDB through functions and the HTTP protocol, eventually you'll need to send the response back to the requesting application. Using this method, you can trigger OrientDB to issue the response as a stream.

Syntax

```
var request = response.sendStream(<code>, <reason>, <content-type>, <content>, <size>)
```

- **`<code>`** Defines the HTTP response status code, as an integer.
- **`<reason>`** Defines the HTTP response status reason, as a string.
- **`<content-type>`** Defines the HTTP response content type, as a string.
- **`<content>`** Defines the HTTP response content, as an input stream.
- **`<size>`** Defines the size of the stream, as a long integer.

Return Value

This method returns the HTTP request object.

Functions - getCharacterSet()

This method is called from the `response` object. It sets the character set for the HTTP response.

Retrieving Character Set

When developing server-side functions for applications that interact with OrientDB through the HTTP protocol, you may find it convenient to retrieve or set the character set value on responses, especially if you are using a non-standard character set. Using this method, you can set the character set for the response. To retrieve the character set, see `getCharacterSet()`.

Syntax

```
var res = response.setCharacterSet(<character-set>)
```

- **`<character-set>`** Defines the character set for the HTTP response, a string value.

Return Value

This method returns the updated response object.

Functions - setContentType()

This method is called from the `response` object. It sets the content type for the HTTP response.

Setting Content Type

When developing applications that interact with OrientDB through the HTTP protocol, you may find it useful to set the content type in the responses. This allows you to tell your application how you want it to handle the return content, ensuring that it in turn passes say XML or JSON data to the appropriate parsers.

Syntax

```
var obj = response.setContentType(<type>)
```

- **`<type>`** Defines the content type you want to set, as a string.

Return Value

This method returns the updated response object.

Functions - getHeader()

This method is called from the `response` object. It sets the additional headers in the response.

Setting Headers

With server-side functions, you may occasionally want to operate on the HTTP responses your function returns. Using this method, you can set the response headers. You may find this useful in situations where you need to log information from the response header or where you want to modify the value the function returns. To retrieve the response to additional headers, see `getHeader()`.

Syntax

```
var obj = response.setHeader(<header>)
```

- **`header`** Defines the string value of the header you want to set.

Return Value

This method returns the updated request object.

Functions - writeContent()

This method is called by the `response` object.

Writing Content

Using this method you can define the HTTP response content. This allows you to define and set the data that you return to your application from OrientDB.

Syntax

```
var response = response.writeContent(<content>)
```

- **`<content>`** Defines the string for the HTTP response content.

Return Value

This method returns the HTTP response object.

Functions - writeHeaders()

This method is called from the `response` object. It sets the content type for the HTTP response.

Writing Headers

Similar to `setContentType()`, this method allows you to set the content type for the HTTP response. This allows you to have OrientDB tell your application how it should prase the response content. Also, allows you to enable the keep-alive.

Syntax

```
var response = response.writeHeaders(<type>, <keep-alive>)
```

- **`<type>`** Defines the content type for the HTTP response. Set as a string variable.
- **`<keep-alive>`** Defines whether you want to use the keep-alive. Set as a boolean value. Defaults to `false`.

Return Value

This method returns the response object.

Functions - writeLine()

This method is called from the `response` object. It writes the given text to the end of the content.

Writing Lines

Using this method you can append lines to the end of the HTTP response content. You may find it useful when adding content in a loop.

Syntax

```
var response = response.writeLine(<line>)
```

- **`<line>`** Defines the string you want to add to the content.

Return Value

This method returns the HTTP response object.

Functions - writeRecord()

This method is called from the `response` object. It writes an OrientDB record to the HTTP response.

Writing Records

When developing applications that interact with OrientDB through functions and the HTTP protocol, you may want to return database records to your application. Using this method, you can add a an individual record identified by its Record ID's to the HTTP response. The record is serialized into the HTTP response using the JSON format.

Syntax

```
var response = response.writeRecord(<record>)

var response = response.writeRecord(<record>, <fetch-plan>)
```

- **`<records>`** Defines the Record ID's for the record you want to write.
- **`<fetch-plan>`** Defines a string to serve as a fetching strategy to determine how the method retrieves the record.

Return Value

This method returns the HTTP response object.

Functions - writeRecords()

This method is called from the `response` object. It writes a list of OrientDB records to the HTTP response.

Writing Records

When developing applications that interact with OrientDB through functions and the HTTP protocol, you may want to return database records to your application. Using this method, you can add a series of records identified by their Record ID's to the HTTP response. The records are serialized into the HTTP response using the JSON format.

Syntax

```
var response = response.writeRecords(<records>)

var response = response.writeRecords(<records>, <fetch-plan>)
```

- **`<records>`** Defines an array of Record ID's for the records you want to write.
- **`<fetch-plan>`** Defines a string to serve as a fetching strategy to determine how the method retrieves the records.

Return Value

This method returns the HTTP response object.

Functions - writeStatus()

This method is called from the `response` object. It sets the HTTP response status code and reason.

Setting HTTP Response Status

In the event that you are developing an application that uses complex HTTP responses from OrientDB to determine how it handles the response content, you can use this method to set the HTTP response status code and reason.

Syntax

```
var res = response.writeStatus(<code>, <reason>)
```

- **`<code>`** Defines the response status code as an integer value, such as `404` or `200`.
- **`<reason>`** Defines the response status text.

Return Value

This method returns the response object.

Database Functions

In previous exmaples, such as `factorial()`, the function is relatively self-contained. It takes an argument from the query, operates on it, and returns the result. This is useful in cases where you need to perform some routine arithmetic operation or manipulate strings in a consistent way, but you can also use functions to perform more complex database operations. That is, the function can receive arguments, interact with the database, then return the results of that interaction.

Database Variable

When you create a function, OrientDB always binds itself to the `orient` variable. Using this variable you can call methods to access, query and operate on the database from the function. The specific method called to access the database depends on the type of database you're using:

Function	Description
`orient.getGraph()`	Retrieves the current Transactional Graph Database instance
`orient.getGraphNoTx`	Retrieves the current Non-transactional Graph Database instance
`orient.getDatabase()`	Retrieves the current Document Database instance

For instance, say you wanted a function to create the given user on the database. You might create one that takes three arguments: `userName`, `passwd` and `roleName`.

```
// Fetch Database
var db = orient.getDatabase();
var role = db.query("SELECT FROM ORole WHERE name = ?", roleName);

if (role == null){
    response.send(404, "Role name not found", "text/plain",
        "Error: Role name not found");
} else {
    db.begin();
    try {
        var result = db.save({"@class", name: userName, password: passwd,
            status: "ACTIVE", roles: role});
            db.commit();
            return result;
    } catch(err){
        db.rollback();
        response.send(500, "Error creating new user", "text/plain",
            err.toString());
    }
}
```

Methods

As demonstrated in the example above, once you've retrieved the database interface, you can begin to call a series of additional methods to operate on the database from within the function.

Method	Description
`addEdge()`	Adds edges to the graph
`addVertex()`	Adds vertices to the graph
`command()`	Issues SQL command
`delete()`	Removes records
`getEdge()`	Retrieves edges
`getVertex()`	Retrieves vertices
`load()`	Retrieves records
`query()`	Queries the database
`removeEdge()`	Removes edges from a graph
`removeVertex()`	Removes vertices from a graph

Database Methods

Method	Description
`isUseLightweightEdges()`	Check if database uses Lightweight Edges
`open()`	Opens the database
`close()`	Closes the database
`setUseLightweightEdges()`	Enable or disable the use of Lightweight Edges

Class Methods

Method	Description
`browseClass()`	Returns all records in a class
`countClass()`	Counts records in given class
`createEdgeType()`	Creates a new class for edges
`createVertexType()`	Creates a new class for vertices
`dropEdgeType()`	Removes an edge class
`dropVertexType()`	Removes a vertex class
`getEdgeBaseType()`	Retrieves the base class for edges, which is `E` by default
`getEdgeType()`	Retrieves the given edge class
`getVertexBaseType()`	Retrieves the base class for vertices, which is `V` by default
`getVertexType()`	Retrieves the given edge class

Cluster Methods

Method	Description
`browseCluster()`	Returns all records in a cluster
`dropCluster()`	Removes cluster
`getClusterIdByName()`	Retrieves the Cluster ID for the given cluster
`getClusterNameById()`	Retrieves logical cluster name
`getClusterNames()`	Retrieve cluster names
`getClusterRecordSizeById()`	Retrieves the number of records in a cluster
`getClusterRecordSizeByName()`	Retrieves the number of records in a cluster
`getClusters()`	Retrieves clusters

Transaction Methods

Method	Description
`begin()`	Initiates a transaction
`commit()`	Commits a transaction
`isAutoStartTx()`	Checks whether transaction auto-start is enabled
`rollback()`	Reverts a transaction
`setAutoStartTx()`	Enables transaction auto-start

User Methods

Method	Description
`getUser()`	Retrieves the current user
`setUser()`	Sets the user

Size Methods

Method	Description
`countClass()`	Counts records in given class
`countEdges()`	Counts edge records
`countVertices()`	Counts vertex records

Functions - addEdge()

This method adds an edge to the database.

Adding Edges

When working with a Graph database, whether transactional or non-transactional, when you want to add an edge to the graph, use this method. This adds a new edge record to the database and ensures that the relevant vertices are also updated, keeping the graph consistent.

Syntax

```
var edge = db.addEdge(<record-id>, <out>, <in>, <label>)
```

- `<record-id>` Defines the Record ID.
- `<out>` Defines the out vertex.
- `<in>` Defines the in vertex.
- `<label>` Defines a name or label for the edge.

> **NOTE**: You can use the `getVertex()` method to retrieve the in and out vertices to connect the edge.

Example

Imagine a social networking application that

Functions - addVertex()

This method adds a vertex to the database.

Adding Vertices

When working with a Graph database, whether transactional or non-transactional, when you want to add a vertex to the graph, use this method. This adds a new vertex record to the database and ensures that the relevant edges are also updated, keeping the graph consistent.

To add an edge from within your function, see `addEdge()`.

Syntax

```
var vertex = db.addVertex(<record-id>)
```

- **`<record-id>`** Defines the Record ID.

Functions - begin()

This method initiates a transaction within the function.

Using Transactions

When using a transactional database, which are retrieved using the methods `orient.getGraph()` or `orient.getDatabase()` and not `orient.getGraphNoTx()` , you can manage transactions within your functions. You may find this useful in situations where you want the function to rollback operations when it encounters conflicts or similar problems.

For more information on transactional methods, see `commit()` and `rollback()` .

Syntax

```
db.begin()
```

Functions - browseClass()

This method returns all records of a given class.

Browsing Classes

OrientDB borrows from the Object Oriented Programming paradigm the concept of a class. Where a cluster defines where the database stores records, classes define what kind of records you want to store. Occasionally, you may find it useful to retrieve all records by class. For instance, you may have a function to perform routine security checks on users with access to your database. Calling this method on the `OUser` and `ORole` classes can retrieve records on all database users and allow you to determine the resources they can access by role.

Syntax

```
var records = db.browseClass(<name>, <polymorphic>)
```

- **`<name>`** Defines the class name.
- **`<polymorphic>`** Defines whether you want the method to retrieve subclasses. By default, it is `false` .

Functions - browseCluster()

This method returns all records in the given cluster.

Browsing Clusters

In OrientDB, a cluster defines where the database stores records. When using a single instance of OrientDB Server, this is whether it's storing the records in memory or where the database looks on your file system. In distributed deployments, it relates to which server instance has the records you want.

Using this method, you can retrieve all records of a given cluster. You may find it useful in situations where you use some logical organizational structure for your data. For instance, if you have an `Account` class for your customers, you might organize clusters by the regions in which you do business. In a function you might take a zip code as an argument and use some logic to determine which cluster for the `Account` class handles the region in which the user is located, then return all accounts for that region.

Syntax

```
var records = db.browseCluster(<name>)
```

- **`<name>`** Defines the cluster name.

Function - close()

This method closes the database connection.

Closing Connections

Occasionally, you may need to close a database connection from within a function, either as part of the operation or to close the connection to a different database.

Syntax

```
db.close()
```

Functions - command()

This methods issues an OrientDB SQL command to the database.

Issuing Commands

Using this method, you can issue OrientDB SQL commands to the database from within your function. You can also pass parameters with the command string to dynamically set variables.

Syntax

```
var results = db.command(<sql>, [<param>, ...])
```

- `<sql>` Defines the OrientDB SQL command.
- `<param>` Defines parameters to substitute in the command, (if any).

Functions - commit()

This method commits a transaction within the function.

Using Transactions

When using a transactional database, which are retrieved using the methods `orient.getGraph()` or `orient.getDatabase()` and not `orient.getGraphNoTx()` , you can manage transactions within your functions. You may find this useful in situations where you want the function to rollback operations when it encounters conflicts or similar problems.

For more information on transactional methods, see `begin()` and `rollback()` .

Syntax

```
db.commit()
```

Functions - countClass()

This method counts the records in the given class.

Counting Records

You may want to check the size of your database from certain angles. Using this method you can find the number of records stored on a particular class. This can be useful in logging or reporting operations. For instance, you might use this with a social networking application to count the number of users registered with the service.

Syntax

```
var count = db.countClass(<class>)
```

- `<class>` Defines the class name.

Functions - countEdges()

This method counts the number of edge records.

Counting Records

On occasion you may find it useful to retrieve or reference the number of records in a database. Using this method, you can count the number of edge records in a database. By default, it returns the records for the edge base class `E`. You can retrieve a count of records in subclasses of `E` by passing the class name as an argument.

Syntax

```
var count = db.countEdges(<class>)
```

- **`<class>`** Defines the class you want to count. Defaults to the base edge class, `E`.

Functions - countEdges()

This method counts the number of vertex records.

Counting Records

On occasion you may find it useful to retrieve or reference the number of records in a database. Using this method, you can count the number of vertex records in a database. By default, it returns the records for the vertex base class `V`. You can retrieve a count of records in subclasses of `V` by passing the class name as an argument.

Syntax

```
var count = db.countVertices(<class>)
```

- **`<class>`** Defines the class you want to count. Defaults to the base edge class, `V`.

Functions - createEdgeType()

This method creates a database class that extends `E`.

Creating Vertex Classes

Classes in OrientDB define the type of records you want to store. Edges in OrientDB belong to the `E` class or to a class that inherits from `E`. Using this method, you can create an edge class on your database. It creates the class and performs additional operations so that the new class extends the base edge class `E`.

Syntax

```
var newClass = db.createEdgeType(<class>, <superclass>)
```

- **`<class>`** Defines the class you want to create.
- **`<superclass>`** Defines the class you want to extend, defaults to the `E` class.

Functions - createVertexType()

This method creates a database class that extends `V`.

Creating Vertex Classes

Classes in OrientDB define the type of records you want to store. Vertices in OrientDB belong to the `V` class or to a subclass of `V`. Using this method, you can create a vertex class on your database. It creates the class and performs additional operations so that the new class extends the base vertex class `V`.

Syntax

```
var newClass = db.createVertexType(<class>, <superclass>)
```

- **`<class>`** Defines the class you want to create.
- **`<superclass>`** Defines the class you want to extend, defaults to the `V` class.

Functions - delete()

This method removes the given record from the database.

Removing Records

OrientDB provides an SQL command called `DELETE` to remove records from the database. However, sometimes you may want to do something a little more dynamic than simple deletion. For instance, a conditional deletion, where the record is only removed when certain conditions in other records are met.

> Bear in mind that edges and vertices require additional operations to remove. If you remove either using this method, it can lead to inconsistencies in your graph. Instead use `removeVertex()` and `removeEdge()` for these record types.

Syntax

```
db.delete(<rid>)
```

- **`<rid>`** Defines the Record ID.

Functions - dropCluster()

Removes the given cluster from the database.

Removing Clusters

Clusters define where in the database OrientDB stores records. They can be persistent and written to disk or volatile and stored in memory. The first number in a Record ID refers to the Cluster ID, an integer used to identify the cluster internally. Using this method, you can remove a cluster from the database. You can also define whether you want to remove the cluster outright or only remove records from the cluster.

Syntax

```
db.dropCluster(<cluster-id>, <truncate>)
```

- `<cluster-id>` Defines the Cluster ID.
- `<truncate>` Defines whether you want to truncate the cluster rather than removing it. Truncating means that you remove the records from the cluster, while leaving it configured on the database for later use.

Functions - dropEdgeType()

This method removes an edge class from the database.

Removing Classes

In cases where you have deprecated or otherwise no longer require an edge class on the database, you remove the class using this function.

Syntax

```
db.dropEdgeType(<class>)
```

- `<class>` Defines the class you want to remove.

Functions - getClusterIdByName()

This method retrieves the Cluster ID for the given cluster name.

Retrieving Cluster ID's

In OrientDB, each record on the database has a numerical identifier called a Record ID, for instance #5:32. The first element in a Record ID is the Cluster ID, the second element is the record's position in the cluster. The Cluster ID is an integer used to identify the cluster internally.

Using this method, you can pass a cluster's name as a string and retrieve the integer for the Cluster ID. You may find this useful in any operation where you need the Cluster ID as an argument.

Syntax

```
var clusterId = db.getClusterIdByName(<name>)
```

- `<name>` Defines the cluster name.

Functions - getClusterNameById()

This method returns the logical cluster name for the given Cluster ID.

Retrieving Cluster Names

In OrientDB, each record on the database has a numerical identifier called a Record ID, for instance #5:32. The first element in a Record ID is the Cluster ID, the second element is the record's position in the cluster. The Cluster ID is an integer used to identify the cluster internally.

Using this method, you can take a given Cluster ID and pass it as an argument to retrieve the logical name for a cluster. You may find it useful when operating on clusters by Cluster ID, as it allows you to fetch a more human readable name for logs.

Syntax

```
var name = db.getClusterNameById(<id>)
```

- `<id>` Defines the Cluster ID.

Functions - getClusterNames()

This method retrieves the names of all clusters in the database.

Retrieving Cluster Names

In OrientDB, a cluster defines where the database stores records. When using a single instance of OrientDB Server, this is whether it's storing the records in memory or where the database looks on your file system. In distributed deployments, it relates to which server instance has the records you want.

Using this method you can retrieve the names of all clusters configured on your database. You may find this useful in functions where you need to operate over several clusters at a time or check for several in particular.

Syntax

```
var names = db.getClusterNames()
```

Functions - getClusterRecordSizeById()

This method returns the number of records in a cluster identified by its Cluster ID.

Retrieving Cluster Counts

OrientDB uses clusters to determine where it stores records. Using this method you can determine the number of records in a cluster. You may find this useful in any sort of counting function to check and report the size of a cluster.

Syntax

```
var size = db.getClusterRecordsById(<id>)
```

- `<id>` Defines the Cluster ID.

Functions - getClusterSizeByName()

This method returns the number of records in a cluster identified by its logical name.

Retrieving Cluster Counts

OrientDB uses clusters to determine where it stores records. Using this method you can determine the number of records in a cluster. You may find this useful in any sort of counting function to check and report the size of a cluster.

Syntax

```
var count = db.getClusterRecordSizeByName(<name>)
```

- `<name>` Defines the cluster name.

Functions - getClusters()

This method retrieves clusters from the database.

Retrieving Clusters

Syntax

```
var clusters = db.getClusters()
```

Functions - getEdge()

This method retrieves an edge.

Retrieving Edges

When you have the Record ID for a given edge and want to retrieve it into the function to operate on, you can do so using this method. It is comparable to the `load()` method on document databases and the `getVertex` method with vertices.

Syntax

```
var edge = db.getEdge(<record-id>)
```

- **`<record-id>`** Defines the Record ID.

Functions - getEdgeBaseType()

This method returns the OrientDB base class for edges, which by default is the `E` class.

Retrieving Classes

OrientDB borrows from the Object Oriented Programming paradigm the concept of a class. Where a cluster indicates where you want to store a record, classes indicate the types of records you want to store. Classes can have inherit from super classes and themselves have subclasses.

In a graph database, the base class for an edge, which is the `E` class. This method retrieves `E` into your function, allowing you to operate on the class.

Syntax

```
var eclass = db.getEdgeBaseType()
```

Functions - getEdgeType()

This methods retrieves the given edge class.

Retrieving Classes

OrientDB borrows the concept of class from the Object Oriented Programming paradigm. Classes can have subclasses and inherit from superclasses. An edge class is any that extends the base edge class `E`.

Using this method, you can retrieve the given edge class instance, allowing you to operate on the class from within your function.

Syntax

```
var eclass = db.getEdgeType(<class>)
```

- **`<class>`** Defines the name of the class.

Functions - getUser()

This method retrieves the current user.

Retrieving Users

When you operate on the database you access it through a particular database users. With this method, you can retrieve the current user from within the function. You may find it useful in logging or when you need the function to operate on a particular user.

Syntax

```
var user = db.getUser()
```

Functions - getVertex()

This method retrieves a vertex from the database.

Retrieving Vertices

When you have the Record ID for a given vertex and want to retrieve it into the function to operate on, you can do so using this method. It is comparable to the `load()` method on document databases. To retrieve an edge, see `getEdge()` .

Syntax

```
var vertex = db.getVertex(<record-id>)
```

- **`<record-id>`** Defines the Record ID.

Functions - getVertexBaseType()

This method returns the OrientDB base class for vertices, which by default is the `V` class.

Retrieving Classes

OrientDB borrows from the Object Oriented Programming paradigm the concept of a class. Where a cluster indicates where you want to store a record, classes indicate the types of records you want to store. Classes can have inherit from super classes and themselves have subclasses.

In a graph database, the base class for a vertex is the `V` class. This method retrieves `V` into your function, allowing you to operate on the class.

Syntax

```
var vclass = db.getVertexBaseType()
```

Functions - getVertexType()

This methods retrieves the given vertex class.

Retrieving Classes

OrientDB borrows the concept of class from the Object Oriented Programming paradigm. Classes can have subclasses and inherit from superclasses. A vertex class is any that extends the base vertex class `V`.

Using this method, you can retrieve the given vertex class instance, allowing you to operate on the class from within your function.

Syntax

```
var vclass = db.getVertexType(<class>)
```

- **`<class>`** Defines the name of the class.

Functions - isAutoStartTx()

This method shows whether the database is configured to auto-start transactions.

Auto-Starting Transactions

OrientDB can auto-start transactions when a client begins operations on the database. Using this function you can determine whether this feature is currently active on the database. To enable or disable transaction auto-start from a function, use the `setAutoStartTx()` function.

Syntax

```
var isEnabled = db.isAutoStartTx()
```

Functions - isUseLightweightEdges()

This method determines whether the database uses lightweight edges.

Using Lightweight Edges

When you create an edge in OrientDB, there are two modes in which it's stored on the database. With regular edges, the edge is a separate record that indicates the connecting vertices and any other informtion you would like to store on it. By contrast, databases using Lightweight Edges don't store the edge as a separate record. Instead, it stores the edge as properties on the connecting vertices.

In cases where the use of Lightweight Edges affects your function, you can use this method to determine whether it's enabled for the database. To change the setting, see the `setUseLightweightEdges()` method.

Syntax

```
var isEnabled = db.isUseLightweightEdges()
```

Functions - load()

This method retrieves the specified record from the database.

Loading Records

When you have the Record ID for a given record and want to retrieve it to operate on from your function, you can get it using this method and set it onto a variable. It is comparable to `getVertex()` and `getEdge()` on graph databases.

Syntax

```
var record = db.load(<rid>, <fetch-plan>, <cache>)
```

- **`<rid>`** Defines the Record ID.
- **`<fetch-plan>`** Defines the Fetching Strategy you want to use. By default, method only fetches the given record.
- **`<cache>`** Defines whether you want to ignore cached result-sets. By default, this is set to `false`.

Functions - open()

This method opens the database using the given credentials.

Opening Databases

Using this method you can open a database and authenticate the connection to a particular user. You might find it useful in cases where you want the function to open the database for you, or when you need to operate briefly on a different database.

Syntax

```
db.open(<user>, <passwd>)
```

- `<user>` Defines the username.
- `<passwd>` Defines the user password.

Functions - query()

This method executes an OrientDB SQL query on the database.

Querying from Functions

There are times when you may need to execute queries within your function. For instance, if you need to check certain conditions on the database to determine what your function returns or you might want to use the function to perform some preprocessing operations before inserting records onto OrientDB.

Syntax

```
var result = db.query(<sql>)
```

- `<sql>` Defines the OrientDB SQL command to execute.

Functions - removeEdge()

This method removes the given edge from the graph.

Removing Vertices

While you can query graph records in the same manner you would query document records, removing them can sometimes require a number of additional steps to update the connecting records and keep the graph consistent. OrientDB provides special functions for vertices and edges to ensure this update happens when you remove records in a graph.

> When removing records in a document database, you can use the standard `delete()` function. To remove a vertex, use `removeVertex()`.

Syntax

```
db.removeEdge(<edge>)
```

- `<edge>` Defines the edge you want to remove.

Functions - removeVertex()

This method removes the given vertex from the graph.

Removing Vertices

While you can query graph records in the same manner you would query document records, removing them can sometimes require a number of additional steps to update the connecting records and keep the graph consistent. OrientDB provides special functions for vertices and edges to ensure this update happens when you remove records in a graph.

> When removing records in a document database, you can use the standard `delete()` function. To remove an edge, use `removeEdge()`.

Syntax

```
db.removeVertex(<vertex>)
```

- **`<vertex>`** Defines the vertex you want to remove.

Functions - rollback()

This method cancels a transaction within the function, reverting the database to its earlier state.

Using Transactions

When using a transactional database, which are retrieved using the methods `orient.getGraph()` or `orient.getDatabase()` and not `orient.getGraphNoTx()`, you can manage transactions within your functions. You may find this useful in situations where you want the function to rollback operations when it encounters conflicts or similar problems.

For more information on transactional methods, see `begin()` and `commit()`.

Syntax

```
db.rollback()
```

Functions - setUser()

This method sets the user for the database.

Setting Users

When you operate on the database you access it through a particular database users. With this method, you can define the user from within the function.

Syntax

```
db.setUser(<user>)
```

- `<user>` Defines the user.

Functions - setAutoStartTx()

This method enables or disables transaction auto-start.

Auto-Starting Transactions

OrientDB can auto-start transactions when a client begins operations on the database. Using this method you can enable or disable transaction auto-start. To check its current state, use the `isAutoStartTx()` function

Syntax

```
db.setAutoStartTx(<state>)
```

- **`<state>`** Defines whether you want to enable transaction auto-start.

Functions - setUseLightweightEdges()

This method enables or disables the use of lightweight edges on the database.

Using Lightweight Edges

When you create an edge in OrientDB, there are two modes in which it's stored on the database. With regular edges, the edge is a separate record that indicates the connecting vertices and any other informtion you would like to store on it. By contrast, databases using Lightweight Edges don't store the edge as a separate record. Instead, it stores the edge as properties on the connecting vertices.

In cases where you need to change how the database handles edges, you can use this method to enable or disable the feature on the database. To check the current use of lightweight edges, see the `isUseLightweightEdges()` method.

Syntax

```
db.setUseLightweightEdges()
```

Plugins

If you're looking for drivers or JDBC connector go to Programming-Language-Bindings.

- OrientDB Spring Data is the official Spring Data Plugin for both Graph and Document APIs
- spring-orientdb is an attempt to provide a PlatformTransactionManager for OrientDB usable with the Spring Framework, in particular with @Transactional annotation. Apache 2 license
- Spring Session OrientDB is a Spring Session extension for OrientDB.

- Play Framework 2.1 PLAY-WITH-ORIENTDB plugin
- Play Framework 2.1 ORIGAMI plugin
- Play Framework 1.x ORIENTDB plugin
- Frames-OrientDB Plugin Play Framework 2.x Frames-OrientDB plugin is a Java O/G mapper for the OrientDB with the Play! framework 2. It is used with the TinkerPop Frames for O/G mapping.

With proper mark-up/logic separation, a POJO data model, and a refreshing lack of XML, Apache Wicket makes developing web-apps simple and enjoyable again. Swap the boilerplate, complex debugging and brittle code for powerful, reusable components written with plain Java and HTML.

Guice (pronounced 'juice') is a lightweight dependency injection framework for Java 6 and above, brought to you by Google. OrientDB Guice plugin allows to integrate OrientDB inside Guice. Features:

- Integration through guice-persist (UnitOfWork, PersistService, @Transactional, dynamic finders supported)
- Support for document, object and graph databases
- Database types support according to classpath (object and graph db support activated by adding jars to classpath)
- Auto mapping entities in package to db scheme or using classpath scanning to map annotated entities
- Auto db creation
- Hooks for schema migration and data initialization extensions
- All three database types may be used in single unit of work (but each type will use its own transaction)

Vert.x is a lightweight, high performance application platform for the JVM that's designed for modern mobile, web, and enterprise applications. Vert.x Persistor Module for Tinkerpop-compatible Graph Databases like OrientDB.

Gephi Visual tool usage with OrientDB and the Blueprints importer

---

OrientDB session store for Connect

Puppet module

Chef

Apache Tomcat realm plugin by Jonathan Tellier

Shibboleth connector by Jonathan Tellier. The Shibboleth System is a standards based, open source software package for web single sign-on across or within organizational boundaries. It allows sites to make informed authorization decisions for individual access of protected online resources in a privacy-preserving manner

---

Griffon plugin, Apache 2 license

JCA connectors

- OPS4J Orient provides a JCA resource adapter for integrating OrientDB with Java EE 6 servers
- OrientDB JCA connector to access to OrientDB database via JCA API + XA Transactions

---

Pacer plugin by Paul Dlug. Pacer is a JRuby graph traversal framework built on the Tinkerpop stack. This plugin enables full OrientDB graph support in Pacer.

EventStore for Axonframework, which uses fully transactional (full ACID support) NoSQL database OrientDB. Axon Framework helps build scalable, extensible and maintainable applications by supporting developers apply the Command Query Responsibility Segregation (CQRS) architectural pattern

---

Accessing OrientDB using Slick

Jackrabbit module to use OrientDB as backend.

---

orientqb

orientqb is a builder for OSQL query language written in Java. orientqb has been thought to help developers in writing complex queries dynamically and aims to be simple but powerful.

Java API

OrientDB is written completely in the Java language. This means that you can use its Java API's without needing to install any additional drivers or adapters.

> For more information, see the OrientDB Java Documentation

Component Architecture

OrientDB provides three different Java API's that allow you to work with OrientDB.

- **Graph API** Use this Java API if you work with graphs and want portable code across TinkerPop Blueprints implementations. It is easiest to switch to this when migrating from other Graph Databases, such as Neo4J or Titan. If you used TinkerPop standard on these, you can use OrientDB as a drop-in replacement.
- **Document API** Use this Java API if your domain fits Document Database use case with schema-less structures. It is easiest to switch to this when migrating from other Document Databases, such as MongoDB and CouchDB.
- **Object API** Use this Java API if you need a full Object Oriented abstraction that binds all database entities to POJO (that is, Plain Old Java Objects). It is easiest to switch to this when migrating from JPA applications.

Each Java API has its own pros and cons. For more information on determining which Java API to use with your application, see Choosing between the Graph or Document API.

	Graph	Document	Object
API	Graph API	Document API	Object Database
Java class	OrientGraph	ODatabaseDocumentTx	OObjectDatabaseTx
Query	Yes	Yes	Yes
Schema Less	Yes	Yes	Yes
Schema full	Yes	Yes	Yes
Speed [1]	90%	100%	50%

> [1]: Speed comparisons show for generic CRUD operations, such as queries, insertions, updates and deletions. Large values are better, where 100% indicates the fastest possible.
>
> In general, the cost of high-level abstraction is a speed penalty, but remember that OrientDB is orders of magnitude faster than the class Relational Database. So, using the Object Database provides a high-level of abstraction with much less code to develop and maintain.

Graph API

With this Java API, you can use OrientDB as a Graph Database, allowing you to work with Vertices and Edges. The Graph API is compliant with the TinkerPop standard.

API: Graph API

Document API

With this Java API, you can handle records and documents. Documents are comprised of fields and fields can be any of the supported types. You can use it with a schema, without, or in a mixed mode.

Additionally, it does not require a Java domain POJO, as is the case with Object Databases.

API: Document API

Object API

With this Java API, you can use OrientDB with JPA-like interfaces where POJO, (that is, Plain Old Java Objects), are automatically bound to the database as documents. You can use this in schema-less or schema-full modes.

> Bear in mind that this Java API has not received improvements since OrientDB version 1.5. Consider using the Document API or Graph API instead, with an additional layer to map to your POJO's.

While you can use both the Graph API and Document API at the same time, the Object API is only compatible with the Document API. It doesn't work well with the Graph API. The main reason is that it requires you to create POJO's that mimic the Vertex and Edge classes, which provides sub-optimal performance in comparison with using the Graph API directly. For this reason, it is recommended that you don't use the Object API with a Graph domain. To evaluate Object Mapping on top of OrientDB Blueprints Graph API, see TinkerPop Frames, Ferma and Totorom.

API: Object Database

Supported Libraries

OrientDB ships with a number of JAR files in the `$ORIENTDB_HOME/lib` directory.

- **`orientdb-core-*.jar`** Provides the core library.
  - *Required*: Always.
  - *Dependencies*: `snappy-*.jar`
  - *Performance Pack (Optional)*:
    - `orientdb-nativeos-*.jar`
    - `jna-*.jar`
    - `jna-platform-*.jar` .
- **`orientdb-client-*.jar`** Provides the remote client.
  - *Required*: When your application connects with a remote server.
- **`orientdb-enterprise-*.jar`** Provides the base package with the protocol and network classes shared by the client and server. Deprecated since version 2.2.
  - *Required*: When your application connects to a remote server.
- **`orientdb-server-*.jar`** Provides the server component.
  - *Required*: When building an embedded server. Used by the OrientDB Server.
- **`orientdb-tools-*.jar`** Provides the console and console commands.
  - *Required*: When you need to execute console commands directly through your application. Used by the OrientDB Console.
- **`orientdb-object-*.jar`** Provides the Object Database interface.
  - *Required*: When you want to use this interface.
  - *Dependencies*: `javassist.jar` and `persistence-api-1.0.jar` .
- **`orientdb-graphdb-*.jar`** Provides the Graph Database interface.
  - *Required*: When you want to use this interface.
  - *Dependencies*: `blueprints-core-*.jar` .
- **`orientdb-distributed-*.jar`** Provides the distributed database plugin.
  - *Required*: When you want to use a server cluster.
  - *Dependencies*: `hazelcast-*.jar` .

Java API Tutorial

In the event that you have only used Relational database systems, you may find much of OrientDB very unfamiliar. Given that OrientDB supports Document, Graph and Object Oriented modes, it requires that you use different Java API's, but there are some similarities between them.

Similar to the JDBC, the TinkerPop produces the Blueprints API, which provides support for basic operations on Graph databases. Using the OrientDB adapter, you can operate on a database without needing to manage OrientDB classes. This makes the resulting code more portable, given that Blueprints offers adapters for other Graph database system.

Tweaking the database configuration itself requires that you use the OrientDB API's directly. It is recommended in these situations that you use a mix, (that is, Blueprints when you can and the OrientDB API's where necessary).

OrientDB Java API's

OrientDB provides three different Java API's that allow you to work with OrientDB. Choose the Java API that supports the mode in which you want to work:

- Graph API
- Document API
- Object API

> For more information on the API's in general, see Java API

Use Case: Graph API

Consider as an example setting up a Graph Database using the Java API with Blueprint.

Connecting to a Graph Database

In order to use the Graph API, you need to create an `OrientGraph` object first:

```
import com.tinkerpop.blueprints.impls.orient.OrientGraph;

OrientGraph graph = new OrientGraph("local:test",
        "username", "password");
```

When your application runs, these lines initialize the `graph` object to your OrientDB database using the Graph API.

Inserting Vertices

While you can work with the generic vertex class `V`, you gain much more power by defining custom types for vertices. For instance,

```
graph.createVertexType("Person");
graph.createVertexType("Address");
```

The Blueprint adapter for OrientDB is thread-safe and where necessary automatically creates transactions. That is, it creates a transaction at the first operation, in the event that you have not yet explicitly started one. You have to specify where these transactions end, for commits or rollbacks.

To add vertices into the database with the Blueprints API:

```
Vertex vPerson = graph.addVertex("class:Person");
vPerson.setProperty("firstName", "John");
vPerson.setProperty("lastName", "Smith");

Vertex vAddress = graph.addVertex("class:Address");
vAddress.setProperty("street", "Van Ness Ave.");
vAddress.setProperty("city", "San Francisco");
vAddress.setProperty("state", "California");
```

When using the Blueprint API, keep in mind that the specific syntax to use is `class:<class-name>`. You must use this syntax when creating an object in order to specify its class. Note that this is not mandatory. It is possible to specify a null value, (which means that the vertex uses the default `V` vertex class, as it is the super-class for all vertices in OrientDB). For instance,

```
Vertex vPerson = graph.addVertex(null);
```

However, when creating vertices in this way, you cannot distinguish them from other vertices in a query.

Inserting Edges

While you can work with the generic edge class `E`, you gain much more power by defining custom types for edges. The method for adding edges is similar to that of vertices:

```
OrientEdge eLives = graph.addEdge(null, vPerson, vAddress, "lives");
```

OrientDB binds the Blueprint label concept to the edge class. That is, you can create an edge of the class `lives` by passing it as a label or as a class name.

```
OrientEdge eLives = graph.addEdge("class:lives", vPerson, vAddress, null);
```

You have now created:

```
[John Smith:Person] --[lives]--> [Van Ness Ave:Address]
```

Bear in mind that, in this example, you have used a partially schema-full mode, given that you defined the vertex types, but not their properties. By default, OrientDB dynamically accepts everything and works in a schema-less mode.

Use Case: SQL Queries

While the Tinkerpop interfaces do allow you to execute fluent queries in SQL or Gremlin, you can also utilize the power of OrientDB SQL through the `.command()` method. For instance,

```
for (Vertex v : (Iterable<Vertex>) graph.command(
        new OCommandSQL(
            "SELECT EXPAND(OUT('bough')) FROM Customer"
          + "WHERE name='Jay'")).execute()) {
    System.out.println("- Bought: " + v);
}
```

In addition to queries, you can also execute any SQL command, such as `CREATE VERTEX`, `UPDATE`, or `DELETE VERTEX`. For instance,

```
int modified = graph.command(
        new OCommandSQL(
            "UPDATE Customer SET local = TRUE"
          + "WHERE 'Rome' IN out('lives').name")).execute());
```

Running this command sets a new property called `local` to `TRUE` on all instances in the class `Customer` where these customers live in Rome.

Graph API

OrientDB follows the TinkerPop Blueprints standard and uses it as the default for the Java Graph API.

When you install OrientDB, it loads a number of Java libraries in the `$ORIENTDB_HOME/lib` directory. In order to use the Graph API, you need to load some of these JAR files into your class-path. The specific files that you need to load varies depending on what you want to do.

To use the Graph API, load the following JAR's:

- `orientdb-core-*.jar`
- `blueprints-core-*.jar`
- `orientdb-graphdb-*.jar`

You may also benefit from these third-party JAR's:

- `jna-*.jar`
- `jna-platform-*.jar`
- `concurrentlinkedhashmap-lru-*.jar`

If you need to connect to a remote server, (as opposed to connecting in Local PLocal or Memory modes), you need to include these JAR's:

- `orientdb-client-*.jar`
- `orientdb-enterprise-*.jar`

To use the TinkerPop Pipes tool, include this JAR:

- `pipes-*.jar`

To use the TinkerPop Gremlin language, include these JAR's:

- `gremlin-java-*.jar`
- `gremlin-groovy-*.jar`
- `groovy-*.jar`

> Bear in mind, beginning with version 2.0, OrientDB disables Lightweight Edges by default when it creates new databases.

Introduction

Tinkerpop provides a complete stack to handle Graph Databases:

- **Blueprints** Provides a collection of interfaces and implementations to common, complex data structures.

  In short, Blueprints provides a one-stop shop for implemented interfaces to help developers create software without getting tied down to the particular, underlying data management systems.

- **Pipes** Provides a graph-based data flow framework for Java 1.6+.

  Process graphs are composed of a set of process vertices connected to one another by a set of communication edges. Pipes supports the splitting, merging, and transformation of data from input to output/

- **Gremlin** Provides a Turing-complete graph-based programming language designed for key/value pairs in multi-relational graphs. Gremlin makes use of an XPath-like syntax to support complex graph traversals. This language has applications in the areas of graph query, analysis, and manipulation.
- **Rexster** Provides a RESTful graph shell that exposes any Blueprints graph as a standalone server.

  Extensions support standard traversal goals such as search, score, rank, and (in concert) recommendation. Rexster makes extensive use of Blueprints, Pipes, and Gremlin. In this way it's possible to run Rexster over various graph systems. to configure Rexster to work with OrientDB, see the guide Rexster Configuration.

- **Sail Ouplementation** Provides support for using OrientDB as an RDF Triple Store.

Getting Started with Blueprints

OrientDB supports three different kinds of storages, depending on the Database URL used:

- **Persistent Embedded Graph Database**: Links to the application as a JAR, (that is, with no network transfer). Use PLocal with the `plocal` prefix. For instance, `plocal:/tmp/graph/test`.
- **In-Memory Embedded Graph Database**: Keeps all data in memory. Use the `memory` prefix, for instance `memory:test`.
- **Persistent Remote Graph Database** Uses a binary protocol to send and receive data from a remote OrientDB server. Use the `remote` prefix, for instance `remote:localhost/test` Note that this requires an OrientDB server instance up and running at the specific address, (in this case, localhost). Remote databases can be persistent or in-memory as well.

In order to use the Graph API, you need to create an instance of the `OrientGraph` class. The constructor receives a Database URL that is the location of the database. If the database already exists, the Graph API opens it. If it doesn't exist, the Graph API creates it.

> **NOTE**: When creating a database through the Graph API, you can only create PLocal and Memory databases. Remote databases must already exist.
>
> **NOTE**: In v. 2.2 and following releases, when using PLocal or Memory,please set MaxDirectMemorySize (JVM setting) to a high value, like 512g `-XX:MaxDirectMemorySize=512g`

When building multi-threaded application, use one instance of `OrientGraph` per thread. Bear in mind that all graph components, such as vertices and edges, are not thread safe. So, sharing them between threads may result in unpredictable results.

Remember to always close the graph instance when you are done with it, using the `.shutdown()` method. For instance:

```
OrientGraph graph = new OrientGraph("plocal:C:/temp/graph/db");
try {
  ...
} finally {
  graph.shutdown();
}
```

Using the Factory

Beginning with version 1.7, OrientDB introduces the `OrientGraphFactory` class as new method for creating graph database instances through the API.

```
// AT THE BEGINNING
OrientGraphFactory factory = new OrientGraphFactory("plocal:/tmp/graph/db").setupPool(1,10);

// EVERY TIME YOU NEED A GRAPH INSTANCE
OrientGraph graph = factory.getTx();
try {
  ...

} finally {
   graph.shutdown();
}
```

> For more information, see Graph Factory.

Transactions

Prior to version 2.1.7, whenever you modify the graph database instance, OrientDB automatically starts an implicit transaction, in the event that no previous transactions are running. When you close the graph instance, it commits the transaction automatically by calling the `.shutdown()` method or through `.commit()`. This allows you to roll changes back where necessary, using the `.rollback()` method.

Beginning in version 2.1.8, you can set the Consistency Level. Changes within the transaction remain temporary until the commit or the closing of the graph database instance. Concurrent threads or external clients can only see the changes after you fully commit the transaction.

For instance,

```
try{
  Vertex luca = graph.addVertex(null); // 1st OPERATION: IMPLICITLY BEGINS TRANSACTION
  luca.setProperty( "name", "Luca" );
  Vertex marko = graph.addVertex(null);
  marko.setProperty( "name", "Marko" );
  Edge lucaKnowsMarko = graph.addEdge(null, luca, marko, "knows");
  graph.commit();
} catch( Exception e ) {
  graph.rollback();
}
```

By surrounding the transaction in `try` and `catch`, you ensure that any errors that occur roll the transaction back to its previous state for all relevant elements. For more information, see Concurrency.

> **NOTE**: Prior to version 2.1.7, to work with a graph always use transactional `OrientGraph` instances and never the non-transactional instances to avoid graph corruption from multi-threaded updates.
>
> Non-transactional graph instances are created with
>
> ```
> OrientGraphNoTx graph = factor.getNoTx();
> ```
>
> This instance is only useful when you don't work with data, but want to define the database schema or for bulk inserts.

Optimistic Transactions

OrientDB supports optimistic transactions. This means that no lock is kept when a transaction runs, but at commit time each graph element version is checked to see if there has been an update by another client.

For this reason, write your code to handle the concurrent updating case:

```
for (int retry = 0; retry < maxRetries; ++retry) {
    try {
        // LOOKUP FOR THE INVOICE VERTEX
        Iterable<Vertex> invoices = graph.getVertices("invoiceId", 2323);
        Vertex invoice = invoices.iterator().next();

        // CREATE A NEW ITEM
        Vertex invoiceItem = graph.addVertex("class:InvoiceItem");
        invoiceItem.field("price", 1000);
        // ADD IT TO THE INVOICE
        invoice.addEdge(invoiceItem);
        graph.commit();

        // OK, EXIT FROM RETRY LOOP
        break;
    } catch( ONeedRetryException e ) {
        // SOMEONE HAVE UPDATE THE INVOICE VERTEX AT THE SAME TIME, RETRY IT
    }
}
```

Using Non-Transactional Graphs

In cases such as massive insertions, you may find the standard transactional graph `OrientGraph` is too slow for your needs. You can speed the operations up by using the non-transactional `OrientGraphNoTx` graph.

With the non-transactional graph, each operation is *atomic* and it updates data on each operation. When the method returns, the underlying storage updates. This works best for bulk inserts and massive operations, or for schema definitions.

> **NOTE**: Using non-transactional graphs may cause corruption in the graph in cases where changes are made through multiple threads at the same time. It is recommended that you only use non-transactional graph instances for single-threaded operations.

Graph Configuration

Beginning in version 1.6 of OrientDB, you can configure the graph by setting all of its properties during construction:

- `blueprints.orientdb.url` Defines the database URL.
- `blueprints.orientdb.username` Defines the username

  Default: `admin`

- `blueprints.orientdb.password` Defines the user password.

  Default: `admin`

- `blueprints.orientdb.saveOriginalIds` Defines whether it saves the original element ID's by using the property `id`. You may find this useful when importing a graph to preserve original ID's.

  Default: `false`

- `blueprints.orientdb.keepInMemoryReferences` Defines whether it avoids keeping records in memory, by using only Record ID's.

  Default: `false`

- `blueprints.orientdb.useCustomClassesForEdges` Defines whether it uses the edge label as the OrientDB class. If the class doesn't exist, it creates it for you.

  Default: `true`

- `blueprints.orientdb.useCustomClassesForVertex` Defines whether it uses the vertex label as the OrientDB class. If the class doesn't exist, it creates it for you.

  Default: `true`

- `blueprints.orientdb.useVertexFieldsForEdgeLabels` Defines whether it stores the edge relationship in the vertex by using the edge class. This allows you to use multiple fields and makes traversals by the edge label (class) faster.

  Default: `true`

- `blueprints.orientdb.lightweightEdges` Defines whether it uses Lightweight Edges. This allows you to avoid creating a physical document per edge. Documents are only created when edges have properties.

  Default: `false`

- `blueprints.orientdb.autoStartTx` Defines whether it auto-starts transactions when the graph is changed by adding or removing vertices, edges and properties.

  Default: `true`

Using Gremlin

You can also use the Gremlin language with OrientDB. To do so, initialize it, using `OGremlinHelper`:

```
OGremlinHelper.global().create()
```

> For more information on Gremlin usage, see:
>
> - How to use the Gremlin language with OrientDB
> - Getting started with Gremlin
> - Usage of Gremlin through HTTP/RESTful API using the Rexter project.

Multi-Threaded Applications

In order to use the OrientDB Graph API with multi-threaded applications, you must use one `OrientGraph` instance per thread.

For more information about multi-threading look at Java Multi Threading. Bear in mind, graph components (such as, Vertices and Edges) are not thread-safe. Sharing them between threads could cause unpredictable errors.

Vertices and Edges

Similar to the Console interface, you can also create, manage and control vertices and edges through the Graph API.

Vertices

To create a new vertex in the current Graph Database instance, call the `Vertex OrientGraph.addVertex(Object id)` method. Note that this ignores the `id` parameter, given that the OrientDB implementation assigns a unique ID once it creates the vertex. To return the unique ID, run the `Vertex.getId()` method on the object.

For instance,

```
Vertex v = graph.addVertex(null);
System.out.println("Created vertex: " + v.getId());
```

Retrieving Vertices

To retrieve all vertices on the object, use the `getVertices()` method. For instance,

```
for (Vertex v : graph.getVertices()) {
    System.out.println(v.getProperty("name"));
}
```

To lookup vertices by a key, call the `getVertices()` method by passing the field name and the value to match. Remember that in order to use indexes, you should use the "class" dot (.) "property name" as field name. Example:

```
for( Vertex v : graph.getVertices("Account.id", "23876JS2") ) {
  System.out.println("Found vertex: " + v );
}
```

To know more about how to define indexes look at: Using Graph Indexes.

Removing Vertices

To remove a vertex from the current Graph Database, call the `OrientGraph.removeVertex(Vertex vertex)` method. This disconnects the vertex from the graph database and then removes it. Disconnection deletes all vertex edges as well.

For instance,

```
graph.removeVertex(luca);
```

Disconnects and removes the vertex `luca`.

Edges

Edges link two vertices in the database. The vertices must exist already. To create a new edge in the current Graph Database, call the `Edge OrientGraph.addEdge(Object id, Vertex outVertex, Vertex inVertex, String label )` method.

Bear in mind that OrientDB ignores the `id` parameter, given that it assigns a unique ID when it creates the edge. To access this ID, use the `Edge.getId()` method. `outVertex` refers to the vertex instance where the edge starts and `inVertex` refers to where the edge ends. `label` indicates the edge label. Specify it as `null` if you don't want to assign a label.

For instance,

```
Vertex luca = graph.addVertex(null);
luca.setProperty("name", "Luca");

Vertex marko = graph.addVertex(null);
marko.setProperty("name", "Marko");

Edge lucaKnowsMarko = graph.addEdge(null, luca, marko, "knows");
System.out.println("Created edge: " + lucaKnowsMarko.getId());
```

For more information on optimizing edge creation through concurrent threads and clients, see Concurrency on Adding Edges.

Retrieving Edges

To retrieve all edges use the getEdges() method:

```
for (Edge e : graph.getEdges()) {
    System.out.println(e.getProperty("age"));
}
```

When using Lightweight Edges, OrientDB stores edges as links rather than records. This improves performance, but as a consequence, the `.getEdges()` method only retrieves records of the class `E`. When using Lightweight Edges, OrientDB only creates records in class `E` under certain circumstances, such as when the edge has properties. Otherwise, the edges exist as links on the in and out vertices.

If you want to use `.getEdges()` to return all edges, disable the Lightweight Edges feature by executing the following command:

```
orientdb> ALTER DATABASE my_db useLightweightEdges=FALSE
```

You only need to run this command once to disable Lightweight Edges. The change only takes effect on edges you create after running it. For existing edges, you need to convert them from links to actual edges before the `.getEdges()` method returns all edges. For more information, see Troubleshooting.

> **NOTE**: Since version 2.0 of OrientDB, the Lightweight Edges feature is disabled by default.

Removing Edges

To remove an edge from the current Graph Database, call the `OrientGraph.removeEdge(Edge edge)` method. It removes the edge connecting two vertices.

For instance,

```
graph.removeEdge(lucaKnowsMarko);
```

Vertex and Edge Properties

Vertices and Edges can have multiple properties. The key to this property is a String, the value any Types supported by OrientDB.

Method	Description
`setProperty(String key, Object value)`	Sets the property.
`Object getProperty(String key)`	Retrieves the property.
`void removeProperty(String key)`	Removes the property.

For instance,

```
vertex2.setProperty("x", 30.0f);
vertex2.setProperty("y", ((float) vertex1.getProperty( "y" )) / 2);

for (String property : vertex2.getPropertyKeys()) {
      System.out.println("Property: " + property + "=" + vertex2.getProperty(property));
}

vertex1.removeProperty("y");
```

Setting Multiple Properties

The OrientDB implementation of the Blueprints extension supports setting multiple properties in one command against vertices and edges, using the `setProperties(Object ...)` method. This improves performance by allowing you to avoid saving the graph element each time you set a property.

For instance,

```
vertex.setProperties( "name", "Jill", "age", 33, "city", "Rome", "born", "Victoria, TX" );
```

You can also pass a Map of values as the first argument. In this case all map entries are set as element properties:

```
Map<String,Object> props = new HashMap<String,Object>();
props.put("name", "Jill");
props.put("age", 33);
props.put("city", "Rome");
props.put("born", "Victoria, TX");
vertex.setProperties(props);
```

Creating Elements and Properties Together

In addition to the above methods, using the OrientDB Blueprint implementation, you can also set the initial properties while creating vertices or edges.

For instance,

- To create a vertex and set the initial properties:

  ```
  graph.addVertex("class:Customer", "name", "Jill",
      "age", 33, "city", "Rome", "born", "Victoria, TX");
  ```

  This line creates a new vertex of the class `Customer`. It sets the initial properties for `name`, `age`, `city`, and `born`.

- The same procedure also works for edges.

  ```
  person1.addEdge("class:Friend", person1, person2, null,
      "since", "2013-07-30");
  ```

  This creates an edge of the class `Friend` between the vertices `person1` and `person2` with the property `since`.

Both methods accept `Map<String, Object>` as a parameter, allowing you to set one property per map entry, such as in the above example.

Using Indices

OrientDB allows query execution against any field of a vertex or edge, indexed or non-indexed. To speed up queries, set up indices on key properties that use in the query.

- For instance, say that you have a query that looks for all vertices with the name `OrientDB`. For instance,

  ```
  graph.getVertices("name", "OrientDB");
  ```

- Without an index against the property `name`, this query can take up a lot of time. You can improve performance by creating a new index against the `name` property:

  ```
  graph.createKeyIndex("name", Vertex.class);
  ```

- In cases where the name must be unique, you can enforce this constraint by setting the index as `UNIQUE`. (This feature is only available in OrientDB).

  ```
  graph.createKeyIndex("name", Vertex.class, new Parameter("type", "UNIQUE"));
  ```

  It applies this constraint to vertices and sub-type instances.

- To create an index against a custom type, such as the `Customer` vertex, use the additional `class` parameter:

  ```
  graph.createKeyIndex("name", Vertex.class, new Parameter("class", "Customer"));
  ```

- You can also have both unique indices against custom types:

  ```
  graph.createKeyIndex("name", Vertex.class, new Parameter("type", "UNIQUE"), new Parameter("class", "Customer"));
  ```

- To create a case-insensitive index, use the `collate` parameter:

  ```
  graph.createKeyIndex("name", Vertex.class, new Parameter("type", "UNIQUE"), new Parameter("class", "Customer"),new Parame
  ter("collate", "ci"));
  ```

- To get a vertex or edge by key prefix, use the class name before the property. For example above, you would use `Customer.name` instead of just `name` to create the index against the field of that class:

  ```
  for (Vertex v : graph.getVertices("Customer.name", "Jay")) {
      System.out.println("Found vertex: " + v);
  }
  ```

- In the event that the class name is not passed, it uses `V` for vertices and `E` for edges:

  ```
  graph.getVertices("name", "Jay");
  graph.getEdges("age", 20);
  ```

> For more information, see Indexes.

Using the Blueprints Extensions

OrientDB is a graph database that merges graph, document and object-oriented worlds together. Below are some of the features exclusive to OrientDB through the Blueprints Extensions.

> For information on tuning your graph database, see Performance Tuning Blueprints.

Custom Types

OrientDB supports custom types for vertices and edges in an Object Oriented manner. This feature is not supported directly through Blueprints, but there is a way to implement them. If you want to create a schema to work with custom types, see Graph Schema.

Additionally, OrientDB introduces a few variants to Blueprint methods for working with types.

Creating Vertices and Edges in Specific Clusters

By default, each class has one cluster with the same name. You can add multiple clusters to the class, allowing OrientDB to write vertices and edges on multiple files. Furthermore, when working in Distributed Mode, you can configure different servers to manage each cluster.

For instance,

```
// SAVE THE VERTEX INTO THE CLUSTER 'PERSON_USA'
// ASSIGNED TO THE NODE 'USA'
graph.addVertex("class:Person,cluster:Person_usa");
```

Retrieving Vertices and Edges by Type

To retrieve all vertices of the `Person` class, use the special `getVerticesOfClass(String className)` method.

For instance,

```
for (Vertex v : graph.getVerticesOfClass("Person")) {
    System.out.println(v.getProperty("name"));
}
```

Here, it retrieves all vertices of the class `Person` as well as all sub-classes. It does this because OrientDB uses polymorphism by default. If you would like to retrieve only those vertices of the `Person` class, excluding sub-types, use the `getVerticesOfClass(String className, boolean polymorphic)` method, specifying `false` in the `polymorphic` argument. For instance,

```
for (Vertex v : graph.getVerticesOfClass("Person", false)) {
    System.out.println(v.getProperty("name"));
}
```

You can also use variations on these with the `.getEdges()` method:

- `getEdgesOfClass(String className)`
- `getEdgesOfClass(String, className, boolean polymorphic)`

Ordered Edges

By default, OrientDB uses a set to handle edge collection. But, sometimes it's better to have an ordered list to access the edge by an offset.

- For instance,

  ```
  person.createEdgeProperty(Direction.OUT, "Photos").setOrdered(true);
  ```

- Whenever you access the edge collection, it orders the edges. Consider the example below, which prints all photos in the database in an ordered way:

  ```
  for (Edge e : loadedPerson.getEdges(Direction.OUT, "Photos")) {
    System.out.println( "Photo name: " + e.getVertex(Direction.IN).getProperty("name") );
  }
  ```

- To access the underlying edge list, you need to use the Document Database API. Consider this example, which swaps the tenth photo with the last:

  ```
  // REPLACE EDGE Photos
  List<ODocument> photos = loadedPerson.getRecord().field("out_Photos");
  photos.add(photos.remove(9));
  ```

- You can get the same result through SQL by executing the following commands in the terminal:

  ```
  orientdb> CREATE PROPERTY out_Photos LINKLIST
  orientdb> ALTER PROPERTY User.out_Photos CUSTOM ORDERED=TRUE
  ```

Detached Elements

When working with web applications, it is very common to query elements and render them to the user, allowing them to apply changes.

Consider what happens when the user changes some fields and saves. Previously, the developer had to track the changes in a separate structure, load the vertex or edge from the database, and then apply the changes to the element. Beginning in version 1.7 OrientDB, there are two new methods in the Graph API on the `OrientElement` and `OrientBaseGraph` classes:

- `OrientElement.detach()` and `OrientBaseGraph.detach(OrientElement)` methods fetch all record content in memory and resets the connection to the Graph instance.
- `OrientElement.attach()` and `OrientBaseGraph.attach(OrientElement)` methods save the detached element back into the database and restores the connection between the Graph Element and the Graph instance.

For instance, to use these begin by loading a vertex and detaching it:

```
OrientGraph g = OrientGraph("plocal:/temp/db");
try {
    Iterable<OrientVertex> results = g.query().has("name", EQUALS, "fast");
    for (OrientVertex v : results)
        v.detach();
} finally {
    g.shutdown();
}
```

Update the element, either from the GUI or by application:

```
v.setProperty("name", "super fast!");
```

When the user saves, re-attach the element and save it to the database:

```
OrientGraph g = OrientGraph("plocal:/temp/db");
try {
    v.attach(g);
    v.save();
} finally {
    g.shutdown();
}
```

FAQ

Do `detach()` methods work recursively to detach all connected elements? No, it only works on the current element.

Can you add an edge against detached elements? No, you can only get, set or remove a property while it's detached. Any other operation that requires the database it throws an `IllegalStateException` exception.

Executing Commands

The OrientDB Blueprints implementation allows you to execute commands using SQL, JavaScript and all other supported languages.

SQL Queries

It is possible to have parameters in a query using prepared queries.

```
for (Vertex v : (Iterable<Vertex>) graph.command(new OCommandSQL(
            "SELECT EXPAND( out('bought') ) FROM Customer WHERE name = 'Jay'")).execute()) {
    System.out.println("- Bought: " + v);
}
```

To execute an asynchronous query:

```
graph.command(
          new OSQLAsynchQuery<Vertex>("SELECT FROM Member",
            new OCommandResultListener() {
              int resultCount =0;
              @Override
              public boolean result(Object iRecord) {
                resultCount++;
                Vertex doc = graph.getVertex( iRecord );
               return resultCount < 100;
              }
            } ).execute();
```

SQL Commands

In addition to queries, you can also execute any SQL command, such as `CREATE VERTEX`, `UPDATE`, or `DELETE VERTEX`. For instance, consider a case where you want to set a new property called `local` to `true` on all the customers that live in Rome.

```
int modified = graph.command(
          new OCommandSQL("UPDATE Customer SET local = true WHERE 'Rome' IN out('lives').name")).execute());
```

If the command modifies the schema, (such as in cases like `CREATE CLASS`, `ALTER CLASS`, `DROP CLASS`, `CREATE PROPERTY`, `ALTER PROPERTY`, and `DROP PROPERTY`, remember you need to force the schema update of the database instance you're using by calling the `.reload()` method.

```
graph.getRawGraph().getMetadata().getSchema().reload();
```

> For more information, see SQL Commands

Prepared statements

Hardwiring values in a command/query is not a good practice, it can lead to SQL injection and it's inefficient in terms of query parsing.

OrientDB allows you to define named and positional parameters for queries and to pass the parameter values to the `execute()` method

Positional parameters

Positional parameters are represented in the query string as question marks `?`. The `execute()` method accepts positional parameters as single values, or as an `Object[]`

eg.

```
graph.command(
          new OCommandSQL("UPDATE Customer SET local = true WHERE name = ? and surname = ?")
      ).execute("John", "Smith")
    );
```

The parameters are assigned in the same order as they appear in the query.

Named parameters

Positional parameters are represented in the query string as a name prefixed by a colon: `:<paramName>`. The `execute()` method accepts positional parameters as a `Map<String, Object>`, where the key is the parameter name (without the `:`) and the value is the parameter value.

eg.

```
Map<String, Object> params = new HashMap<String, Object>();
params.put("theName", "John");
params.put("theSurname", "Smith");

graph.command(
          new OCommandSQL("UPDATE Customer SET local = true WHERE name = :theName and surname = :theSurname")
      ).execute(params)
    );
```

You can also use the same parameter multiple times in the same query, eg. the following is valid:

```
Map<String, Object> params = new HashMap<String, Object>();
params.put("theName", "John");

graph.command(
          new OCommandSQL("UPDATE Customer SET local = true WHERE name = :theName and surname = :theName")
      ).execute(params)
    );
```

SQL Batch

To execute multiple SQL commands in a batch, use the `OCommandScript` and SQL as the language. This is recommended when creating edges on the server-side, to minimize the network roundtrip.

```
String cmd = "BEGIN\n";
cmd += "LET a = CREATE VERTEX SET script = true\n";
cmd += "LET b = SELECT FROM V LIMIT 1\n";
cmd += "LET e = CREATE EDGE FROM $a TO $b RETRY 100\n";
cmd += "COMMIT\n";
cmd += "return $e";

OIdentifiable edge = graph.command(new OCommandScript("sql", cmd)).execute();
```

> For more information, see SQL Batch

Database Functions

To execute database functions, you must write it in JavaScript or any other supported languages. For the examples, consider a function called `updateAllTheCustomersInCity(cityName)` that executes the same update as above. Note the `'Rome'` attribute passed in the attribute apssed int he `execute()` method:

```
graph.command(new OCommandFunction(
    "updateAllTheCustomersInCity")).execute("Rome"));
```

Executing Code

To execute code on the server side, you can select between the supported language, (which by default is JavaScript):

```
graph.command(
          new OCommandScript("javascript", "for(var i=0;i<10;++i){ print('\nHello World!'); }")).execute());
```

This prints the line "Hello World!" ten times in the server conssole, or in the local console, if the database has been opened in PLocal mode.

Accessing the Graph

The TinkerPop Blueprints API is quite raw and doesn't provide ad hoc methods for very common use cases. To get around this, you may need to access the `ODatabaseGraphTx` object to extend what you can do through the underlying graph engine. Common operations are:

- Counting incoming and outgoing edges without browsing them.
- Getting incoming and outgoing vertices without browsing the edges.
- Executing a query using an SQL-like language integrated in the engine.

The `OrientGraph` class provides the method `.getRawGraph()` to return the underlying Document database. For instance,

```
final OrientGraph graph = new OrientGraph("plocal:C:/temp/graph/db");
try {
  List<ODocument> result = graph.getRawGraph().query(
            new OSQLSynchQuery("SELECT FROM V WHERE color = 'red'"));
} finally {
  graph.shutdown();
}
```

Security

If you want to use OrientDB security, use the construction that retrieves the Database URL, user and password. For more information on OrientDB security, see Security. By default, it uses the `admin` user.

Graph Factory

The TinkerPop Blueprints standard does not implement a proper "factory" method for creating and pooling Graph Database instances. In applications based on OrientDB, you may occasionally encounter legacy workarounds that attempt to implement this feature by mixing the Graph and Document API's.

With the introduction of the `OrientGraphFactory` class, managing graphs is much simpler.

- By default, the class acts as a factory, creating new Graph Database instances every time.
- You can configure it to work as a pool, recycling Graph Database instances as need.
- When running in PLocal mode, if the database itself doesn't exist, the class creates it for you.
- It returns both transactional and non-transactional Graph Database instances.
- When you call the `shutdown()` method, it returns the pooled instance so that you can reuse the pool.

Using Graph Factories

Using a Graph Factory in your application is relatively straightforward. First, create and configure a factory instance using the `OrientGraphFactory` class. Then, use the factory whenever you need to create an `OrientGraph` instance and shut it down to return it to the pool. When you're done with the factory, close it to release all instances and free up system resources.

Creating Factories

The basic way to create the factory with the default user `admin`, (which passes the password `admin`, by default):

```
OrientGraphFactory factory = new OrientGraphFactory("plocal:/temp/mydb");
```

In the event that you have implemented basic security housekeeping on your database, like changing the default `admin` password or if you want to use a different user, you can pass the username and password as arguments to `OrientGraphFactory`:

```
OrientGraphFactory factory = new OrientGraphFactory("plocal:/temp/mydb", "jayminer", "amigarocks");
```

Recycling Pools

In addition to creating new Graph Database instances, the factory can also create recyclable pools, by using the `setPool()` method:

```
OrientGraphFactory factory = new OrientGraphFactory("plocal:/temp/mydb").setupPool(1, 10);
```

This sets the pool up to use a minimum of 1 and a maximum of 10 Graph Database instances.

Using Graph Instances

Once you create and configure the factory, you can get the Graph Database instance to start using it with your application. OrientDB provides three methods for getting this instance. Which you use depends on preference and whether you want a transactional instance.

- To retrieve a transactional instance, use the `getTx()` method on your factory object:

  ```
  OrientGraph txGraph = factory.getTx();
  ```

- To retrieve a non-transactional instance, use the `getNoTx()` method on your factory object:

  ```
  OrientGraphNoTx noTxGraph = factory.getNoTx();
  ```

- Alternatively, if you plan to get several Graph Database instances of the same type, you can use the `setTransacational()` method to define the kind you want, then use the `get()` method for each instance you retrieve:

```
factory.setTransactional(false);
OrientGraphNoTx noTxGraph = (OrientGraphNoTx) factory.get();
```

Shutting Down Graph Instances

When you're done with a Graph Database instance, you can return it to the pool by calling the `shutdown()` method on the instance. This method does not close the instance. The instance remains open and available for the next requester:

```
graph.shutdown();
```

Releasing the Graph Factory

When you're ready to release all instances, call the `close()` method on the factory. In the case of pool usage, this also frees up the system resources claimed by the pool:

```
factory.close();
```

Legacy Workarounds

In older OrientDB applications, you may occasionally encounter legacy workarounds that antedate the `OrientGraphFactory` class.

For instance,

```
ODatabaseDocumentPool pool = new ODatabaseDocumentPool("plocal:/temp/mydb");
OrientGraph g = new OrientGraph(pool.acquire());
```

The code works around the absence of a factory class in TinkerPop Blueprints by mixing the Graph and Document API's together at an application level.

With the factory class, you no longer need to rely upon such hackneyed methods.

Graph Schema

In OrientDB you can use a Graph Database in schema-less mode, or you can enforce strict data model through a schema. When using a schema, you can use it on all data or only define constraints on certain fields, allowing users to add custom fields to records.

The schema mode you use is defined at a class-level. So, for instance you might have a class `Employee` that is schema-full with a class `EmployeeInformation` that is schema-less.

- **Schema-Full**: Enables the strict-mode at a class level and sets all fields to mandatory.
- **Schema-Less**: Creates classes with no properties. By default, this is non-strict-mode, allowing records to have arbitrary fields.
- **Schema-Hybrid**: Creates classes and defines some of the fields, while leaving the records to define custom fields. Sometimes, this is also called **Schema-Hybrid**.

> **NOTE**: Bear in mind that any changes you make to the class schema are not transactional. You must execute them outside of a transaction.

To access the schema, you can use either SQL or the API. The examples in the following pages use the Java API.

- **Graph Database Classes**
- **Graph Database Properties**

> For more information, see Blog: Using Schema with Graphs

Classes in the Graph Database

Classes are a concept drawn from the Object-Oriented Programming paradigm. OrientDB defines it as a record-type. In the Relational Model, it is nearest the concept of the table. Classes can be schema-less, schema-full or mixed. Classes can inherit from another class, shaping a tree of classes. Due to inheritance, the subclass extends the parent class, inheriting all the attributes as though they were its own.

Each class has one cluster defined as its default cluster, but it can support multiple clusters. In this case by default, OrientDB writes new records in the default cluster, but reads always occur on defined clusters. When you create a new class, it creates a default physical cluster with the same name as the class, changed to lowercase.

In Graph Databases, the structure follows tow classes: `V` as the base class for vertices and `E` as the base class for edges. OrientDB builds these classes automatically when you create the graph database. In the event that you don't have these classes, create them, (see below).

Working with Vertex and Edge Classes

While you can build graphs using `V` and `E` class instances, it is strongly recommended that you create custom types for vertices and edges.

To create a custom vertex class (or type) use the `createVertexType(<name>)` :

```
// Create Graph Database Instance
OrientGraph graph = new OrientGraph("plocal:/tmp/db");

// Create Custom Vertex Class
OrientVertexType account = graph.createVertexType("Account");
```

To create a vertex of that class `Account` , pass a string with the format `"class:<name>"` :

```
// Add Vertex Instance
Vertex v = graph.addVertex("class:Account");
```

In Blueprints, edges have the concept of labels used in distinguishing between edge types. OrientDB binds the concept of edge labels to edge classes. There is a similar method in creating custom edge types, using `createEdgeType(<name)` :

```
// Create Graph Database Instance
OrientGraph graph = new OrientGraph("plocal:/tmp/db");

// Create Custom Vertex Classes
OrientVertexType accountVertex = graph.createVertexType("Account");
OrientVertexType addressVertex = graph.createVertexType("Address");

// Create Custom Edge Class
OrientEdgeType livesedge = graph.createEdgeType("Lives");

// Create Vertices
Vertex account = graph.addVertex("class:Account");
Vertex address = graph.addVertex("class:Address");

// Create Edge
Edge e = account.addEdge("Lives", address);
```

Inheritance Tree

Classes can extend other classes. Beginning with version 2.1, OrientDB supports multiple inheritances. To create a class that extend a class different from `V` or `E` , pass the class name in the construction:

```
graph.createVertexType(<class>, <super-class>); // Vertex
graph.createEdgeType(<class>, <super-class>); // Edge
```

For instance, create the base class `Account`, then create two subclasses: `Provider` and `Customer`:

```
// Create Vertex Base Class
graph.createVertexType("Account");

// Create Vertex Subclasses
graph.createVertexType("Customer", "Account");
graph.createVertexType("Provider", "Account");
```

Retrieve Types

Classes are polymorphic. If you search for generic vertices, you also receive all custom vertex instances:

```
// Retrieve Vertices
Iterable<Vertex> allVertices = graph.getVertices();
```

To retrieve custom classes, use the `getVertexType()` and the `getEdgeType` methods. For instance, retrieving from the `graph` database instance:

```
OrientVertexType accountVertex = graph.getVertexType("Account");
OrientEdgeType livesEdge = graph.getEdgeType("Lives");
```

Drop Persistent Types

To drop a persistent class, use the `dropVertexType()` and `dropVertexType()` methods. For instance, dropping from the `graph` database instance:

```
graph.dropVertexType("Address");
graph.dropEdgeType("Lives");
```

Graph Database Properties

Fields in classes are called properties. In this guide, you can consider properties synonymous with fields.

Working with Properties

In order to use properties in your application, you first need to create the class and set an instance. Once you have the class set, you can begin to work with properties on that class.

Creating Properties

You can define properties for that class. For instance,

```
// Retrieve Vertex
OrientVertexType accountVertex = graph.getVertexType("Account");

// Create Properties
accountVertex.createProperty("id", OType.INTEGER);
accountVertex.createProperty("birthDate", OType.DATE);
```

Bear in mind, each field must belong to a Type.

Dropping Properties

To drop a persisten class property, use the `OClass.dropProperty()` method. For instance,

```
accountVertex.dropProperty("name");
```

This drops the property `name`. OrientDB does not remove dropped properties from the record unless you delete them explicitly using the SQL `UPDATE` command with the `REMOVE` clause.

```
// Drop the Property
accountVertex.dropProperty("name");

// Remove the Records
database.command(new OCommandSQL("UPDATE Account REMOVE name").execute();
```

Using Constraints

OrientDB supports a number of constraints for each field:

Constraint	Method	Description
Minimum Value	`setMin()`	Defines the minimum value. Property accepts strings and also works on date ranges.
Maximum Value	`setMax()`	Defines the maximum value. Property accepts strings and also works on date ranges.
Mandatory	`setMandatory()`	Defines whether the property must be specified.
Read Only	`setReadonly()`	Defines whether you can update the property after creating the record.
Not Null	`setNotNull()`	Defines whether the property accepts null values.
Unique		Defines whether the property must be unique.
Regex		Defines whether the property must satisfy a Regular Expressions value.
Ordered	`setOrdered()`	Defines whether the edge list must be ordered, ensuring that a `List` is not used in place of a `Set` .

For example,

```
// Create Unique Nickname
profile.createProperty("nick", OType.STRING).setMin("3").setMax("30")
    .setMandatory(true).setNotNull(true);
profile.createIndex("nickIdx", OClass.INDEX_TYPE.UNIQUE, "nick");

// Create User Properties
profile.createProperty("name", OType.STRING).setMin("3").setMax("30");
profile.createProperty("surname", OType.STRING).setMin("3").setMax("30");
profile.createProperty("registeredOn", OType.DATE).setMin("2010-01-01 00:00:00");
profile.createProperty("lastAccessOn", OType.DATE).setMin("2010-01-01 00:00:00");
```

Indexes as Constraints

In order to set the property value to unique, use the `UNIQUE` index as a constraint by passing a `Parameter` object with the key `type` . For instance,

```
graph.createKeyIndex("id", Vertex.class, new Parameter("type", "UNIQUE"));
```

OrientDB applies this constraint to all vertex and subclass instances. To specify an index against a custom type, use the `class` parameter.

```
graph.createKeyIndex("name", Vertex.class, new Parameter("class", "Member"));
```

You can also define a unique index against a custom type:

```
graph.createKeyIndex("id", Vertex.class, new Parameter("type", "UNIQUE"),
    new Parameter("class", "Member"));
```

You can then retrieve a vertex or an edge by key prefixing the class name to the field. For instance, using the about `Member.name` in place of only `name` , which allows you to use the index created against the filed `name` in the class `Member` .

```
for( Vertex v : graph.getVertices("Member.name", "Jay")) {
    System.out.println("Found vertex: " + v)
}
```

If the class name is not passed, then it uses `V` for the vertices and `E` for edges.

```
graph.getVertices("name", "Jay");
graph.getEdges("age", 20);
```

> For more information, see
>
> - Indexes
> - Graph Schema
> - Graph Database

Partitioned Graphs

You can partition graphs using the Record-level Security feature. This allows you to separate database records as sandboxes, where "restricted" records are inaccessible to unauthorized users. For more information about other solution for Multi-Tenant applications, look at Multi-Tenant.

This tutorial provides a demonstration of sandboxing with the Graph API and the TinkerPop stack. Partitioned Graph Databases allow you to build Multi-tenant applications.

> **Requirements**:
>
> - OrientDB 1.2.0-SNAPSHOT or higher
> - TinkerPop Blueprints 2.2.0 or higher

Creating a Graph Database

To create a Partitioned Graph Database, you first need to create a database in which to partition. For instance, the example below covers creating a `blog` database of the Graph type on the local file system:

```
$ cd $ORIENTDB_HOME/bin
$ ./console.sh
OrientDB console v.1.2.0-SNAPSHOT www.orientechnologies.com
Type 'help' to display all the commands supported.

Installing extensions for GREMLIN language v.2.2.0-SNAPSHOT

orientdb> CREATE DATABASE local:../databases/blog
          admin admin_passwd local graph
Creating database [local:../databases/blog] using the storage type [local]...
Database created successfully.

Current database is: local:../databases/blog
```

Enabling Partitioned Graphs

Once you have created the Graph Database instance, you need to enable partitioning. Alter the `V` and `E` classes to extend the `ORestricted`. Doing so allows you to restrict vertex and edge instances.

```
orientdb {db=blog}> ALTER CLASS V SUPERCLASS ORestricted
Class updated successfully

orientdb {db=blog}> ALTER CLASS E SUPERCLASS ORestricted
Class updated successfully
```

Create Users

With the database online and the vertex and edge classes altered to restricted, create two test users, (`luca` and `steve`), to use in exploring how sandboxing works on a Partioned Graph Database.

These users represent users who blog on the application you're building. Their level of permissions and authorization relates to the `writer` role. First, ensure that the `writer` role exists on your database:

```
orientdb {db=blog}> SELECT RID, name, rules FROM ORole
                WHERE name = "writer"

---+------+--------+--------------------------------------------
 # | RID  | name   | rules
---+------+--------+--------------------------------------------
 0 | #4:2 | writer | {database=2, database.schema=7,
   |      |        | database.cluster.internal=2,
   |      |        | database.cluster.orole=2,
   |      |        | database.cluster.ouser=2,
   |      |        | database.class.*=15, database.cluster.*=15,
   |      |        | database.command=15, database.hook.record=15}
---+------+--------+--------------------------------------------

3 item(s) found. Query executed in 0.045 sec(s).
```

Running these commands shows that you do in fact have a `writer` role configured on your database and that it uses a Record ID of `#4:2`. You can now create the users. There are two methods available to you, depending on which version of OrientDB you use.

Creating Users

Beginning with version 2.1, OrientDB now features a `CREATE USER` command. This allows you to create a new user in the current database, as opposed to inserting the credentials into the `OUser` and `ORole` classes.

To create users for Luca and Steve, run the following commands:

```
orientdb {db=blog}> CREATE USER luca IDENTIFIED BY luca_passwd
                ROLE writer

orientdb {db=blog}> CREATE USER steve IDENTIFIED BY steve_passwd
                ROLE writer
```

The users are now active on your database as `luca` and `steve`.

Inserting Users

For older implementations of OrientDB, there is no `CREATE USER` command available to you. To add users before version 2.1, you need to use `INSERT` statements to add the new values into the `OUser` class, using the record ID for the `writer` role, which you found above as `#4:2`:

```
orientdb {db=blog}> INSERT INTO OUser SET name = 'luca',
                status = 'ACTIVE', password = 'luca_passwd',
                roles = [#4:2]

Inserted record 'OUser#5:4{name:luca,password: {SHA-256}D70F47790F
689414789EEFF231703429C7F88A10210775906460EDBF38589D90,roles:[1]}
v1' in 0,001000 sec(s).

orientdb {db=blog}> INSERT INTO OUser SET name = 'steve',
                status = 'ACTIVE', password = 'steve_passwd',
                roles = [#4:2]

Inserted record 'OUser#5:3{name:steve,password: {SHA-256}F148389D0
80CFE85952998A8A367E2F7EAF35F2D72D2599A5B0412FE4094D65C,roles:[1]}
v1' in 0,001000 sec(s).
```

Creating Graphs

In order to work with the partition, you need to create graphs for each user on the database. This requires that you log out from the `admin` user and log back in as Luca and then Steve, then create vertexes and edges with which to work.

Create a Graph for Luca

First, using `DISCONNECT` and `CONNECT` disconnect from your admin session on the `blog` database and reconnect as Luca's user.

```
orientdb {db=blog}> DISCONNECT
Disconnecting from the database [blog]...OK

orientdb> CONNECT local:../databases/blog luca luca_passwd
Connecting to database [local:../databases/blog] with user 'luca'...OK

orientdb {db=blog}>
```

Now that you're logged in under Luca's user, using the `CREATE VERTEX` command, create two vertices: one for restaurants and one for pizza.

```
orientdb {db=blog}> CREATE VERTEX SET label = 'food',
                name = 'Pizza'
Created vertex 'V#9:0{label:food,name:Pizza,_allow:[1]} v0' in 0,001000 sec(s).

orientdb {db=blog}> CREATE VERTEX SET label = 'restaurant',
                name = "Dante's Pizza"
Created vertex 'V#9:1{label:restaurant,name:Dante's Pizza,_allow:[1]} v0' in 0,000000
sec(s).
```

Connect these vertices with an edge for menus, using `CREATE EDGE` :

```
orientdb {db=blog}> CREATE EDGE FROM #9:0 TO #9:1 SET label = 'menu'
Created edge '[E#10:0{out:#9:0,in:#9:1,label:menu,_allow:[1]}
v1]' in 0,003000 sec(s).
```

You can check the status using `SELECT` against these vertices:

```
orientdb {db=blog}> SELECT FROM V

 ---+------+------------+---------------+-----------------
  # | RID  | label      | name          | _allow | out
 ---+------+------------+---------------+--------+--------
  0 | #9:0 | food       | Pizza         | [1]    | [1]
  1 | #9:1 | restaurant | Dante's Pizza | [1]    | null
 ---+------+------------+---------------+--------+--------
 2 item(s) found. Query executed in 0.034 sec(s).
```

Creating a Graph for Steve

Now let's connect to the database using the 'Steve' user and check if there are vertices:

```
orientdb {db=blog}> DISCONNECT
Disconnecting from the database [blog]...OK

orientdb> CONNECT local:../databases/blog steve steve_passwd
Connecting to database [local:../databases/blog] with user 'steve'...OK

orientdb {db=blog}> SELECT FROM V

0 item(s) found. Query executed in 0.0 sec(s).
```

This confirms that the user Steve does not have access to vertices created by Luca. Now, create some as Steve:

```
orientdb {db=blog}> CREATE VERTEX SET label = 'car',
              name = 'Ferrari Modena'

Created vertex 'V#9:2{label:car,name:Ferrari Modena,_allow:[1]} v0' in 0,000000 sec(s).

orientdb {db=blog}> CREATE VERTEX SET label = 'driver',
              name = 'steve'

Created vertex 'V#9:3{label:driver,name:steve,_allow:[1]} v0' in 0,000000 sec(s).

orientdb {db=blog}> CREATE EDGE FROM #9:2 TO #9:3
              SET label = 'drive'

Created edge '[E#10:1{out:#9:2,in:#9:3,label:drive,_allow:[1]} v1]' in 0,002000 sec(s).
```

Run the `SELECT` query from earlier to see the vertices you've created:

```
orientdb {db=blog}> SELECT FROM V

---+------+--------+----------------+--------+------
 # | RID  | label  | name           | _allow | out
---+------+--------+----------------+--------+------
 0 | #9:2 | car    | Ferrari Modena | [1]    | [1]
 1 | #9:3 | driver | steve          | [1]    | null
---+------+--------+----------------+--------+------

2 item(s) found. Query executed in 0.034 sec(s).
```

As you can see, Steve's user still can't see vertices and edged that were created by other users. For the sake of example, see what happens when you try to create an edge that connects vertices from different users:

```
orientdb {db=blog}> CREATE EDGE FROM #9:2 TO #9:0
              SET label = 'security-test'

Error: com.orientechnologies.orient.core.exception.OCommandExecutionException:
Error on execution of command: OCommandSQL [text=create edge from #9:2 to #
9:0 set label = 'security-test']
Error: java.lang.IllegalArgumentException: Source vertex '#9:0' does not exist
```

The partition used by Luca remains totally isolated from the one used by Steve. OrientDB operates on the assumption that the other partition doesn't exist, it remains invisible to the current user while still present in the database.

TinkerPop Stack

The Record-level Security feature is very powerful because it acts at a low-level within the OrientDB engine. This allows for better integration of security features with the Java API and the TinkerPop stack.

For instance, try to display all vertices and edges using Gremlin:

```
gremlin> g.V
==> [v[#9:2], v[#9:3]]
Script executed in 0,448000 sec(s).

gremlin> g.E
==> e[#10:1][#9:2-drive->#9:3]
Script executed in 0,123000 sec(s).
```

> This feature works with other technologies that rely on TinkerPop Blueprints:
>
> - TinkerPop Rexter
> - TinkerPop Pipes
> - TinkerPop Furnace
> - TinkerPop Frames
> - ThinkAurelius Faunus

Graph Database Comparison

This is a comparison page between GraphDB projects. To know more about the comparison of DocumentDBs look at this comparison.

We want to keep it always updated with the new products and more features in the matrix. If any information about any product is not updated or wrong, please change it if you've the permissions or send an email to any contributors with the link of the source of the right information.

Feature matrix

Feature	OrientDB	Neo4j	DEX	InfiniteGraph
Release	1.0-SNAPSHOT	1.7M03	4.5.1	2.1
Product Web Site	http://www.orientdb.org	http://www.neo4j.org	http://www.sparsity-technologies.com	http://objectivity.com/INFINIT
License	Open Source Apache 2	Open Source GPL, Open Source AGPL and Commercial	Commercial	Commercial
Query languages	Extended SQL, Gremlin	Cypher Gremlin	Not available, only via API	Gremlin, Java API
Transaction support	☐ ACID	☐ ACID	☐	☐ ACID
Protocols	Embedded via Java API, remote as Binary and REST	Embedded via Java API and remote via REST	?	Embedded via Java API, Remot access via TCP
Replication	Multi-Master	Master-Slave	No	☐
Custom types	☐ Supports custom types and polymorphism	☐	☐	☐ Supports custom types anc polymorphism
Self loops	☐	☐	☐	☐

Blueprints support

The products below all support the TinkerPop Blueprints API at different level of compliance. Below the supported ones. As you can see OrientDB is the most compliant implementation of TinkerPop Blueprints!

Feature	OrientDB	Neo4j	DEX	
Release	1.0-SNAPSHOT	1.7M03	4.5.1	2.1
Product Web Site	http://www.orientdb.org	http://www.neo4j.org	http://www.sparsity-technologies.com	http://objec
Implementation details	OrientDB impl	Neo4j impl	DEX impl	InfiniteGra
allowsDuplicateEdges	☐	☐	☐	?
allowsSelfLoops	☐	☐	☐	?
isPersistent	☐	☐	☐	?
supportsVertexIteration	☐	☐	☐	?
supportsEdgeIteration	☐	☐	☐	?
supportsVertexIndex	☐	☐	☐	?
supportsEdgeIndex	☐	☐	☐	?
ignoresSuppliedIds	☐	☐	☐	?
supportsTransactions	☐	☐	☐	?
allowSerializableObjectProperty	☐	☐	☐	?
allowBooleanProperty	☐	☐	☐	?
allowDoubleProperty	☐	☐	☐	?
allowFloatProperty	☐	☐	☐	?
allowIntegerProperty	☐	☐	☐	?
allowPrimitiveArrayProperty	☐	☐	☐	?
allowUniformListProperty	☐	☐	☐	?
allowMixedListProperty	☐	☐	☐	?
allowLongProperty	☐	☐	☐	?
allowMapProperty	☐	☐	☐	?
allowStringProperty	☐	☐	☐	?

Micro benchmark

The table below reports the time to complete the Blueprints Test Suite. This is **not a benchmark between GraphDBs** and unfortunately doesn't exist a public benchmark shared by all the vendors :-(

So this table is just to give an idea about the performance of each implementation in every single module it supports. The support is based on the compliance level reported in the table above. For the test default settings were used. To run these tests on your own machine follow these simple instructions.

Lower means faster. In **bold** the fastest implementation for each module.

Module	OrientDB	Neo4j	DEX	I
Release	1.4	1.9.M05	4.8.0	2.1
Product Web Site	http://www.orientdb.org	http://www.neo4j.org	http://www.sparsity-technologies.com	http://objectiv
VertexTestSuite	**1,524.06**	1,595.27	4,488.28	?
EdgeTestSuite	**1,252.21**	1,253.73	3,865.85	?
GraphTestSuite	**1,664.75**	2,400.34	4,680.80	?
QueryTestSuite	306.58	**188.52**	612.73	?
IndexableGraphTestSuite	4,620.61	11,299.02	**1070.75**	?
IndexTestSuite	**2,072.23**	5,239.92	not supported	?
TransactionalGraphTestSuite	**1,573.93**	3,579.50	not supported	?
KeyIndexableGraphTestSuite	**571.42**	845.84	not supported	?
GMLReaderTestSuite	778.08	**682.83**	not supported	?
GraphMLReaderTestSuite	**814.38**	864.70	2,316.79	?
GraphSONReaderTestSuite	**424.77**	480.81	1223.24	?

All the tests are executed against the same HW/SW configuration: MacBook Pro (Retina) 2013 - 16 GB Ram - MacOSX 12.3.0 - SDD 7200rpm. Similar results executed on Linux CentOS.

Run the tests

To run the Blueprints Test Suite you need java6+, Apache Maven and Git. Follow these simple steps:

1. `> git clone git://github.com/tinkerpop/blueprints.git`
2. `> mvn clean install`

Lightweight Edges

In OrientDB Graph databases, there are two types of edges available:

- **Regular Edges**, which are instances or extensions of the `E` class and have their own records on the database.
- **Lightweight Edges**, which have no properties and have no identity on the database, since they only exist within the vertex records.

The advantage of Lightweight Edges lies in performance and disk usage. Similar to links, because it doesn't create the underlying document it runs faster and saves on space, with the addition of allowing for bidirectional connections. This allows you to use the `MOVE VERTEX` command to refactor your graph without breaking any links.

> **NOTE**: Beginning in version 2.0, OrientDB disables Lightweight Edges by default.

Edge Representations

In order to better understand edges and their use, consider the differences in how OrientDB handles and stores them in records. The examples below show a Graph Database built to form a social network where you need to connect to `Account` vertices by an edge to indicate that one user is friends with another.

Regular Edge Representation

When building a social network using regular edges, the edge exists as a separate record, which is either an instance of `E` or an extension of that class.

```
+---------------------+    +---------------------+    +---------------------+
|   Account Vertex    |    |     Friend Edge     |    |    Account Vertex   |
|       #10:33        |    |       #17:11        |    |       #10:12        |
+---------------------+    +---------------------+    +---------------------+
|out_Friend: [#17:11] |<-->|out: [#10:33]        |    |                     |
+---------------------+    |         in: [#10:12]|<-->|in_Friend: [#17:11]  |
                           +---------------------+    +---------------------+
```

When using regular edges, the vertices are connected through an edge document. Here, the account vertices for users #10:33 and #10:12 are connected by the edge class instance #17:11, which indicates that the first user is friends with the second. The outgoing `out_Friend` property on #10:33 and the incoming `in_Friend` property on #10:12 each take #17:11 as a link to that edge.

When you cross this relationship, OrientDB loads the edge document #17:11 to resolve the other side of the relationship.

Lightweight Edge Representation

When building a social network using Lightweight Edges, the edge doesn't exist as a document, but rather are connected directly to each other through properties on the relevant vertices.

```
+---------------------+    +---------------------+
|   Account Vertex    |    |    Account Vertex   |
|       #10:33        |    |       #10:12        |
+---------------------+    +---------------------+
|out_Friend: [#10:12] |<-->|in_Friend: [#10:33]  |
+---------------------+    +---------------------+
```

When using Lightweight Edges, instead of an edge document, each vertex (here, #10:33 and #10:12), are connected directly to each other. The outgoing `out_Friend` property in the #10:30 document contains the direct link to the vertex #10:12. The occurs in #10:12, where the incoming `in_Friend` property contains a direct link to #10:30.

When you cross this relationship, OrientDB can register the relationship without needing to load a third record to resolve the relationship, thus it doesn't create an edge document.

Enabling Lightweight Edges

Beginning in version 2.0, OrientDB disables Lightweight Edges by default on new databases. The reasoning behind this decision is that it is easier for users and applications to operate on OrientDB from SQL when the edges are regular edges. Enabling it by default created a number of issues with beginners just starting out with OrientDB who were unprepared for how it handles Lightweight Edges.

In the event that you would like to restore Lightweight Edges as the default, you can do so by updating the database. From the Java API, add the following lines:

```
OrientGraph g = new OrientGraph("mygraph");
g.setUseLightweightEdges(true);
```

You can do the same from the Console using the `ALTER DATABASE` command:

```
orientdb> ALTER DATABASE custom useLightweightEdges=true
```

Note that changing the `useLightweightEdges` to `true` does not update or transform any existing edges on the database. They remain in the same state they held before you issued the command. That said, all new edges you create after running the above commands are Lightweight Edges.

Lightweight Edges vs. Regular Edges

There are advantages and disadvantages to using Lightweight Edges. You should consider these before switching your application over to one or the other:

Advantages

- *Faster Creation and Traversal*: Given that OrientDB can operate on the relationship between two vertices without needing to load any additional documents, it's able to create and traverse relationships faster with Lightweight Edges than it can with Regular Edges.

Disadvantages

- *Edge Properties*: Given that Lightweight Edges do not have records of their own, you cannot assign properties to the edge.
- *SQL Issues*: It is more difficult to interact with OrientDB through SQL given that there is no document underlying the edges.

Graph Batch Insert

Creating big graphs in OrientDB is a common operation, so OrientDB provides some APIs to make it fast and easy.

Basic use case

For a basic use case, where your graph fits in the following constraints:

- a single vertex class
- a single edge class
- vertices are identified by a numeric (Long) id
- no properties on vertices and edges OrientDB provides an API called OGraphBatchInsertBasic

This API is designed for fast batch import of simple graphs, starting from an empty (or non existing) DB.

These limitations are intended to have best performance on a very simple use case. If these limitations do not fit your requirements you can rely on other implementations.

Typical usage:

```
OGraphBatchInsertSimple batch = new OGraphBatchInsertSimple("plocal:your/db", "admin", "admin");
batch.begin();
batch.createEdge(0L, 1L);
batch.createEdge(0L, 2L);
...
batch.end();
```

There is no need to create vertices before connecting them:

```
batch.createVertex(0L);
batch.createVertex(1L);
batch.createEdge(0L, 1L);
```

is equivalent to (but less performing than):

```
batch.createEdge(0L, 1L);
```

`batch.createVertex()` is needed only if you want to create unconnected vertices.

Slightly more complex use case

For a use case, where your graph fits in the following constraints:

- a single vertex class
- a single edge class
- vertices are identified by a numeric (Long) id
- edges and/or vertices have properties

OrientDB provides an API called OGraphBatchInsert.

This API is designed for fast batch import of simple graphs, starting from an empty (or non existing) DB.

This batch insert procedure is made of four phases, that have to be executed in the correct order:

1. begin(): initializes the database
2. create edges (with or without properties) and vertices
3. set properties on vertices
4. end(): flushes data to db

Typical usage:

```
OGraphBatchInsert batch = new OGraphBatchInsert("plocal:your/db", "admin", "admin");

//phase 1: begin
batch.begin();

//phase 2: create edges
Map<String, Object> edgeProps = new HashMap<String, Object>();
edgeProps.put("foo", "bar");
batch.createEdge(0L, 1L, edgeProps);
batch.createVertex(2L);
batch.createEdge(3L, 4L, null);
...

//phase 3: set properties on vertices, THIS CAN BE DONE ONLY AFTER EDGE AND VERTEX CREATION
Map<String, Object> vertexProps = new HashMap<String, Object>();
 vertexProps.put("foo", "bar");
batch.setVertexProperties(0L, vertexProps);
...

//phase 4: end
batch.end();
```

There is no need to create vertices before connecting them:

```
    batch.createVertex(0L);
    batch.createVertex(1L);
    batch.createEdge(0L, 1L, props);
```

is equivalent to (but less performing than):

```
    batch.createEdge(0L, 1L, props);
```

`batch.createVertex(Long)` is needed only if you want to create unconnected vertices.

Custom Batch Insert

Creating graphs consists mainly of two operations:

- creating vertices
- connecting them with edges

Adding a single edge to an existing database actually consists of three operations:

- creating the edge document
- updating the left vertex to point to the edge
- updating the right vertex to point to the edge

and typically, at low level, it's even more complex:

- load vertex1 by key (index lookup)
- load vertex2 by key (index lookup)
- create edge document setting out = vertex1.@rid and in = vertex2.@rid
- add the edge RID to vertex1.out_EdgeClass
- add the edge RID to vertex2.in_EdgeClass

As a result, the creation of an edge can be considered an expensive operation compared to a simple document creation.

In some circumstances, the batch graph creation can be made faster.

Taking into consideration that RIDs are assigned sequentially for clusters, that edges and vertices are just ODocuments and that `out_*` and `in_*` properties for vertices can be manually manipulated at document level, in some circumstances (based on your specific domain structure) you can write a custom piece of code that "predicts" RIDs that will be assigned to edge and vertex documents, and do the edge creation as a single write operation.

Below a simple example.

Suppose that you have to create a graph like the following:

```
Vertex1 -Edge1-> Vertex2 -Edge2-> Vertex3
```

Vertex class is `VertexClass` and edge class is `EdgeClass`.

Let's suppose that vertices will be inserted in cluster `9 (cluster vertexclass)` and that edges will be inserted in cluster `10 (cluster edgeclass)`. Let's also suppose that both clusters are empty and newly created.

If you insert all the vertices in the given order, you will be sure that:

- Vertex1 will have @RID = #9:0
- Vertex2 will have @RID = #9:1
- Vertex3 will have @RID = #9:2

If you insert all the edges in the given order, you will be sure that:

- Edge1 will have @RID = #10:0
- Edge2 will have @RID = #10:1

> NOTE: When using transactions, the order of RID assignment can change, so this technique could not work.

This said, you can use the Document API to create the graph structure:

```
ORecordId ridVertex1 = new ORecordId(9L, 0L); // #9:0, the RID of Vertex1, that still does not exist
ORecordId ridVertex2 = new ORecordId(9L, 1L); // #9:1, the RID of Vertex2, that still does not exist
ORecordId ridVertex3 = new ORecordId(9L, 2L); // #9:2, the RID of Vertex3, that still does not exist

ORecordId ridEdge1 = new ORecordId(10L, 0L); // #10:0, the RID of Edge1, that still does not exist
ORecordId ridEdge2 = new ORecordId(10L, 1L); // #10:1, the RID of Edge2, that still does not exist

ODocument vertex1 = new ODocument("VertexClass");
vertex1.field("foo", "bar");// set property names and values
ORidBag outBag1 = new ORidBag();
outBag1.add(ridEdge1); // add the RID of the corresponding outgoing edge
vertex1.field("out_EdgeClass", outBag1);
db.save(vertex1, "vertexclass"); //make sure that you are saving on the right cluster (vertexclass)

/*
At this point, in the databse you have a single record (#9:0) that represents a vertex.
You know that the record RID is #9:0 because it's the first record created in cluster 9.
The vertex points to an edge (#10:0) that still does not exist and that will be created later
*/



ODocument vertex2 = new ODocument("VertexClass");
vertex2.field("foo", "bar");// set property names and values
ORidBag outBag2 = new ORidBag();
outBag2.add(ridEdge2); // add the RID of the corresponding outgoing edge
vertex2.field("out_EdgeClass", outBag2);
ORidBag inBag2 = new ORidBag();
inBag2.add(ridEdge1); // add the RID of the corresponding incoming edge
vertex2.field("in_EdgeClass", inBag2);
db.save(vertex2, "vertexclass"); //make sure that you are saving on the right cluster

ODocument vertex3 = new ODocument("VertexClass");
vertex3.field("foo", "bar");// set property names and values
ORidBag inBag3 = new ORidBag();
inBag3.add(ridEdge2); // add the RID of the corresponding incoming edge
vertex3.field("in_EdgeClass", inBag3);
db.save(vertex3, "vertexclass"); //make sure that you are saving on the right cluster

ODocument edge1 = new ODocument("EdgeClass");
edge1.field("foo", "bar"); //set edge fields
edge1.field("out", ridVertex1); //set pointers to the right vertices
edge1.field("in", ridVertex2);
db.save(edge1, "edgeclass"); //make sure that you are saving on the right cluster (edgeclass)

ODocument edge2 = new ODocument("EdgeClass");
edge2.field("foo", "bar"); //set edge fields
edge2.field("out", ridVertex2); //set pointers to the right vertices
edge2.field("in", ridVertex3);
db.save(edge2, "edgeclass"); //make sure that you are saving on the right cluster
```

As you can see, this batch insert consists of exactly five write (insert) operations, and no load and update operations are made.

Saving documents with links to other not (yet) existing documents is allowed, so saving a vertex that points to a non existing edge will result in a correct operation. The graph consistency will be restored as soon as you create the edge.

Document API

The Document Database in OrientDB is the foundation of higher level implementations, like the Object Database and the Graph Database. The Document API supports:

- Multi-thread Access
- Transactions
- Queries
- Traverse

It is also very flexible and can be used in schema-full, schema-less or mixed schema modes.

```
// OPEN THE DATABASE
ODatabaseDocumentTx db = new ODatabaseDocumentTx(
   "remote:localhost/petshop").open("admin", "admin_passwd");

// CREATE A NEW DOCUMENT AND FILL IT
ODocument doc = new ODocument("Person");
doc.field( "name", "Luke" );
doc.field( "surname", "Skywalker" );
doc.field( "city", new ODocument("City")
   .field("name","Rome")
   .field("country", "Italy") );

// SAVE THE DOCUMENT
doc.save();

db.close();
```

Bear in mind, in this example that we haven't declared the `Person` class previously. When you save this `ODocument` instance, OrientDB creates the `Person` class without constraints. For more information on declaring persistent classes, see Schema Management.

In order to use the Document API, you must include the following jars in your classpath:

- `orientdb-core-*.jar`
- `concurrentlinkedhashmap-lru-*.jar`

Additionally, in the event that you would like to interface with the Document API on a remote server, you also need to include these jars:

- `orientdb-client-*.jar`
- `orientdb-enterprise-*.jar`

> For more information, see
>
> - Javadoc: JavaDoc
> - OrientDB Studio Web tool.

Using the Document Database

In order to work with documents in your application, you first need to open a database and create an instance in your code. For instance, when opening on localhost through the `remote` interface, you might use something like this:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx
   ("remote:localhost/petshop")
   .open("admin", "admin_passwd");
```

Once you have the database open, you can create, update and delete documents, as well as query document data to use in your application.

> For more information, see
>
> - Working with Document Databases
> - Working with Documents

Creating Documents

You can create documents through the `ODocument` object by setting up the document instance and then calling the `save()` method on that instance. For instance,

```
ODocument animal = new ODocument("Animal");
animal.field("name", "Gaudi");
animal.field("location", "Madrid");
animal.save()
```

Updating Documents

You can update persistent documents through the Java API by modifying the document instance and then calling `save()` on the database. Alternatively, you can call `save()` on the document instance itself to synchronize the changes to the database.

```
animal.field( "location", "Nairobi" );
animal.save();
```

When you call `save()`, OrientDB only updates the fields you've changed.

> **NOTE**: This behavior can vary depending on whether you've begun a transaction. For more information, see Transactions.

For instance, using the code below you can raise the price of animals by 5%:

```
for (ODocument animal : database.browseClass("Animal")) {
  animal.field("price", animal.field("price") * 1.05);
  animal.save();
}
```

Deleting Documents

Through the Java API, you can delete documents through the `delete()` method on the loaded document instance. For instance,

```
animal.delete();
```

> **NOTE**: This behavior can vary depending on whether you've begun a transaction. For more information, see Transactions.

For instance, using the code below you can delete all documents of the `Animal` class:

```
for (ODocument animal : database.browseClass("Animal")) {
   animal.delete();
}
```

Schema

OrientDB is flexible when it comes to schemas. You can use it,

- **Schema Full**, such as in Relational Databases.
- **Schema Less**, such as with many NoSQL Document Databases.
- **Scheme Hybrid**, which mixes the two.

> For more information, see Schemas.

In order to use the schema with documents, create the `ODocument` instance with the constructor `ODocument(String className)`, passing it the class name as an argument. If you haven't declared the class, OrientDB creates it automatically with no fields. This won't work during transactions, given it can't apply schema changes in transactional contexts.

Security

Few NoSQL implementations support security. OrientDB does. For more information on its security features, see Security.

To manage the security from within the Java API, get the Security Manager and use it to operate on users and roles. For instance,

```
OSecurity sm = db.getMetadata().getSecurity();
OUser user = sm.createUser(
   "john.smith", "smith_passwd",
   new String[]{"admin"});
```

Once you've have users on the database, you can retrieve information about them through the Java API:

```
OUser user = db.getUser();
```

Transactions

Transactions provide you with a practical way to group sets of operations together. OrientDB supports ACID transactions, ensuring that either all operations succeed or none of them do. The database always remains consistent.

> For more information, see Transactions.

OrientDB manages transactions at the database-level. Currently, it does not support nesting transactions. A database instance can only have one transaction running at a time. It provides three methods in handling transactions:

- **`begin()`** Starts a new transaction. If a transaction is currently running, it's rolled back to allow the new one to start.
- **`commit()`** Ends the transaction, making the changes persistent. If an error occurs during the commit, the transaction is rolled back and it raises an `OTransactionException` exception.
- **`rollback()`** Ends the transaction and removes all changes.

Optimistic approach

In its current release, OrientDB uses Optimistic Transactions, in which no lock is kept and all operations are checked on the commit. This improves concurrency, but can throw an `OConcurrentModificationException` exception in cases where the records are modified by concurrent client threads. In this scenario, the client code can reload the updated records and repeat the transaction.

When optimistic transactions run, they keep all changes in memory on the client. If you're using remote storage, it sends no changes to the server until you call the `commit()` method. All changes transfer in a block to reduce network latency, speed up the execution and increase concurrency. This differs from most Relational Databases, where changes are sent immediately to the server during transactions.

Usage

OrientDB commits transactions only when you call the `commit()` method and no errors occur. The most common use case for transactions is to enclose all database operations within a `try` / `finally` blocks. When you close the database, (that is, when your application executes the `finally` block), OrientDB automatically rolls back any pending transactions still running.

For instance,

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx(url);
db.open("admin", "admin_passwd");

db.begin();
try {
  // YOUR CODE
  db.commit();
} finally {
  db.close();
}
```

Index API

While you can set up and configure Indices through SQL, the recommended and most efficient way is through the Java API.

The main class to use when working with indices is the `OIndexManager`

```
OIndexManager idxManager = database.getMetadata().getIndexManager();
```

The Index Manager allows you to manage the index life-cycle for creating, deleting and retrieving index instances. The most common usage is through a single index. You can get the index reference:

```
OIndex<?> idx = database.getMetadata().getIndexManager()
   .getIndex("Profile.name");
```

Here, `Profile.name` is the index name. By default, OrientDB assigns the name as `<class-name>.<property-name>` for automatic indices created against the class property.

The `OIndex` interface is similar to a Java Map and provides methods to get, put, remove and count items.

Below are examples of retrieving records using a `UNIQUE` index against aname field and a `NOTUNIQUE` index against the gender field:

```
OIndex<?> nameIdx = database.getMetadata().getIndexManager()
   .getIndex("Profile.name");

// THIS IS A UNIQUE INDEX, SO IT RETRIEVES A OIdentifiable
OIdentifiable luke = nameIdx.get("Luke");
if( luke != null )
   printRecord((ODocument) luke.getRecord());

OIndex<?> genderIdx = database.getMetadata().getIndexManager()
   .getIndex("Profile.gender");

// THIS IS A NOTUNIQUE INDEX, SO IT RETRIEVES A Set<OIdentifiable>
Set<OIdentifiable> males = genderIdx.get( "male" );
for (OIdentifiable male : males)
   printRecord((ODocument) male.getRecord());
```

OrientDB manages automatic indices using hooks, you can use manual indices to store values. To create a new entry, use the `put()` method:

```
OIndex<?> addressbook = database.getMetadata().getIndexManager()
   .getIndex("addressbook");

addressbook.put( "Luke", new ODocument("Contact").field( "name", "Luke" ) );
```

Working with Document Databases

Before you can execute any operation on a Document Database, you first need to open an instance in your application. You can do so by either opening an existing instance or creating a new one. Bear in mind that database instances are not thread-safe, so only use one database per thread.

Whether you want to open or create a database, you first need a valid database URL. The URL defines where OrientDB looks for the database and what kind it should use. For instance, `memory` refers to a database that is in-memory only and volatile, `plocal` to one that is embedded and `remote` to a database either on a remote server or accessed through localhost. For more information, see Database URL.

Managing Database Instances

When you finish with a database instance, you must close it in order to free up the system resources that it uses. To ensure this, the most common practice is to enclose the database operation within a `try` / `finally` block. For instance,

```
// Open the /tmp/test Document Database
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/tmp/test");
db.open("admin", "admin");

try {
     // Enter your code here...
} finally {
     db.close()
}
```

Using this layout, the database is automatically closed once it finishes executing your code. If you want to open the database in memory, use `memory:` or `remote:` if you want to use a remote instance instead a PLocal connection.

> Remember, the `ODatabaseDocumentTx` class is not thread-safe. When using multiple threads, use separate instances of this class. This way, they share the same storage instance, (with the same Database URL), and the same level-2 cache.
>
> For more information, see Multi-Threading with Java.

Creating New Databases

In the event that the database doesn't exist already, you can create one through the Java API. From the local file system, this is relatively straight forward, using `plocal` :

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx
        ("plocal:/tmp/database/petshop")
        .create();
```

When you create a db for the first time, OrientDB creates three users and three roles for you

Users:

- User `admin` (password "admin") with role `admin`
- User `reader` (password "reader") with role `reader`
- User `writer` (password "writer") with role `writer`

Roles:

- Role `admin` - full control on the database
- Role `reader` - read only permissions
- Role `writer` - read/write permissions, but no schema manipulation

For more information on how to add/remove users/roles, change roles to a user, change passwords, please refer to Database-Security

For creating database instances on remote servers, the process is a little more complicated. You need the user and password to access the remote OrientDB Server instance. By default, OrientDB creates the `root` user when the server first starts. Checking the file in `$ORIENTDB_HOME/config/orientdb-server-config.xml` , which also provides the password.

To create a new document database, here `ExampleDB` at the address `dbhost` using file system storage, add the following lines to your application:

```
new OServerAdmin("remote:dbhost")
      .connect("root", "root_passwd")
      .createDatabase("ExampleDB", "document", "plocal").close();
```

This uses the `root` user to connect to the OrientDB Server, then creates the `ExampleDB` Document Database. To create a graph database, replace `document` with `graph` . To store the database in memory, replace `plocal` with `memory` .

Opening Existing Databases

When dealing with existing databases, you need to open the instance instead of creating it. The database instance shares the connection, rather than the storage. If it's a `plocal` storage, then all the database instances synchronize on it. IF it's a `remote` storage, then all the database instances share the network connection.

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx
      ("remote:localhost/petshop")
      .open("admin", "admin"); //default password
```

Using Database Pools

It is not always the best practice to create database instances every time you need them. In cases where your application needs to scale to many database connections, such as in the case of a web app, it's often better to pool the instances together to improve performance:

```
// OPEN DATABASE
OPartitionedDatabasePool pool =  new OPartitionedDatabasePool(url , "admin", "admin_passswrd");
ODatabaseDocumentTx db = pool.acquire();

try {
   // YOUR CODE
   ...
} finally {
   db.close()
}

// eventually close the pool with pool.close()
```

Remember to close the database instance using the `.close()` database method as you would with classic non-pooled databases. In cases like the above example, closing the database instance does not actually close it. Rather, closing it releases the instance back to the pool, where it's made available to future requests.

> **NOTE**: It's best practice to use the `try` / `finally` blocks here, as it helps you to ensure the database instances are returned to the pool rather than left open.

Database pool size

You can manually define the size of the database pool (ie. the minimum and maximum number of database instances in the pool) explicitly passing these values to the constructor. Bear in mind, when using deployments of this kind, you need to close the pool when it's no longer needed:

```
// CREATE A NEW POOL WITH 1 - 10 INSTANCES
OPartitionedDatabasePool pool =  new OPartitionedDatabasePool(url , "admin", "admin_passswrd", 1, 10);
...
pool.close()
```

Dropping a database

To drop a database (`plocal`) ODatabaseDocumentTx provides the following API:

```
db.drop();
```

To drop a db in `remote` , you can use the following:

```
new OServerAdmin("remote:dbhost")
      .connect("root", "root_passwd")
      .dropDatabase("dbName", "plocal");
```

Working with Documents

When you have data ready in your database, you need a way to access it in order to manipulate it through your application. There are three ways to do this:

- Retrieving Documents through the Java API.
- Querying Documents through SQL.
- Traversing Documents through the Java Traverse API.

Retrieving Documents

Using the Java API, you can retrieve documents into an object or access the database otherwise using the `ODocument` object. If you're more comfortable working in Java than SQL, this solution may work best for you.

- Browse all documents in a cluster:

  ```
  for (ODocument doc : database.browserCluster("CityCars")) {
     System.out.println(doc.field("model"));
  }
  ```

- Browe all records in a class:

  ```
  for (ODocument animal : database.browseClass("Animal")) {
     System.out.println(animal.field("name"));
  }
  ```

- Count records in a class:

  ```
  long cars = database.countClass("Cars");
  ```

- Count records in a cluster:

  ```
  long cityCars = database.countCLuster("CityCar");
  ```

Querying Documents

While OrientDB is a NoSQL database implementation, it does support a subset of SQL. This allows it to process links to documents and graphs. For example,

```
List<ODocument> result = db.query(
   new OSQLSynchQuery<ODocument>(
      "SELECT FROM Animal WHERE id = 10
      AND NAME LIKE 'G%'"
   )
);
```

For more information on the OrientDB syntax, see SQL.

> **NOTE**: OrientDB is a Graph Database. This means that it is very efficient at traversals. You can use this feature to optimize your queries, such as with pivoting.

Asynchronous Queries

In addition to the standard SQL queries, OrientDB also has support for asynchronous queries. Here, the result is not collected and returned in a synchronous manner, (as above), but rather it uses a callback every time it finds a record that satisfies the predicates.

```
database.command(
   new OSQLAsynchQuery<ODocument>(
      "SELECT FROM Animal WHERE name = 'Gipsy'",
      new OCommandResultListener() {
         resultCount = 0;
         @Override
         public boolean result(Object iRecord) {
            resultCount++;
            ODocument doc = (ODocument) iRecord;

            // ENTER YOUR CODE TO WORK WITH DOCUMENT

            return resultCount > 20 ? false: true;
         }

         @Override
         public void end() {}
      }
   )
).execute();
```

When OrientDB executes an asynchronous query, it only needs to allocate memory for each of the individual callbacks as it encounters them. You may find this a huge benefit in cases where you need to work with large result-sets. Asynchronous Query are not designed for data manipulation, avoid executed update or transaction inside the listener code.

Non-Blocking Queries

Both synchronous and asynchronous queries are blocking queries. What this means is that the first instruction you issue after `db.query()` or `db.command().execute()` executes only after you invoke the last callback or receive the complete result-set.

Beginning in version 2.1, OrientDB introduces support for non-blocking queries. These use a similar API to asynchronous queries. That is, you have a callback that gets invoked for every record in the result-set. However, the behavior is very different. Execution on the current thread continues without blocking on `db.query()` or `db.command().execute()` while it invokes the callback on a different thread. This means that you can close the database instance while still receiving callbacks from the query result.

```
Future future = database.command(
   new OSQLNonBlockingQuery<Object>(
      "SELECT FROM Animal WHERE name = 'Gipsy'",
      new CommandResultListener(){
         resultCount = 0;
         @Override
         public boolean result(Object iRecord){
            // ENTER YOUR CODE HERE

            System.out.print("callback "+resultCount+" invoked");
            return resultCount > 20 ? false: true;
         }

         @Override
         public void end(){}
      }
   )
).execute();

System.outprintln("query executed");

future.get();
```

When the code executes, the results look something like this:

```
query executed
callback 0 invoked
callback 1 invoked
callback 2 invoked
callback 3 invoked
callback 4 invoked
```

You might also get results something like this:

```
query executed
callback 0 invoked
callback 1 invoked
query executed
callback 2 invoked
callback 3 invoked
callback 4 invoked
```

Whether this occurs depends on race conditions on the two parallel threads. That is, a case where one fires the query execution and then continues with "query executed", while the other invokes the callbacks.

The `future.get()` method is a blocking call that returns only after the last callback invocation. You can avoid this in cases where you don't need to know when the query terminates.

Prepared Queries

Similar to the Prepared Statement of JDBC, OrientDB now supports prepared queries. With prepared queries, the database pre-parses the query so that, in cases of multiple executions of the same query, they run faster than the class SQL queries. Furthermore, the pre-parsing mitigates SQL Injection.

Prepared queries use two kinds of markers to substitute parameters on execution:

- **`?`** Syntax is used in reference to position parameters. For instance,

  ```
  OSQLSynchQuery<ODocument> query = new OSQLSynchQuery<ODocument>(
     "SELECT FROM Profile WHERE name = ? AND surname = ?"
  );

  List<ODocument> result = database.command(query).execute("Barack", "Obama");
  ```

- **`:<parameter>`** Syntax is used in reference to named parameters. For instance,

  ```
  OSQLSynchQuery<ODcument> query = new OSQLSynchQuery<ODcument>(
     "SELECT FROM Profile WHERE name = :name AND surname = :surname"
  );

  Map<String,Object> params = new HashMap<String,Object>();
  params.put("name", "Barack");
  params.put("surname", "Obama");

  List<ODocument> result = database.command(query).execute(params);
  ```

> **NOTE**: With prepared queries, the parameter substitution feature only works with `SELECT` statements. It does not work with `SELECT` statements nested with other query types, such as `CREATE VERTEX`.

SQL Commands

In addition to queries, you can also execute SQL commands through the Java API. These require that you use the `.command()` method, passing it an `OCommandSQL` object. For instance,

```
int recordsUpdated = db.comamnd(
   new OCommandSQL("UPDATE Animal SET sold = false"
).execute();
```

When the command modifies the schema, such as `CREATE CLASS` or `ALTER PROPERTY`, remember that you also need to force a schema update on the database instance you're using.

```
db.getMetadata().getSchema().reload();
```

Traversing Documents

When using SQL, the process of combining two or more tables is handled through joins. Since OrientDB is a Graph Database, it can operate across documents by traversing links. This is much more efficient than the SQL join. For more information, see Java Traverse API.

In the example below, the application operates on a database with information on movies. It traverses each movie instance following links up to the fifth depth level:

```
for (OIdentifiable id : new OTraverse()
         .field("in").field("out")
         .target(database.browseClass("Movie").iterator())
         .predicate(new OCommandPredicate() {
   public boolean evaluate(ORecord<?> iRecord, OCommandContext iContext) {
      return ((Integer) iContext.getVariable("depth")) <= 5;
            }
         })
     ){
   System.out.println(id);
}
```

Schema

While you can use OrientDB in schema-less mode, there are times in which you need to enforce your own data model using a schema. OrientDB supports schema-full, schema-less and schema hybrid modes.

- **Schema Full** Enables the strict-mode at a class-level defining all fields as mandatory.
- **Schema Less**: Enables the creation of classes with no properties. This non-strict mode is the default in OrientDB and allows records to have arbitrary fields.
- **Schema Hybrid**: Enalbes a mix of classes that are schema-full and schema-less.

> Bear in mind, changes made to the schema are not transactional. You must execute these operations outside of a transaction.

To access the schema through the Console, use `SELECT` .

In order to access and work with the schema through the Java API, you need to get the `OMetadata` object from the database, then call the `getSchema()` method. For instance,

```
OSchema schema = database.getMetadata().getSchema();
```

This sets the object `schema` , which you can then use to further define classes and properties:

- **Document Database Classes**
- **Document Database Properties**

> For more information on how to access the schema through SQL or the Console, see Querying the Schema.

Classes in the Document Database

The Class is a concept taken from the Object-Oriented Programming paradigm. In OrientDB, classes define types of records. Conceptually, it's closest to the table in Relational Databases. You can make classes schema-less, schema-full or schema-hybrid.

Classes can inherit from other classes. Inheritance means that the sub-class extends the parent class, inheriting all its attributes as if they were its own.

Each class has its own clusters. These are either logical or physical. By default, they are logical. A class must have at least one defined cluster as its default cluster, but can support multiple clusters. OrientDB writes new records in the default cluster, but reads always involve all defined clusters.

When you create a new class, OrientDB creates a new physical cluster with it, which has the same name, but set in lowercase.

Creating Persistent Classes

Each class contains one or more properties, (which are also called fields). This mode is similar to the classical model of Relational Databases, where you define tables before storing records.

For instance, consider the use case of creating an `Account` class through the Java API. By default, new Physical Clusters are created to store the class instances.

```
OClass account = database.getMetadata().getSchema()
   .createClass("Account");
```

To create a new vertex or edge type, you need to extend the `V` or `E` classes, respectively. For example,

```
OClass person = database.getMetadata().getSchema()
   .createClass("Account", database.getMetadata()
      .getSchema().getClass("V"));
```

For more information, see Graph Schema.

Retrieving Persistent Classes

To retrieve persistent classes, use the `.getClass(String)` method. If the class does not exist, the method returns a null value.

```
OClass account = database.getMetadata().getSchema()
   .getClass("Account");
```

Dropping Persistent Classes

To drop a persistent class use the `OSchema.dropClass(String)` method. For instance,

```
database.getMetadata(0.getSchema.dropClass("Account");
```

When you use this method, OrientDB does not remove records from the removed class unless you explicitly delete them before dropping the class. For instance,

```
database.command(new OCommandSQL("DELETE FROM Account")
   .execute()
database.getMetadate().getSchema.dropClass("Account");
```

Constraints

To use constraints in schema-full mode, set the strict mode at the class-level. You can do this by calling the `setStrictMode(true)` method. In this case, you must pre-define all properties of the record.

Properties in the Document Database

Properties are the fields in a class. Generally, you can consider properties as synonymous with fields.

Working with Properties

Much as you can define properties using SQL, you can also define them through the Document API.

Creating Properties

Once you have created a class, you can create properties on that class. For instance,

```
OClass account = database.getMetadata().getschema()
   .createClass("Account");
account.createProperty("id", OType.INTEGER);
account.createProperty("birthDate", OType.DATE);
```

Bear in mind, the property must belong to a Type.

Dropping Properties

To drop persistent class properties, use the `OClass.dropProperty(String)` method. For instance,

```
database.getMetadata().getSchema().getClass("Account)
   .dropProperty("name");
```

When you drop properties using this method, note that doing so does not result in your removing records unless you explicitly do so. In order to do so, you need to issue an SQL command through the application to run an `UPDATE...REMOVE` statement. For instance,

```
database.getMetadata().getSchema().getClass("Account")
   .dropProperty("name");
database.command(new OCommandSQL(
   "UPDATE Account REMOVE name")).execute();
```

Working with Relationships

OrientDB supports two types of relations:

- Referenced Relationships
- Embedded Relationships

Referenced Relationships

In a referenced relationship, OrientDB uses a direct link to the referenced records. This removes the need for the computationally costly `JOIN` operation common in Relational Databases. For instance,

```
            customer
 Record A  ----------->  Record B
CLASS=Invoice          CLASS=Customer
 RID=5:23                RID=10:2
```

Here, *Record A* contains a reference to *Record B* in the property `customer`, Note that both records are accessible through any other record. Each has a Record ID.

1-1 and n-1 Referenced Relationships

In the case of one-to-one and many-to-one Referenced Relationships, you can express them through the `LINK` type. For instance, using the Document API:

```
OClass customer = database.getMetadata().getSchema()
   .createClass("Customer");
customer.createProperty("name", OType.STRING);

OClass invoice = database.getMetadata().getSchema()
   .createClass("Invoice");
invoice.createProperty("id", OType.INTEGER);
invoice.createProperty("date", OType.DATE);
invoice.createProperty("customer", OType.LINK, customer);
```

Here, records of the class `Invoice` link to records of the class `Customer` using the field `customer`.

1-n and n-M Referenced Relationships

In one-to-many and many-to-many Referenced Relationships, you can express the relationship using collections of links, such as:

- **`LINKLIST`** Ordered list of links, which can accept duplicates.
- **`LINKSET`** Unordered set of links, which doesn't accept duplicates.
- **`LINKMAP`** Ordered map of links, with a *String* key, which also doesn't accept duplicates.

For instance, in the case of a one-to-many relationship between the classes `Order` and `OrderItem`:

```
OClass orderItem = db.getMetadata().getSchema
   .createClass("OrderItem");
orderItem.createProperty("id", OType.INTEGER);
orderItem.createProperty("animal", Otype.LINK, animal);

OClass order = db.getMetadata().getSchema()
   .createClass("Order");
order.createProperty("id", OType.INTEGER);
order.createProperty("date", OType.DATE);
order.createProperty("item", OType.LINKLIST, orderItem);

db.getMetadata().getSchema().save();
```

Embedded Relationships

In the case of Embedded Relationships, the relationships are defined inside the records that embed them. The relationship is stronger than the Referenced Relationship. The embedded record does not have its own Record ID, which means that you cannot reference it directly through other records. It's only accessible through the containing record. If you delete the container record, the embedded record is removed with it. For instance,

```
                 address
  Record A    <>-------->   Record B
CLASS=Account             CLASS=Address
  RID=5:23                 NO RECORD ID
```

Here, *Record A* contains the whole of *Record B* in the property `address`. You can reach *Record B* only by traversing the container record. For instance,

```
orientdb> SELECT FROM account WHERE address.city = 'Rome'
```

1-1 and N-1 Embedded Relationships

In one-to-one and many-to-one relationships, use the `EMBEDDED` type. For instance,

```
OClass address = database.getMetadata().getSchema()
   .createClass("Address");

OClass account = database.getMetadata().gtschema()
   .createClass("Account");
account.createProperty("id", OType.INTEGER);
account.createProperty("birthDate", OType.DATE);
account.createProperty("address", OType.EMBEDDED, address);
```

Here, records of the class `Account` embed records of the class `Address` .

1-n and n-M Embedded Relationships

In the cases of one-to-many and many-to-many relationships , use embedded link collections types:

- **`EMBEDDEDLIST`** Ordered list of embedded links to records.
- **`EMBEDDEDSET`** Unordered set of records, which doesn't accept duplicates.
- **`EMBEDDEDMAP`** Ordered map with records as the value and a String instance as the key, which doesn't accept duplicates.

For instance, consider a one-to-many relationship between `Order` and `OrderItem` :

```
OClass orderItem = db.getMetadata().getSchema()
   .createClass("OrderItem");
orderItem.createProperty("id", OType.INTEGER);
orderITem.createProperty("animal", OType.LINK, animal);

OClass order = db.getMetadata().getSchema()
   .createClass("Order");
order.createProperty("id", OType.INTEGER);
order.createProperty("date", OType.DATE);
order.createProperty("items", OType.EMBEDDEDLIST,
   orderItem);
```

Working with Constraints

OrientDB supports a number of constrains on each property:

- **Minimum Value**: `setMin()` Defines the smallest acceptable value for the property. Works with strings and in date ranges.
- **Maximum Value**: `setMax()` Defines the largest acceptable value for the property. Works with strings and in date ranges.
- **Mandatory**: `setMandatory()` Defines a required property.
- **Read Only**: `setReadonly()` Defines whether you can update the property after creating it.
- **Not Null**: `setNotNull()` Defines whether property can receive null values.
- **Unique**: `setUnique()` Defines whether property rejects duplicate values. Note: Using this constraint can speed up searches made against this property.
- **Regular Expression** `setRegexp()` Defines whether the property must satisfy the given regular expression.

For instance, consider the properties you might set on the `Profile` class for a social networking service.

```
profile.createProperty("nick", OType.STRING).setMin("3")
   .setMax("30").setMandatory(true).setNotNull(true);
profile.createIndex("nickIdx", OClass.INDEX_TYPE
   .UNIQUE, "nick");

profile.createProperty("name", OType.STRING).setMin("3")
   .setMax("30");
profile.createProperty("surname", OType.STRING)
   .setMin("3").setMax("30");
profile.createProperty("registeredOn" OType.DATE)
   .setMin("2010-01-01 00:00:00");
profile.createProperty("lastAccessOn", OType.DATE)
   .setMin("2010-01-01 00:00:00");
```

Indexes as Constraints

To ensure that a property value remains unique, use the `UNIQUE` index constraint:

```
profile.createIndex("EmployeeId", OClass.INDEX_TYPE
   .UNIQUE, "id");
```

To ensure that a group of properties remains unique, create a composite index from multiple fields:

```
profile.createIndex("compositeIdx", OClass.INDEX_TYPE
   .NOTUNIQUE, "name", "surname");
```

For more information on indexes, see Indexes.

Working with Fields

OrientDB has a powerful way to extract parts of a Document field. This applies to the Java API, SQL Where conditions, and SQL projections.

To extract parts you have to use the square brackets.

Extract punctual items

Single item

Example: tags is an EMBEDDEDSET of Strings containing the values ['Smart', 'Geek', 'Cool'].

The expression tags[0] will return 'Smart'.

Single items

Inside square brackets put the items separated by comma ",".

Following the tags example above, the expression tags[0,2] will return a list with [Smart, 'Cool'].

Range items

Inside square brackets put the lower and upper bounds of an item, separated by "-".

Following the tags example above, the expression tags[1-2] returns ['Geek', 'Cool'].

Usage in SQL query

Example:

```
SELECT * FROM profile WHERE phones['home'] LIKE '+39%'
```

Works the same with double quotes.

You can go in a chain (contacts is a map of map):

```
SELECT * FROM profile WHERE contacts[phones][home] LIKE '+39%'
```

With lists and arrays you can pick an item element from a range:

```
SELECT * FROM profile WHERE tags[0] = 'smart'
```

and single items:

```
SELECT * FROM profile WHERE tags[0,3,5] CONTAINSALL ['smart', 'new', 'crazy']
```

and a range of items:

```
SELECT * FROM profile WHERE tags[0-5] CONTAINSALL ['smart', 'new', 'crazy']
```

Condition

Inside the square brackets you can specify a condition. Any SQL condition is supported.

Example:

```
employees[label = 'Ferrari']

employees[age > 25 AND name = 'John']
```

Use in graphs

You can cross a graph using a projection. This an example of traversing all the retrieved nodes with name "Tom". "out" is outEdges and it's a collection. Previously, a collection couldn't be traversed with the . notation. Example:

```
SELECT out.in FROM v WHERE name = 'Tom'
```

This retrieves all the vertices connected to the outgoing edges from the Vertex with name = 'Tom'.

A collection can be filtered with the equals operator. This an example of traversing all the retrieved nodes with name "Tom". The traversal crosses the out edges but only where the linked (in) Vertex has the label "Ferrari" and then forward to the:

```
SELECT out[in.label = 'Ferrari'] FROM v WHERE name = 'Tom'
```

Or selecting vertex nodes based on class:

```
SELECT out[in.@class = 'Car'] FROM v WHERE name = 'Tom'
```

Or both:

```
SELECT out[label='drives'][in.@class = 'Car'] FROM v WHERE name = 'Tom'
```

As you can see where brackets ([]) follow brackets, the result set is filtered in each step like a Pipeline.

NOTE: This doesn't replace the support of GREMLIN. GREMLIN is much more powerful because it does thousands of things more, but it's a simple and, at the same time, powerful tool to traverse relationships.

Document Database Comparison

This is a comparison page between OrientDB and other DocumentDB projects . To know more about the comparison of OrientDB against GraphDBs look at this comparison.

> NOTE: If any information about any product is outdated or wrong, please send an email to the committers with the link of the source of the right information. Thanks!

Features matrix

Feature	OrientDB	MongoDB	CouchDB
Web Site	http://www.orientdb.org	http://www.mongodb.org	http://www.couchdb.org
Supported models	Document and Graph	Document	Document
Transactions	Yes, ACID	No	Yes, ACID
Query languages	Extended SQL, Gremlin	Mongo Query Language	Non supported, JS API

Object API

ⓘ	**Object API allows to work with POJOs that bind OrientDB documents. This API is not able work on top of Graph-API. If you are interested on using a Object-Graph mapping framework, look at the available ones that work on top of Graph-API layer: Object-Graph Mapping.**

Requirements

To use the Object APi include the following jars in your classpath:

```
orientdb-core-*.jar
orientdb-object-*.jar
```

If you're using the Object Database interface connected to a remote server (not local/embedded mode) include also:

```
orientdb-client-*.jar
orientdb-enterprise-*.jar
```

Introduction

The OrientDB **Object Interface** works on top of the Document-Database and works like an Object Database: manages Java objects directly. It uses the Java Reflection to register the classes and Javassist tool to manage the Object-to-Document conversion. Please consider that the Java Reflection in modern Java Virtual Machines is really fast and the discovering of Java meta data is made only at first time.

Future implementation could use also the byte-code enhancement techniques in addition.

The proxied objects have a ODocument bounded to them and transparently replicate object modifications. It also allows lazy loading of the fields: they won't be loaded from the document until the first access. To do so the object MUST implement getters and setters since the Javassist Proxy is bounded to them. In case of object load, edit an update all non loaded fields won't be lost.

The database instance has an API to generate new objects already proxied, in case a non-proxied instance is passed it will be serialized, wrapped around a proxied instance and returned.

Read more about the Binding between Java Objects and Records.

Quick example of usage:

```
// OPEN THE DATABASE
OObjectDatabaseTx db = new OObjectDatabaseTx ("remote:localhost/petshop").open("admin", "admin");

// REGISTER THE CLASS ONLY ONCE AFTER THE DB IS OPEN/CREATED
db.getEntityManager().registerEntityClasses("foo.domain");

// CREATE A NEW PROXIED OBJECT AND FILL IT
Account account = db.newInstance(Account.class);
account.setName( "Luke" );
account.setSurname( "Skywalker" );

City rome =  db.newInstance(City.class,"Rome",  db.newInstance(Country.class,"Italy"));
account.getAddresses().add(new Address("Residence", rome, "Piazza Navona, 1"));

db.save( account );

// CREATE A NEW OBJECT AND FILL IT
Account account = new Account();
account.setName( "Luke" );
account.setSurname( "Skywalker" );

City rome = new City("Rome", new Country("Italy"));
account.getAddresses().add(new Address("Residence", rome, "Piazza Navona, 1"));

// SAVE THE ACCOUNT: THE DATABASE WILL SERIALIZE THE OBJECT AND GIVE THE PROXIED INSTANCE
account = db.save( account );
```

Connection Pool

One of most common use case is to reuse the database avoiding to create it every time. It's also the typical scenario of the Web applications.

```
// OPEN THE DATABASE
OObjectDatabaseTx db= OObjectDatabasePool.global().acquire("remote:localhost/petshop", "admin", "admin");

// REGISTER THE CLASS ONLY ONCE AFTER THE DB IS OPEN/CREATED
db.getEntityManager().registerEntityClass("org.petshop.domain");

try {
  ...
} finally {
  db.close();
}
```

The close() method doesn't close the database but release it to the owner pool. It could be reused in the future.

Database URL

In the example above a database of type Database Object Transactional has been created using the storage: remote:localhost/petshop. This address is a URL. To know more about database and storage types go to Database URL.

In this case the storage resides in the same computer of the client, but we're using the **remote** storage type. For this reason we need a OrientDB Server instance up and running. If we would open the database directly bypassing the server we had to use the **local** storage type such as "plocal:/usr/local/database/petshop/petshop" where, in this case, the storage was located in the /usr/local/database/petshop folder on the local file system.

Multi-threading

The OObjectDatabaseTx class is non thread-safe. For this reason use different OObjectDatabaseTx instances by multiple threads. They will share local cache once transactions are committed.

Inheritance

Starting from the release 0.9.19 OrientDB supports the Inheritance. Using the ObjectDatabase the inheritance of Documents fully matches the Java inheritance.

When registering a new class Orient will also generate the correct inheritance schema if not already generated.

Example:

```
public class Account {
  private String name;
// getters and setters
}

public class Company extends Account {
  private int employees;
// getters and setters
}
```

When you save a Company object, OrientDB will save the object as unique Document in the cluster specified for Company class. When you search between all the Account instances with:

```
SELECT FROM account
```

The search will find all the Account and Company documents that satisfy the query.

Use the database

Before to use a database you need to open or create it:

```
// CREATE AN IN MEMORY DATABASE
OObjectDatabaseTx db1 = new OObjectDatabaseTx("memory:petshop").create();

// OPEN A REMOTE DATABASE
OObjectDatabaseTx db2 = new OObjectDatabaseTx("remote:localhost/petshop").open("admin", "admin");
```

The database instance will share the connection versus the storage. if it's a local storage, then all the database instances will be synchronized on it. If it's a remote storage then the network connection will be shared among all the database instances.

To get the reference to the current user use:

```
OUser user = db.getUser();
```

Once finished remember to close the database to free precious resources.

```
db.close();
```

Working with POJO

Please read the POJO binding guide containing all the information about the management of POJO.

Work in schema-less mode

The Object Database can be used totally in schema-less mode as long as the POJO binding guide requirements are followed. Schema less means that the class must be created but even without properties. Take a look to this example:

```
OObjectDatabaseTx db = new OObjectDatabaseTx("remote:localhost/petshop").open("admin", "admin");
db.getEntityManager().registerEntityClass(Person.class);

Person p = db.newInstance(Person.class);
p.setName( "Luca" );
p.setSurname( "Garulli" );
p.setCity( new City( "Rome", "Italy" ) );

db.save( p );
db.close();
```

This is the very first example. While the code it's pretty clear and easy to understand please note that we didn't declared "Person" structure before now. However Orient has been able to recognize the new object and save it in persistent way.

Work in schema-full mode

In the schema-full mode you need to declare the classes you're using. Each class contains one or multiple properties. This mode is similar to the classic Relational DBMS approach where you need to create tables before storing records. To work in schema-full mode take a look at the Schema APIs page.

Create a new object

The best practice to create a Java object is to use the OObjectDatabaseTx.newInstance() API:

```
public class Person {
  private String name;
  private String surname;

  public Person(){
  }

  public Person(String name){
   this.name = name;
  }

  public Person(String name, String surname){
   this.name = name;
   this.surname = surname;
  }
// getters and setters
}

OObjectDatabaseTx db = new OObjectDatabaseTx("remote:localhost/petshop").open("admin", "admin");
db.getEntityManager().registerEntityClass(Person.class);

// CREATES A NEW PERSON FROM THE EMPTY CONSTRUCTOR
Person person = db.newInstance(Person.class);
person.setName( "Antoni" );
person.setSurname( "Gaudi" );
db.save( person );

// CREATES A NEW PERSON FROM A PARAMETRIZED CONSTRUCTOR
Person person = db.newInstance(Person.class,  "Antoni");
person.setSurname( "Gaudi" );
db.save( person );

// CREATES A NEW PERSON FROM A PARAMETRIZED CONSTRUCTOR
Person person = db.newInstance(Person.class,"Antoni","Gaudi");
db.save( person );
```

However any Java object can be saved by calling the db.save() method, if not created with the database API will be serialized and saved. In this case the user have to assign the result of the db.save() method in order to get the proxied instance, if not the database will always treat the object as a new one. Example:

```
// REGISTER THE CLASS ONLY ONCE AFTER THE DB IS OPEN/CREATED
db.getEntityManager().registerEntityClass(Animal.class);

Animal animal = new Animal();
animal.setName( "Gaudi" );
animal.setLocation( "Madrid" );
animal = db.save( animal );
```

Note that the behaviour depends by the transaction begun if any. See Transactions.

Browse all the records in a cluster

```
for (Object o : database.browseCluster("CityCars")) {
  System.out.println( ((Car) o).getModel() );
```

Browse all the records of a class

```
for (Animal animal : database.browseClass(Animal.class)) {
  System.out.println( animal.getName() );
```

Count records of a class

```
long cars = database.countClass("Car");
```

Count records of a cluster

```
long cityCars = database.countCluster("CityCar");
```

Update an object

Any proxied object can be updated using the Java language and then calling the db.save() method to synchronize the changes to the repository. Behaviour depends by the transaction begun if any. See Transactions.

```
animal.setLocation( "Nairobi" );
db.save( animal );
```

Orient will update only the fields really changed.

Example of how to update the price of all the animals by 5% more:

```
for (Animal animal : database.browseClass(Animal.class)) {
  animal.setPrice(animal.getPrice() * 105 / 100);
  database.save(animal);
}
```

If the db.save() method is called with a non-proxied object the database will create a new document, even if said object were already saved

Delete an object

To delete an object call the db.delete() method on a proxied object. If called on a non-proxied object the database won't do anything. Behaviour also depends by the transaction begun if any. See Transactions.

```
db.delete( animal );
```

Example of deletion of all the objects of class "Animal".

```
for (Animal animal : database.browseClass(Animal.class))
  database.delete(animal);
```

Cascade deleting

Object Database uses JPA annotations to manage cascade deleting. It can be done expliciting (orphanRemoval = true) or using the CascadeType. The first mode works only with @OneToOne and @OneToMany annotations, the CascadeType works also with @ManyToMany annotation.

Example:

```
public class JavaCascadeDeleteTestClass {
  ...

  @OneToOne(orphanRemoval = true)
  private JavaSimpleTestClass  simpleClass;

  @ManyToMany(cascade = { CascadeType.REMOVE })
  private Map<String, Child>   children    = new HashMap<String, Child>();

  @OneToMany(orphanRemoval = true)
  private List<Child>          list = new ArrayList<Child>();

  @OneToMany(orphanRemoval = true)
  private Set<Child> set = new HashSet<Child>();
  ...

  // GETTERS AND SETTERS
}
```

so calling

```
database.delete(testClass);
```

or

```
for (JavaCascadeDeleteTestClass testClass : database.browseClass(JavaCascadeDeleteTestClass.class))
  database.delete(testClass);
```

will also delete JavaSimpleTestClass instances contained in "simpleClass" field and all the other documents contained in "children","list" and "test"

Attaching and Detaching

Since version 1.1.0 the Object Database provides attach(Object) and detach(Object) methods to manually manage object to document data transfer.

Attach

With the attach method all data contained in the object will be copied in the associated document, overwriting all existing informations.

```
Animal animal = database.newInstance(Animal.class);
animal.name = "Gaudi" ;
animal.location = "Madrid";
database.attach(animal);
database.save(animal);
```

in this way all changes done within the object without using setters will be copied to the document.

There's also an attachAndSave(Object) methods that after attaching data saves the object.

```
Animal animal = database.newInstance(Animal.class);
animal.name = "Gaudi" ;
animal.location = "Madrid";
database.attachAndSave(animal);
```

This will do the same as the example before

Detach

With the detach method all data contained in the document will be copied in the associated object, overwriting all existing informations. The detach(Object) method returns a proxied object, if there's a need to get a non proxied detached instance the detach(Object,boolean) can be used.

```
Animal animal = database.load(rid);
database.detach(animal);
```

this will copy all the loaded document information in the object, without needing to call all getters. This methods returns a proxied instance

```
Animal animal = database.load(rid);
animal = database.detach(animal,true);
```

this example does the same as before but in this case the detach will return a non proxied instance.

Since version 1.2 there's also the detachAll(Object, boolean) method that detaches recursively the entire object tree. This may throw a StackOverflowError with big trees. To avoid it increase the stack size with -Xss java option. The boolean parameter works the same as with the detach() method.

```
Animal animal = database.load(rid);
animal = database.detachAll(animal,true);
```

Lazy detachAll

(Since 2.2)

When calling detachAll(object,true) on a large object tree, the call may become slow, especially when working with remote connections. It will recurse through every link in the tree and load all dependencies.

To only load parts of the object tree, you can add the @OneToOne(fetch=FetchType.LAZY) annotation like so:

```
public class LazyParent {

    @Id
    private String id;

    @OneToOne(fetch = FetchType.LAZY)
    private LazyChild child;
...
public class LazyChild {

    @Id
    private ORID id;

    private String name;

    public ORID getId() {
        return id;
    }

    public void setId(ORID id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

In the above example, when calling detachAll(lazyParent,true), the child variable (if a link is available) will contain a normal LazyChild object, but only with the id loaded. So the name property will be null, as will any other property that is added to the class. The id object can be used to load the LazyChild object in a later stage.

Execute a query

Although OrientDB is part of NoSQL databases, supports the SQL engine, or at least a subset of it with such extensions to work with objects and graphs.

To know more about the SQL syntax supported go to: SQL-Query.

Example:

```
List<Animal> result = db.query(
  new OSQLSynchQuery<Animal>("select * from Animal where ID = 10 and name like 'G%'"));
```

Right usage of the graph

OrientDB is a graph database. This means that traversing is very efficient. You can use this feature to optimize queries. A common technique is the Pivoting.

SQL Commands

To execute SQL commands use the `command()` method passing a OCommandSQL object:

```
int recordsUpdated = db.command(
  new OCommandSQL("UPDATE Animal SET sold = false")).execute();
```

See all the SQL Commands.

Get the ODocument from a POJO

The OObjectDatabaseTx implementation has APIs to get a document from its referencing object:

```
ODocument doc = db.getRecordByUserObject( animal );
```

In case of non-proxied objects the document will be a new generated one with all object field serialized in it.

Get the POJO from a Record

The Object Database can also create an Object from a record.

```
Object pojo = db.getUserObjectByRecord(record);
```

Schema Generation

Since version 1.5 the Object Database manages **automatic Schema generation** based on registered entities. This operation can be

- manual
- automatic

The ObjectDatabase will generate class properties based on fields declaration if not created yet.

Changes in class fields (as for type changing or renaming) types won't be updated, this operation has to be done manually

Manual Schema Generation

Schema can be generated manually for single classes or entire packages:

Version 1.6

```
db.getMetadata().getSchema().generateSchema(Foo.class); // Generates the schema for Foo class
db.getMetadata().getSchema().generateSchema("com.mycompany.myapp.mydomainpackage");  // Generates the schema for all classes c
ontained in the given package
```

Version 1.5

```
db.generateSchema(Foo.class); // Generates the schema for Foo class
db.generateSchema("com.mycompany.myapp.mydomainpackage"); // Generates the schema for all classes contained in the given packa
ge
```

Automatic Schema Generation

By setting the "automaticSchemaGeneration" property to true the schema will be generated automatically on every class declaration.

```
db.setAutomaticSchemaGeneration(true);
db.getEntityManager().registerClass(Foo.class); // Generates the schema for Foo class after registering.
db.getEntityManager().registerEntityClasses("com.mycompany.myapp.mydomainpackage"); // Generates the schema for all classes co
ntained in the given package after registering.
```

class Foo could look like, generating one field with an Integer and ignoring the String field.

```
public class Foo {
  private transient String field1; // ignore this field
  private Integer field2; // create a Integer
}
```

Standard schema management equivalent

Having the Foo class defined as following

```
public class Foo{
private String text;
private Child reference;
private int number;
//getters and setters
}
```

schema generation will create "text", "reference" and "number" properties as respectively STRING, LINK and INTEGER types.

The default schema management API equivalent would be

```
OClass foo = db.getMetadata().getSchema().getClass(Foo.class);
OClass child = db.getMetadata().getSchema().getClass(Child.class)
foo.createProperty("text",OType.STRING);
foo.createProperty("number",OType.INTEGER);
foo.createProperty("text",OType.LINK, child);
db.getMetadata().getSchema().save();
```

Schema synchronizing

Since version 1.6 there's an API to synchronize schema of all registered entities.

```
db.getMetadata().getSchema().synchronizeSchema();
```

By calling this API the ObjectDatabase will check all registered entities and generate the schema if not generated yet. This management is useful on multi-database enviroments

Old Implementation ODatabaseObjectTx

Until the release 1.0rc9 the Object Database was implemented as the class `com.orientechnologies.orient.db.object.ODatabaseObjectTx`. This class is deprecated, but if you want to continue to use it change the package to: `com.orientechnologies.orient.object.db`.

Introduction

This implementation and documentation refers to all ODatabaseObjectXXX deprecated classes.

The Orient Object DB works on top of the Document-Database and it's able to treat Java objects without the use of pre-processor, byte enhancer or Proxy classes. It uses the simpler way: the Java Reflection. Please consider that the Java reflection in modern Java Virtual Machines is really fast and the discovering of Java meta data is made at first time. Future implementation could use the byte-code enhancement techniques in addition.

Read more about the Binding between Java Objects and Records.

Quick example of usage:

```
// OPEN THE DATABASE
ODatabaseObjectTx db = new ODatabaseObjectTx ("remote:localhost/petshop").open("admin", "admin");

db.getEntityManager().registerEntityClasses("foo.domain");

// CREATE A NEW ACCOUNT OBJECT AND FILL IT
Account account = new Account()
account.setName( "Luke" );
account.setSurname( "Skywalker" );

City rome = new City("Rome", new Country("Italy"));
account.getAddresses().add(new Address("Residence", rome, "Piazza Navona, 1"));

db.save( account );
```

Connection Pool

One of most common use case is to reuse the database avoiding to create it every time. It's also the typical scenario of the Web applications.

```
// OPEN THE DATABASE
ODatabaseObjectTx db= ODatabaseObjectPool.global().acquire("remote:localhost/petshop", "admin", "admin");

...

db.close();
```

The close() method doesn't close the database but release it to the owner pool. It could be reused in the future.

Inheritance

Starting from the release 0.9.19 OrientDB supports the Inheritance. Using the ObjectDatabase the inheritance of Documents fully matches the Java inheritance.

Example:

```
public class Account {
  private String name;
}

public class Company extends Account {
  private int employees;
}
```

When you save a Company object, OrientDB will save the object as unique Document in the cluster specified for Company class. When you search between all the Account instances with:

```
SELECT FROM account
```

The search will find all the Account and Company documents that satisfy the query.

Object Binding

The ObjectDatabase implementation makes things easier for the Java developer since the binding between Objects to Records is transparent.

How it works?

OrientDB uses Java reflection and Javassist Proxy to bound POJOs to Records directly. Those proxied instances take care about the synchronization between the POJO and the underlying record. Every time you invoke a setter method against the POJO, the value is early bound into the record. Every time you call a getter method the value is retrieved from the record if the POJO's field value is null. Lazy loading works in this way too.

So the Object Database class works as wrapper of the underlying Document-Database.

NOTE: In case a non-proxied object is found it will be serialized, proxied and bounded to a corresponding Record.

Requirements

Declare persistent classes

Before to use persistent POJOs OrientDB needs to know which classes are persistent (between thousands in your classpath) by registering the persistent packages and/or classes. Example:

```
database.getEntityManager().registerEntityClasses("com.orientechnologies.orient.test.domain");
```

This must be done only right after the database is created or opened.

Naming conventions

OrientDB follows some naming conventions to avoid writing tons of configuration files but just applying the rule "Convention over Configuration". Below those used:

1. Java classes will be bound to persistent classes defined in the OrientDB schema with the same name. In OrientDB class names are case insensitive. The Java class name is taken without the full package. For example registering the class `Account` in the package `com.orientechnologies.demo`, the expected persistent class will be "Account" and not the entire `com.orientechnologies.demo.Account`. This means that class names, in the database, are always unique and can't exist two class with the same name even if declared in different packages.
2. Java class's attributes will be bound to the fields with the same name in the persistent classes. Field names are case sensitive.

Empty constructor

All the Java classes must have an empty constructor to let to OrientDB to create instances.

Getters and Setters

All your classes must have getters and setters of every field that needs to be persistent in order to let to OrientDB to manage proxy operations. Getters and Setters also need to be named same as the declaring field: Example:

```
public class Test {

  private String textField;
  private int intField;

  public String getTextField() {
    return textField;
  }

  public void setTextField( String iTextField ) {
    textField = iTextField;
  }

  // THIS DECLARATION WON'T WORK, ORIENTDB WON'T BE ABLE TO RECOGNIZE THE REAL FIELD NAME
  public int getInt(){
    return intField;
  }

  // THIS DECLARATION WON'T WORK, ORIENTDB WON'T BE ABLE TO RECOGNIZE THE REAL FIELD NAME
  public void setInt(int iInt){
    intField = iInt;
  }
}
```

Collections and Maps

To avoid ClassCastExecption when the Java classes have Collections and Maps, the interface must be used rather than the Java implementation. The classic mistake is to define in a persistent class the types ArrayList, HashSet, HashMap instead of List, Set and Map.

Example:

```
public class MyClass{
   // CORRECT
   protected List<MyElement> correctList;

   // WRONG: WILL THROW A ClassCastException
   protected ArrayList<MyElement> wrongList;

   // CORRECT
   protected Set<MyElement> correctSet;

   // WRONG: WILL THROW A ClassCastException
   protected TreeSet<MyElement> wrongSet;

   // CORRECT
   protected Map<String,MyElement> correctMap;

   // WRONG: WILL THROW A ClassCastException
   protected HashMap<String,MyElement> wrongMap;
}
```

POJO binding

OrientDB manages all the POJO attributes in persistent way during read/write from/to the record, except for the fields those:

- have the *transient* modifier
- have the *static* modifier,
- haven't getters and setters
- are set with anonymous class types.

OrientDB uses the Java reflection to discovery the POJO classes. This is made only once during the registration of the domain classes.

Default binding

This is the default. It tries to use the getter and setter methods for the field if they exist, otherwise goes in RAW mode (see below). The convention for the getter is the same as Java: `get<field-name>` where field-name is capitalized. The same is for setter but with 'set' as prefix instead of 'get': `set<field-name>`. If the getter or setter is missing, then the raw binding will be used.

Example: Field '`String name`' -> `getName()` and `setName(String)`

Custom binding

Since v1.2 Orient provides the possibility of custom binding extending the OObjectMethodFilter class and registering it to the wanted class.

- The custom implementation must provide the `public boolean isHandled(Method m)` to let Orient know what methods will be managed by the ProxyHandler and what methods won't.
- The custom implementation must provide the `public String getFieldName(Method m)` to let orient know how to parse a field name starting from the accessing method name. In the case those two methods are not provided the default binding will be used

The custom MethodFilter can be registered by calling `OObjectEntityEnhancer.getInstance().registerClassMethodFilter(Class<?>, customMethodFilter);`

Domain class example:

```
public class CustomMethodFilterTestClass {

  protected String standardField;

  protected String UPPERCASEFIELD;

  protected String transientNotDefinedField;

  // GETTERS AND SETTERS
  ...

}
```

Method filter example:

```
 public class CustomMethodFilter extends OObjectMethodFilter {
    @Override
    public boolean isHandled(Method m) {
      if (m.getName().contains("UPPERCASE")) {
        return true;
      } else if (m.getName().contains("Transient")) {
        return false;
      }
      return super.isHandled(m);
    }

    @Override
    public String getFieldName(Method m) {
      if (m.getName().startsWith("get")) {
        if (m.getName().contains("UPPERCASE")) {
          return "UPPERCASEFIELD";
        }
        return getFieldName(m.getName(), "get");
      } else if (m.getName().startsWith("set")) {
        if (m.getName().contains("UPPERCASE")) {
          return "UPPERCASEFIELD";
        }
        return getFieldName(m.getName(), "set");
      } else
        return getFieldName(m.getName(), "is");
    }
  }
```

Method filter registration example:

```
OObjectEntityEnhancer.getInstance().registerClassMethodFilter(CustomMethodFilterTestClass.class, new CustomMethodFilter());
```

Read a POJO

You can read a POJO from the database in two ways:

- by calling the method `load(ORID)`
- by executing a query `query(q)`

When OrientDB loads the record, it creates a new POJO by calling the empty constructor and filling all the fields available in the source record. If a field is present only in the record and not in the POJO class, then it will be ignored. Even when the POJO is updated, any fields in the record that are not available in the POJO class will be untouched.

Save a POJO

You can save a POJO to the database by calling the method `save(pojo)`. If the POJO is already a proxied instance, then the database will just save the record bounded to it. In case the object is not proxied the database will serialize it and save the corresponded record: **In this case the object MUST be reassinged with the one returned by the database**

Fetching strategies

Starting from release 0.9.20, OrientDB supports Fetching-Strategies by using the **Fetch Plans**. Fetch Plans are used to customize how OrientDB must load linked records. The ODatabaseObjectTx uses the Fetch Plan also to determine how to bind the linked records to the POJO by building an object tree.

Custom types

To let OrientDB use not supported types use the custom types. They MUST BE registered before domain classes registration, if not all custom type fields will be treated as domain classes. In case of registering a custom type that is already register as a domain class said class will be removed.

Important: java.lang classes cannot be managed this way

Example to manage an enumeration as custom type:

Enum declaration

```
public enum SecurityRole {
    ADMIN("administrador"), LOGIN("login");
    private String    id;

    private SecurityRole(String id) {
        this.id = id;
    }

    public String getId() {
        return id;
    }

    @Override
    public String toString() {
        return id;
    }

    public static SecurityRole getByName(String name) {
        if (ADMIN.name().equals(name)) {
            return ADMIN;
        } else if (LOGIN.name().equals(name)) {
            return LOGIN;
        }
        return null;
    }

    public static SecurityRole[] toArray() {
        return new SecurityRole[] { ADMIN, LOGIN };
    }
}
```

Custom type management

```
  OObjectSerializerContext serializerContext = new OObjectSerializerContext();
  serializerContext.bind(new OObjectSerializer<SecurityRole, String>() {

    public Object serializeFieldValue(Class<?> type, SecurityRole role) {
      return role.name();
    }

    public Object unserializeFieldValue(Class<?> type, String str) {
      return SecurityRole.getByName(str);
    }
  });

  OObjectSerializerHelper.bindSerializerContext(null, serializerContext);

// NOW YOU CAN REGISTER YOUR DOMAIN CLASSES
database.getEntityManager().registerEntityClass(User.class);
```

OrientDB will use that custom serializer to marshall and unmarshall special types.

ODatabaseObjectTx (old deprecated implementation)

Available since v1.0rc9

The ObjectDatabase implementation makes things easier for the Java developer since the binding between Objects to Records is transparent.

How it works?

OrientDB uses Java reflection and doesn't require that the POJO is enhanced in order to use it according to the JDO standard and doesn't use Proxies as do many JPA implementations such as Hibernate. So how can you work with plain POJOs?

OrientDB works in two ways:

- Connected mode

- Detached mode

Connected mode

The ODatabaseObjectTx implementation is the gateway between the developer and OrientDB. ODatabaseObjectTx keeps track of the relationship between the POJO and the Record.

Each POJO that's read from the database is created and tracked by ODatabaseObjectTx. If you change the POJO and call the `ODatabaseObjectTx.save(pojo)` method, OrientDB recognizes the POJO bound with the underlying record and, before saving it, will copy the POJO attributes to the loaded record.

This works with POJOs that belong to the same instance. For example:

```
ODatabaseObjectTx db = new ODatabaseObjectTx("remote:localhost/demo");
db.open("admin", "admin");

try{
  List<Customer> result = db.query( new OSQLSynchQuery<Customer>(db, "select from customer") );
  for( Customer c : result ){
    c.setAge( 100 );
    db.save( c ); // <- AT THIS POINT THE POJO WILL BE RECOGNIZED AS KNOWN BECAUSE IS
                 // ALWAYS LOADED WITH THIS DB INSTANCE
  }

} finally {
  db.close;
}
```

When the `db.save( c )` is called, the ODatabaseObjectTx instance already knows obout it because has been retrieved by using a query through the same instance.

Detached mode

In a typical Front-End application you need to load objects, display them to the user, capture the changes and save them back to the database. Usually this is implemented by using a database pool in order to avoid leaving a database instance open for the entire life cycle of the user session.

The database pool manages a configurable number of database instances. These instances are recycled for all database operations, so the list of connected POJOs is cleared at every release of the database pool instance. This is why the database instance doesn't know the POJO used by the application and in this mode if you save a previously loaded POJO it will appear as a NEW one and is therefore created as new instance in the database with a new RecordID.

This is why OrientDB needs to store the record information inside the POJO itself. This is retrieved when the POJO is saved so it is known if the POJO already has own identity (has been previously loaded) or not (it's new).

To save the Record Identity you can use the JPA **@Id** annotation above the property interested. You can declare it as:

- **Object**, the suggested, in this case OrientDB will store the ORecordId instance
- **String**, in this case OrientDB will store the string representation of the ORecordId
- **Long**, in this case OrientDB will store the right part of the RecordID. This works only if you've a schema for the class. The left side will be rebuilt at save time by getting the class id.

Example:

```
public class Customer{
  @Id
  private Object id; // DON'T CREATE GETTER/SETTER FOR IT TO PREVENT THE CHANGING BY THE USER APPLICATION,
                // UNLESS IT'S NEEDED

  private String name;
  private String surname;

  public String getName(){
    return name;
  }
  public void setName(String name){
    this.name = name;
  }

  public String getSurname(){
    return name;
  }
  public void setSurname(String surname){
    this.surname = surname;
  }
}
```

OrientDB will save the Record Identity in the **id** property even if getter/setter methods are not created.

If you work with transactions you also need to store the Record Version in the POJO to allow MVCC. Use the JPA **@Version** annotation above the property interested. You can declare it as:

- **java.lang.Object** (suggested) - a **com.orientechnologies.orient.core.version.OSimpleVersion** is used
- **java.lang.Long**
- **java.lang.String**

Example:

```
public class Customer{
  @Id
  private Object id; // DON'T CREATE GETTER/SETTER FOR IT TO PREVENT THE CHANGING BY THE USER APPLICATION,
                // UNLESS IT'S NEEDED

  @Version
  private Object version; // DON'T CREATE GETTER/SETTER FOR IT TO PREVENT THE CHANGING BY THE USER APPLICATION,
                     // UNLESS IT'S NEEDED

  private String name;
  private String surname;

  public String getName(){
    return name;
  }
  public void setName(String name){
    this.name = name;
  }

  public String getSurname(){
    return name;
  }
  public void setSurname(String surname){
    this.surname = surname;
  }
}
```

Save Mode

Since OrientDB doesn't know what object is changed in a tree of connected objects, by default it saves all the objects. This could be very expensive for big trees. This is the reason why you can control manually what is changed or not via a setting in the ODatabaseObjectTx instance:

```
db.setSaveOnlyDirty(true);
```

or by setting a global parameter (see Parameters):

```
OGlobalConfiguration.OBJECT_SAVE_ONLY_DIRTY.setValue(true);
```

To track what object is dirty use:

```
db.setDirty(pojo);
```

To unset the dirty status of an object use:

```
db.unsetDirty(pojo);
```

Dirty mode doesn't affect in memory state of POJOs, so if you change an object without marking it as dirty, OrientDB doesn't know that the object is changed. Furthermore if you load the same changed object using the same database instance, the modified object is returned.

Requirements

Declare persistent classes

In order to know which classes are persistent (between thousands in your classpath), you need to tell OrientDB. Using the Java API is:

```
database.getEntityManager().registerEntityClasses("com.orientechnologies.orient.test.domain");
```

OrientDB saves only the final part of the class name without the package. For example if you're using the class `Account` in the package `com.orientechnologies.demo`, the persistent class will be only "Account" and not the entire `com.orientechnologies.demo.Account`. This means that class names, in the database, are always unique and can't exist two class with the same name even if declared in different packages.

Empty constructor

All your classes must have an empty constructor to let to OrientDB to create instances.

POJO binding

All the POJO attributes will be read/stored from/into the record except for fields with the *transient* modifier. OrientDB uses Java reflection but the discovery of POJO classes is made only the first time at startup. Java Reflection information is inspected only the first time to speed up the access to the fields/methods.

There are 2 kinds of binding:

- Default binding and
- Raw binding

Default binding

This is the default. It tries to use the getter and setter methods for the field if they exist, otherwise goes in RAW mode (see below). The convention for the getter is the same as Java: `get<field-name>` where field-name is capitalized. The same is for setter but with 'set' as prefix instead of 'get': `set<field-name>`. If the getter or setter is missing, then the raw binding will be used.

Example: Field '`String name`' -> `getName()` and `setName(String)`

Raw binding

This mode acts at raw level by accessing the field directly. If the field signature is **private** or **protected**, then the accessibility will be forced. This works generally in all the scenarios except where a custom SecurityManager is defined that denies the change to the accessibility of the field.

To force this behaviour, use the JPA 2 **@AccessType** annotation above the relevant property. For example:

```
public class Customer{
  @AccessType(FIELD)
  private String name;

  private String surname;

  public String getSurname(){
    return name;
  }
  public void setSurname(String surname){
    this.surname = surname;
  }
}
```

Read a POJO

You can read a POJO from the database in two ways:

- by calling the method `load(ORID)`
- by executing a query `query(q)`

When OrientDB loads the record, it creates a new POJO by calling the empty constructor and filling all the fields available in the source record. If a field is present only in the record and not in the POJO class, then it will be ignored. Even when the POJO is updated, any fields in the record that are not available in the POJO class will be untouched.

Callbacks

You can define some methods in the POJO class that are called as callbacks before the record is read:

- @OBeforeDeserialization called just BEFORE unmarshalling the object from the source record
- @OAfterDeserialization called just AFTER unmarshalling the object from the source record

Example:

```
public class Account{
  private String name;
  transient private String status;

  @OAfterDeserialization
  public void init(){
    status = "Loaded";
  }
}
```

Callbacks are useful to initialize transient fields.

Save a POJO

You can save a POJO to the database by calling the method `save(pojo)`. If the POJO is already known to the ODatabaseObjectTx instance, then it updates the underlying record by copying all the POJO attributes to the records (omitting those with *transient* modifier).

Callbacks

You can define in the POJO class some methods called as callback before the record is written:

- @OBeforeSerialization called just BEFORE marshalling the object to the record

- @OAfterSerialization called just AFTER marshalling the object to the record

Example:

```
public class Account{
  private String name;
  transient private Socket s;

  @OAfterSerialization
  public void free(){
    s.close();
  }
}
```

Callbacks are useful to free transient resources.

== Fetching strategies =v

Starting from release 0.9.20, OrientDB supports Fetching-Strategies by using the **Fetch Plans**. Fetch Plans are used to customize how OrientDB must load linked records. The ODatabaseObjectTx uses the Fetch Plan also to determine how to bind the linked records to the POJO by building an object tree.

Custom types

To let OrientDB use not supported types use the custom types. Register them before to register domain classes. Example to manage a BigInteger (that it's not natively supported):

```
OObjectSerializerContext serializerContext = new OObjectSerializerContext();
serializerContext.bind(new OObjectSerializer<BigInteger, Integer>() {

  public Integer serializeFieldValue(Class<?> itype,  BigInteger iFieldValue) {
    return iFieldValue.intValue();
  }

  public  BigInteger unserializeFieldValue(Class<?> itype,  Integer iFieldValue) {
    return new  BigInteger(iFieldValue);
  }

});
OObjectSerializerHelper.bindSerializerContext(null, serializerContext);

// NOW YOU CAN REGISTER YOUR DOMAIN CLASSES
database.getEntityManager().registerEntityClass(Customer.class);
```

OrientDB will use that custom serializer to marshall and unmarshall special types.

Traverse

OrientDB is a graph database. This means that the focal point is on relationships (links) and how they are managed. The standard SQL language is not enough to work with trees or graphs because it lacks the recursion concept. This is the reason why OrientDB provides a new command to traverse trees and graphs: TRAVERSE. Traversing is the operation that crosses relationships between records (documents, vertexes, nodes, etc). This operation is much much faster than executing a JOIN in a Relational database.

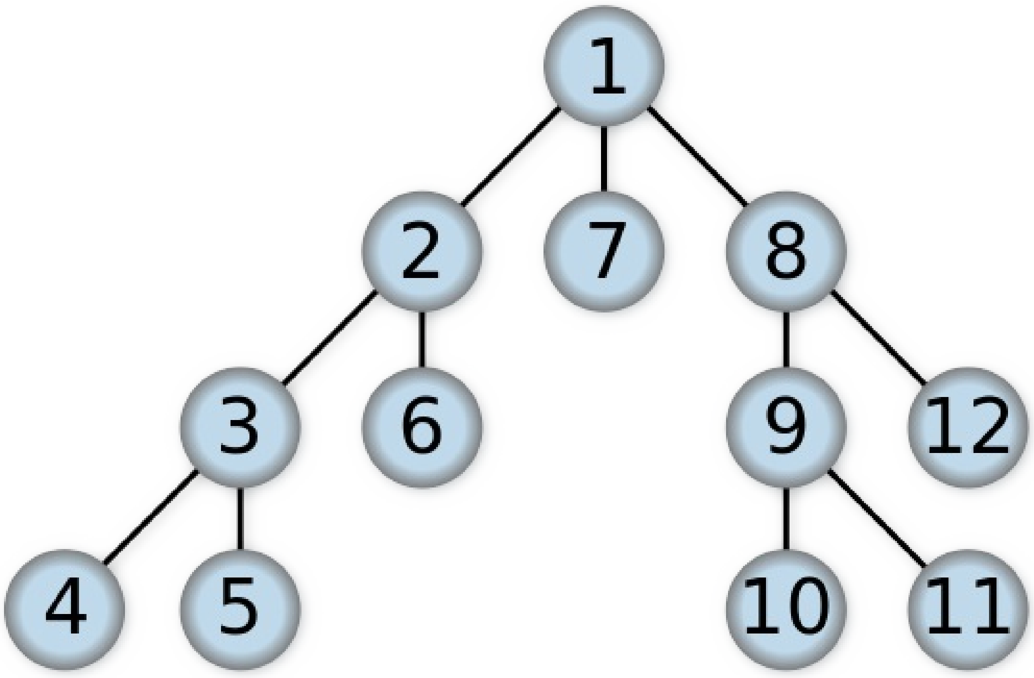
The main concepts of Traversal are:

- **target**, as the starting point where to traverse records. Can be:
  - **class**
  - **cluster**
  - **set of records**, specifying its Record ID
  - **sub-command** that returns an `Iterable<OIdentifiable>` . You can nest multiple select and traverse all together
- **fields**, the fields to traverse. Use `*` , `any()` or `all()` to traverse all fields in a document
- **limit**, the maximum number of records to retrieve
- **predicate**, as the predicate to execute against each traversed document. If the predicate returns true, the document is returned, otherwise it is skipped
- **strategy**, indicates how the graph traversed:
  - *DEPTH_FIRST*, the default,
  - *BREADTH_FIRST*,

Traversing strategies

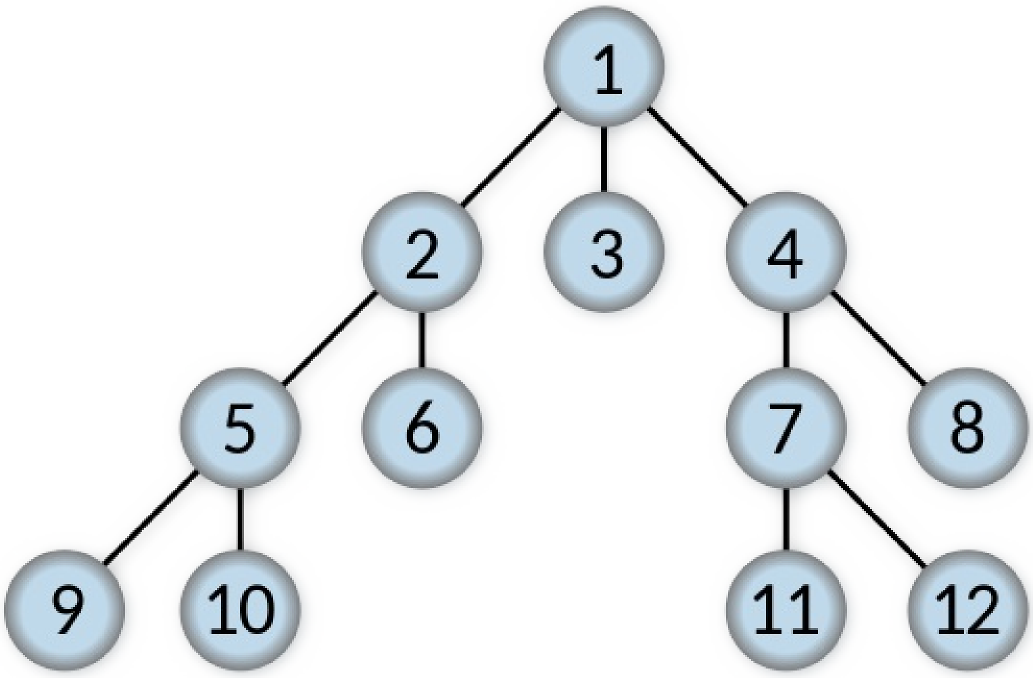
DEPTH_FIRST strategy

This is the default strategy used by OrientDB for traversal. It explores as far as possible along each branch before backtracking. It's implemented using recursion. To know more look at Depth-First algorithm. Below the ordered steps executed while traversing the graph using *DEPTH_FIRST* strategy:



BREADTH_FIRST strategy

It inspects all the neighboring nodes, then for each of those neighbor nodes in turn, it inspects their neighbor nodes which were unvisited, and so on. Compare **BREADTH_FIRST** with the equivalent, but more memory-efficient iterative deepening **DEPTH_FIRST** search and contrast with **DEPTH_FIRST** search. To know more look at Breadth-First algorithm. Below the ordered steps executed while traversing the graph using *BREADTH_FIRST* strategy:



Context variables

During traversal some context variables are managed and can be used by the traverse condition:

- **$depth**, as an integer that contain the depth level of nesting in traversal. First level is 0
- **$path**, as a string representation of the current position as the sum of traversed nodes
- **$stack**, as the stack current node traversed
- **$history**, as the entire collection of visited nodes

The following sections describe various traversal methods.

SQL Traverse

The simplest available way to execute a traversal is by using the SQL Traverse command. For instance, to retrieve all records connected **from** and **to Movie** records up to the **5th level of depth**:

```
for (OIdentifiable id : new OSQLSynchQuery<ODocument>("traverse in, out from Movie while $depth <= 5")) {
  System.out.println(id);
}
```

Look at the command syntax for more information.

Native Fluent API

Native API supports fluent execution guaranteeing compact and readable syntax. The main class is `OTraverse` :

- `target(<iter:Iterable<OIdentifiable>>)` , to specify the target as any iterable object like collections or arrays of OIdentifiable objects.
- `target(<iter:Iterator<OIdentifiable>>)` , to specify the target as any iterator object. To specify a class use `database.browseClass(<class-name>).iterator()`
- `target(<record:OIdentifiable>, <record:OIdentifiable>, ... )` , to specify the target as a var ars of OIterable objects

- `field(<field-name:string>)` , to specify the document's field to traverse. To add multiple field call this method in chain. Example: `.field("in").field("out")`
- `fields(<field-name:string>, <field-name:string>, ...)` , to specify multiple fields in one call passing a var args of Strings
- `fields(Collection<field-name:string>)` , to specify multiple fields in one call passing a collection of String
- `limit(<max:int>)` , as the maximum number of record returned
- `predicate(<predicate:OCommandPredicate>)` , to specify a predicate to execute against each traversed record. If the predicate returns true, then the record is returned as result, otherwise false. it's common to create an anonymous class specifying the predicate at the fly
- `predicate(<predicate:OSQLPredicate>)` , to specify the predicate using the SQL syntax.

In the traverse command context iContext you can read/put any variable. Traverse command updates these variables:

- **depth**, as the current depth of nesting
- **path**, as the string representation of the current path. You can also display it. Example: `select $path from (traverse * from V)`
- **stack**, as the List of operation in the stack. Use it to access to the history of the traversal. It's a `List<OTraverseAbstractProcess<?>>` where process implementations are:
  - `OTraverseRecordSetProcess` , usually the first one it's the base target of traverse
  - `OTraverseRecordProcess` , represent a traversed record
  - `OTraverseFieldProcess` , represent a traversal through a record's field
  - `OTraverseMultiValueProcess` , use on fields that are multivalue: arrays, collections and maps
- **history**, as the set of records traversed as a `Set<ORID>` .

Example using an anonymous OCommandPredicate as predicate

```
for (OIdentifiable id : new OTraverse()
              .field("in").field("out")
              .target( database.browseClass("Movie").iterator() )
              .predicate(new OCommandPredicate() {

    public Object evaluate(ORecord iRecord, ODocument iCurrentResult, OCommandContext iContext) {
      return ((Integer) iContext.getVariable("depth")) <= 5;
    }
  })) {

  System.out.println(id);
}
```

Example using the OSQLPredicate as predicate

```
for (OIdentifiable id : new OTraverse()
              .field("in").field("out")
              .target(database.browseClass("Movie").iterator())
              .predicate( new OSQLPredicate("$depth <= 5"))) {

  System.out.println(id);
}
```

Other examples

OTraverse gets any Iterable, Iterator and Single/Multi OIdentifiable. There's also the limit() clause. To specify multiple fields use fields(). Full example:

```
for (OIdentifiable id : new OTraverse()
              .target(new ORecordId("#6:0"), new ORecordId("#6:1"))
              .fields("out", "in")
              .limit(100)
              .predicate( new OSQLPredicate("$depth <= 10"))) {

  System.out.println( id);
}
```

Live Query

(Since 2.1)

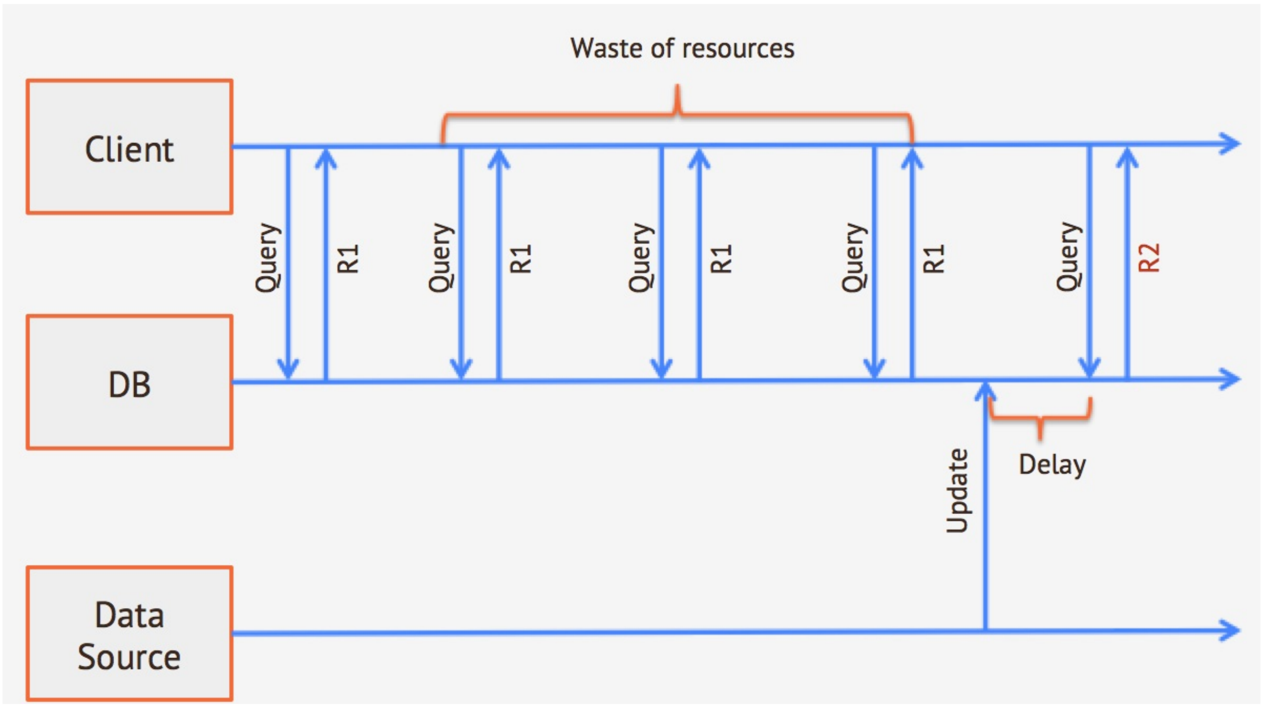
Writing **realtime, reactive applications** is hard task with traditional query paradigm. Think about a simple use case like updating a web page with fresh data coming from the database and keeping it updated over time; also consider that updates can be made by different data sources (multiple applications, manual DBA operations...).

With a traditional approach, the client has to poll the database to obtain fresh data. This approach has three fundamental problems:

- the client never knows whether something has changed in the DB, so it will execute polling queries even when nothing has changed. This can be a big waste of resources, especially when the query is expensive
- if you need (near) realtime data, the client will have to poll the database very often
- results arrive to the client at fixed time intervals, so if a change happens in the database at the middle of that time interval, the result will arrive to the client only at the next query

The image below summarizes this situation

traditional query polling approach

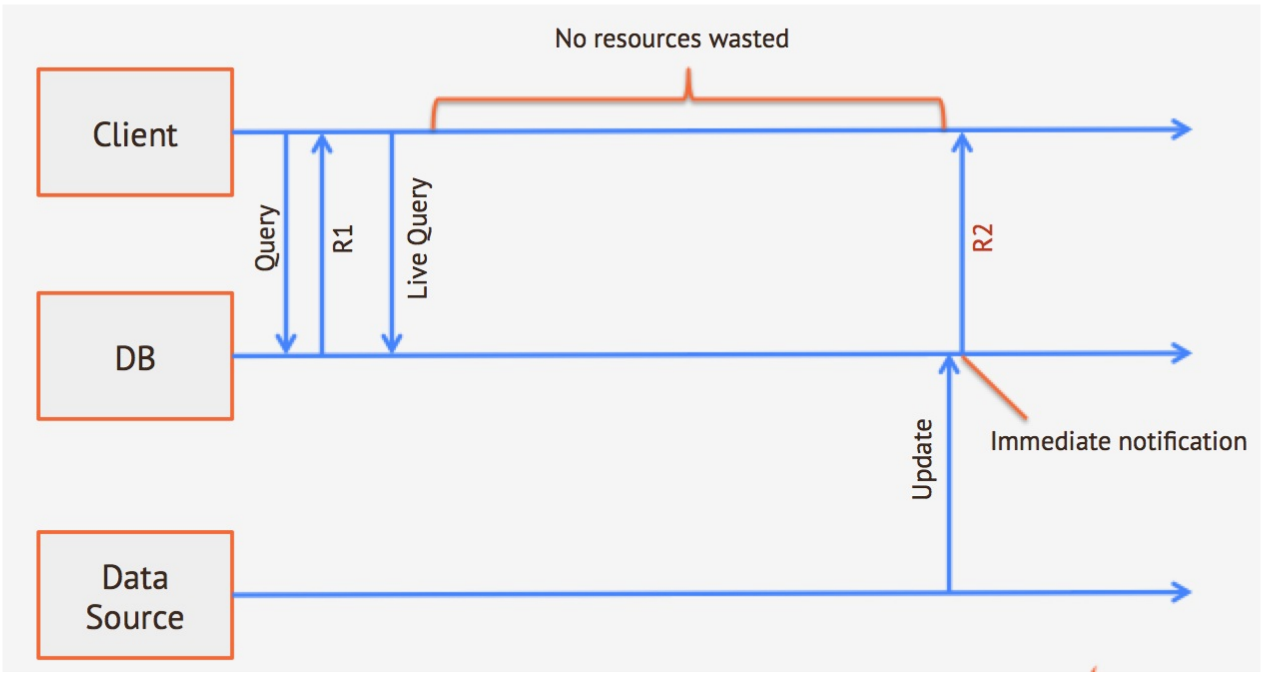


You have to make a choice here

- you can decide to have long polling intervals, reducing execution overhead, but having updated results later
- you can decine to have short polling intervals, having updated results sooner, but with a high execution overhead

With **LiveQuery** you can **subscribe** for changes on a particular class (or on a subset of records based on a WHERE condition); OrientDB will **push** changes to the client as soon as they happen in the database.

LiveQuery approach



Advantages are obvious:

- you do not have to poll the database, so there is no waste of resources when data do not change
- you get notifications as soon as changes happen in the db (no matter what the data source is)

Traditional queries vs. Live Query

When executing a SELECT statement (synchronous or asynchronous), you expect the system to return results that are currently present in the database and that match your selection criteria. You expect your result set to be finite and your query to execute in a given time.

A live query acts in a slightly different way:

- it **does not** return data as they are at the moment of the query execution
- it returns **changes** that happen to the database from that moment on and that match your criteria
- it never ends (unless you terminate it or an error occurs)
- it is asynchronous and **push** based: the server will send you data as soon as they are available, you just have to provide a callback.

To make the difference explicit, here is a simple example (just the flow of results in a meta-language, not a working example)

Standard query

A client executes a query on the DB

```
SELECT FROM PERSON
```

The client will receive a result that represents the current situation in the database:

```
RID,   NAME,    SURNAME
#12:0, "John",  "Smith"
#12:1, "Foo",   "Bar"

Number of results: 2
```

Another client inserts new data in the DB

```
INSERT INTO PERSON SET NAME = 'Jenny'
```

The first client will not receive this record, because the SELECT result set is closed. In short, this INSERT operation will not affect the previous query.

LIVE query

The client executes this query:

```
LIVE SELECT FROM PERSON
```

the immediate result of this query is just the unique identifier of the query itself (no data are returned, even if data are present in the DB)

```
token: 1234567 // Unique identifier of this live query, needed for unsubscribe
```

Another client inserts new data in the DB

```
INSERT INTO PERSON SET name = 'Jenny'
```

The first client will receive a message with the following content (schematic):

```
content: {@rid: #12:0, name: 'Jenny'}
operation: insert
```

Another client updates existing data

```
UPDATE PERSON SET NAME = 'Kerry' WHERE NAME = 'Jenny'
```

The first client will receive a message with the following content (schematic):

```
content: {@rid: #12:0, name: 'Kerry'}
operation: update
```

Now the first client can decide to unsubscribe from this LiveQuery

```
LIVE UNSUBSCRIBE 1234567
```

From now on, the live query will not return any other results to the client.

When should you use LiveQuery

LiveQuery is particularly useful in the following scenarios:

- when you need continuous (realtime) updates and you have multiple clients accessing different data subsets: polling is a an expensive operation, having thousands of clients that execute continuous polling could crash any server; in the best case it will be a waste of resources, especially if updates happen rarely
- when you have multiple data sources that insert/update data: if you have a single data source that populate the database, then you can intercept it and let it directly notify the clients for changes; unfortunately it almost never happens, in the majority of the use cases you will have multiple data sources, sometimes automatic (eg. applications) sometimes manual (your DBA that does data cleaning) and you want all these changes to be immediately notified to the client.
- when you develop on a push-based/reactive infrastructure: if you work on a message-driven infrastructoure or with a reactive framework, working with traditional (synchronous, blocking) queries can be a real pain; having a database that follows the same paradigm and that provides push notifications for data change will let you write applications in a more consistent way.

Supported interfaces

Live Query is currently supported from the following interfaces

- Java
- Node.js (OrientJS)

Enabling LiveQuery

Since version 2.2 the live query are enabled by default, to disable it set the property `query.live.support` to false.

LiveQuery in Java

To implement LiveQuery in Java you need two elements:

- a statement, to be executed by OLiveQuery
- a listener that asynchronous receives result

The listener has to implement OLiveResultListener. It just has a callback method that takes the live query token and the record that was modified (with the operation that occurred, eg. insert, update or delete)

```
class MyLiveQueryListener implements OLiveResultListener {

    public List<ORecordOperation> ops = new ArrayList<ORecordOperation>();

    @Override
    public void onLiveResult(int iLiveToken, ORecordOperation iOp) throws OException {
        System.out.println("New result from server for live query "+iLiveToken);
        System.out.println("operation: "+iOp.type);
        System.out.println("content: "+iOp.record);
    }

    public void onError(int iLiveToken) {
        System.out.println("Live query terminate due to error");
    }

    public void onUnsubscribe(int iLiveToken) {
        System.out.println("Live query terminate with unsubscribe");
    }

}
```

To actually execute the live query, you can use the `db.query()` method passing a `OLiveQuery` object as an argument, etc.

```
ODatabaseDocumentTx db = ... // I suppose you have an active DB instance

// Instantiate the query listener
MyLiveQueryListener listener = new MyLiveQueryListener();

// Execute the query
List<ODocument> result = db.query(new OLiveQuery<ODocument>("live select from Test", listener));

// Get the query token, it is needed for unsubscribe
String token = result.get(0).field("token"); // 1234567

// From now you will receive results from the server for every change that matches your query criteria.

// If you or someone else executes an INSERT on the server
db.command(new OCommandSQL("insert into test set name = 'foo', surname = 'bar'")).execute();

// Your MyLiveQueryListener.onLiveResult() will be invoked. In this case the result will be
// New result from server for live query 1234567 <- a token generated by the server
// operation: 3 <- ORecordOperation.CREATED
// content: {@Rid: "#12:0", name: "foo", surname: "bar"}

db.command(new OCommandSQL("update test set name = 'baz' where surname = 'bar'")).execute();

// New result from server for live query 1234567
// operation: 1 <- ORecordOperation.UPDATED
// content: {@Rid: "#12:0", name: "baz", surname: "bar"}

db.command(new OCommandSQL("live unsubscribe 1234567")).execute();

// From now you will not receive any other results
```

LiveQuery in Node.js

To use LiveQuery in Node.js you just have to import "orientjs" module with

`npm install orientjs`

Here is a simple example that shows how to use LiveQuery with OrientJS

```
var OrientDB = require('orientjs');
var server = OrientDB({host: 'localhost', port: 2424, useToken: true});
var db = server.use({name: 'test', username: 'admin', password: 'admin'});

db.liveQuery("live select from V")
  .on('live-insert', function(data){
     //new record inserted in the database,
     var myRecord = data.content;
     // your code here...
  })
  .on('live-delete', function(data){
      //record just deleted, receiving the old content
      var myRecord = data.content;
      // your code here...
  })
  .on('live-update', function(data){
      //record updated, receiving the new content
      var myRecord = data.content;
      // your code here...
  })
```

What's next

OrientDB team is working hard to make it stable and to support it on all the clients. To make live query stable in OrientDB 2.2, the following steps are needed:

- add tests for connection failure
- check for memory leaks
- add tests it in distributed mode
- give an additional check to the OrientJs implementation

We are also considering integrations with existing frameworks like (Meteor)

Starting from 2.2 Live Query will be released as Stable and will be covered by commercial support too.

OrientDB supports multi-threads access to the database. `ODatabase*` and `OrientGraph*` instances are not thread-safe, so you've to get *an instance per thread* and each database instance can be used *only in one thread per time*. The `ODocument` , `OrientVertex` and `OrientEdge` classes are non thread-safe too, so if you share them across threads you can have unexpected errors hard to recognize. For more information about how concurrency is managed by OrientDB look at Concurrency.

Since v2.1 OrientDB doesn't allow implicit usage of multiple database instances from the same thread. Any attempt to manage multiple instances in the same thread must explicitly call the method `db.activateOnCurrentThread()` against the database instance BEFORE you use it.

Multiple database instances point to the same storage by using the same URL. In this case Storage is thread-safe and orchestrates requests from different `ODatabase*` instances.

```
ODatabaseDocumentTx-1------+
                           +----> OStorage (url=plocal:/temp/db)
ODatabaseDocumentTx-2------+
```

The same as for Graph API:

```
OrientGraph-1------+
                   +----> OStorage (url=plocal:/temp/db)
OrientGraph-2------+
```

Database instances share the following objects:

- Schema
- Index Manager
- Security

These objects are synchronized for concurrent contexts by storing the current database in the ThreadLocal variable. Every time you create, open or acquire a database connection, the database instance is **automatically** set into the current ThreadLocal space, so in normal use this is hidden from the developer.

The current database is always reset for all common operations like load, save, etc.

Working with databases

The simplest way to work with multiple threads on the same database/graph is creating a new database/graph instance in the thread's `run()` method scope. In this way OrientDB will set the new database/graph automatically in the ThreadLocal for further usage.

Example:

```
OrientGraphFactory factory = new OrientGraphFactory("remote:localhost/mydb").setupPool(10, 20);

new Thread( new Runnable() {
  public void run(){
    OrientBaseGraph graph = factory.getTx();
    try{
      // OPERATION WITH THE GRAPH INSTANCE
      graph.addVertex("class:Account", "name", "Amiga Corporation");
    } finally {
      graph.shutdown();
    }
  }
} ).start();
```

If you are using a database/graph instance created in another thread, make sure you activate the instance before using it. The API is `graph.makeActive()` for the Graph API and `database.activateOnCurrentThread()` for the Document API.

Using multiple databases

When multiple database instances are used by the same thread, it's necessary to explicitely set the database/graph instance before to use it.

Example of using two database in the same thread:

```
ODocument rec1 = database1.newInstance();
ODocument rec2 = database2.newInstance();

rec1.field("name", "Luca");
database1.activateOnCurrentThread(); // MANDATORY SINCE 2.1
database1.save(rec1); // force saving in database1 no matter where the record came from

rec2.field("name", "Luke");
database2.activateOnCurrentThread(); // MANDATORY SINCE 2.1
database2.save(rec2); // force saving in database2 no matter where the record came from
```

In version 2.0.x, method `activateOnCurrentThread()` does not exist, you can use `setCurrentDatabaseInThreadLocal()` instead.

Get current database

To get the current database from the ThreadLocal use:

```
ODatabaseDocument database = (ODatabaseDocument) ODatabaseRecordThreadLocal.INSTANCE.get();
```

Manual control

Beware when you reuse database instances from different threads or then a thread handle multiple databases. In this case you can override the current database by calling this manually:

```
database.activateOnCurrentThread(); //v 2.1
// for OrientDB v. 2.0.x: database.setCurrentDatabaseInThreadLocal();
```

Where database is the current database instance. Example:

```

database1.activateOnCurrentThread();
ODocument rec1 = database1.newInstance();
rec1.field("name", "Luca");
rec1.save();

database2.activateOnCurrentThread();
ODocument rec2 = database2.newInstance();
rec2.field("name", "Luke");
rec2.save();
```

Custom database factory

Since v1.2 Orient provides an interface to manage custom database management in MultiThreading cases:

```
public interface ODatabaseThreadLocalFactory {
  public ODatabaseRecord getThreadDatabase();
}
```

Examples:

```
public class MyCustomRecordFactory implements ODatabaseThreadLocalFactory {

  public ODatabaseRecord getDb(){
   return ODatabaseDocumentPool.global().acquire(url, "admin", "admin");
  }
}


public class MyCustomObjectFactory implements ODatabaseThreadLocalFactory {
  public ODatabaseRecord getThreadDatabase(){
   return OObjectDatabasePool.global().acquire(url, "admin", "admin").getUnderlying().getUnderlying();
  }
}
```

Registering the factory:

```
ODatabaseThreadLocalFactory customFactory = new MyCustomRecordFactory();
 Orient.instance().registerThreadDatabaseFactory(customFactory);
```

When a database is not found in current thread it will be called the factory getDb() to retrieve the database instance.

Close a database

What happens if you are working with two databases and close just one? The Thread Local isn't a stack, so you loose the previous database in use. Example:

```
ODatabaseDocumentTx db1 = new ODatabaseDocumentTx("local:/temo/db1").create();
ODatabaseDocumentTx db2 = new ODatabaseDocumentTx("local:/temo/db2").create();
...

db2.close();

// NOW NO DATABASE IS SET IN THREAD LOCAL. TO WORK WITH DB1 SET IT IN THE THREAD LOCAL
db1.activateOnCurrentThread();
...
```

Multi Version Concurrency Control

If two threads update the same record, then the last one receive the following exception: "OConcurrentModificationException: Cannot update record #X:Y in storage 'Z' because the version is not the latest. Probably you are updating an old record or it has been modified by another user (db=vA your=vB)"

This is because every time you update a record, the version is incremented by 1. So the second update fails checking the current record version in database is higher than the version contained in the record to update.

This is an example of code to manage the concurrency properly:

Graph API

```
for( int retry = 0; retry < maxRetries; ++retry ) {
  try{
    // APPLY CHANGES
    vertex.setProperty( "name", "Luca" );
    vertex.addEdge( "Buy", product );

    break;
  } catch( ONeedRetryException e ) {
    // RELOAD IT TO GET LAST VERSION
    vertex.reload();
    product.reload();
  }
}
```

Document API

```
for( int retry = 0; retry < maxRetries; ++retry ) {
  try{
    // APPLY CHANGES
    document.field( "name", "Luca" );

    document.save();
    break;
  } catch( ONeedRetryException e ) {
    // RELOAD IT TO GET LAST VERSION
    document.reload();
  }
}
```

The same in transactions:

```
for( int retry = 0; retry < maxRetries; ++retry ) {
  db.begin();
  try{
    // CREATE A NEW ITEM
    ODocument invoiceItem = new ODocument("InvoiceItem");
    invoiceItem.field( "price", 213231 );
    invoiceItem.save();

    // ADD IT TO THE INVOICE
    Collection<ODocument> items = invoice.field( items );
    items.add( invoiceItem );
    invoice.save();

    db.commit();
    break;
  } catch( OTransactionException e ) {
    // RELOAD IT TO GET LAST VERSION
    invoice.reload();
  }
}
```

Where `maxRetries` is the maximum number of attempt of reloading.

What about running transaction?

Transactions are bound to a database, so if you change the current database while a tx is running, the deleted and saved objects remain attached to the original database transaction. When it commits, the objects are committed.

Example:

```
ODatabaseDocumentTx db1 = new ODatabaseDocumentTx("local:/temo/db1").create();

db1.begin();

ODocument doc1 = new ODocument("Customer");
doc1.field("name", "Luca");
doc1.save(); // NOW IT'S BOUND TO DB1'S TX

ODatabaseDocumentTx db2 = new ODatabaseDocumentTx("local:/temo/db2").create(); // THE CURRENT DB NOW IS DB2

ODocument doc2 = new ODocument("Provider");
doc2.field("name", "Chuck");
doc2.save(); // THIS IS BOUND TO DB2 BECAUSE IT'S THE CURRENT ONE

db1.activateOnCurrentThread();
db1.commit(); // WILL COMMIT DOC1 ONLY
```

Transaction Propagation

During application development there are situations when a transaction started in one method should be propagated to other method.

Lets suppose we have 2 methods.

```
public void method1() {
 database.begin();
 try {
  method2();
  database.commit();
 } catch(Exception e) {
   database.rollback();
 }
}

public void method2() {
  database.begin();
  try {
    database.commit();
  } catch(Exception e) {
    database.rollback();
  }
}
```

As you can see transaction is started in first method and then new one is started in second method. So how these transactions should interact with each other. Prior 1.7-rc2 first transaction was rolled back and second was started so were risk that all changes will be lost.

Since 1.7-rc2 we start nested transaction as part of outer transaction. What does it mean on practice?

Lets consider example above we may have two possible cases here:

First case:

1. begin outer transaction.
2. begin nested transaction.
3. commit nested transaction.
4. commit outer transaction.

When nested transaction is started all changes of outer transaction are visible in nested transaction and then when nested transaction is committed changes are done in nested transaction are not committed they will be committed at the moment when outer transaction will be committed.

Second case:

1. begin outer transaction.
2. begin nested transaction.
3. rollback nested transaction.
4. commit outer transaction.

When nested transaction is rolled back, changes are done in nested transaction are not rolled back. But when we commit outer transaction all changes will be rolled back and ORollbackException will be thrown.

So what instances of database should we use to get advantage of transaction propagation feature:

1. The same instance of database should be used between methods.
2. Database pool can be used, in such case all methods which asks for db connection in same thread will have the same the same database instance.

Binary Data

OrientDB natively handles binary data, namely BLOB. However, there are some considerations to take into account based on the type of binary data, the size, the kind of usage, etc.

Sometimes it's better to store binary records in a different path then default database directory to benefit of faster HD (like a SSD) or just to go in parallel if the OS and HW configuration allow this.

In this case create a new cluster in a different path:

```
db.addCluster("physical", "binary", "/mnt/ssd", "binary" );
```

All the records in cluster `binary` will reside in files created under the directory `/mnt/ssd`.

Techniques

Store on file system and save the path in the document

This is the simpler way to handle binary data: store them to the file system and just keep the path to retrieve them.

Example:

```
ODocument doc = new ODocument();
doc.field("binary", "/usr/local/orientdb/binary/test.pdf");
doc.save();
```

Pros:

- Easy to write
- 100% delegated to the File System

Cons:

- Binary data can't be automatically distributed using the OrientDB cluster

Store it as a Document field

ODocument class is able to manage binary data in form of `byte[]` (byte array). Example:

```
ODocument doc = new ODocument();
doc.field("binary", "Binary data".getBytes());
doc.save();
```

This is the easiest way to keep the binary data inside the database, but it's not really efficient on large BLOB because the binary content is serialized in Base64. This means a waste of space (33% more) and a run-time cost in marshalling/unmarshalling.

Also be aware that once the binary data reaches a certain size (10 MB in some recent testing), the database's performance can decrease significantly. If this occurs, the solution is to use the `ORecordBytes` solution described below.

Pros:

- Easy to write

Cons:

- Waste of space +33%
- Run-time cost of marshalling/unmarshalling
- Significant performance decrease once the binary reaches a certain large size

Store it with ORecordBytes

The `ORecordBytes` class is a record implementation able to store binary content without conversions (see above). This is the faster way to handle binary data with OrientDB but needs a separate record to handle it. This technique also offers the highest performance when storing and retrieving large binary data records.

Example:

```
ORecordBytes record = new ORecordBytes("Binary data".getBytes());
record.save();
```

Since this is a separate record, the best way to reference it is to link it to a Document record. Example:

```
ORecordBytes record = new ORecordBytes("Binary data".getBytes());

ODocument doc = new ODocument();
doc.field("id", 12345);
doc.field("binary", record);
doc.save();
```

In this way you can access to the binary data by traversing the `binary` field of the parent's document record.

```
ORecordBytes record = doc.field("binary");
byte[] content = record.toStream();
```

You can manipulate directly the buffer and save it back again by calling the `setDirty()` against the object:

```
byte[] content = record.toStream();
content[0] = 0;
record.setDirty();
record.save();
```

Or you can work against another `byte[]` :

```
byte[] content = record.toStream();
byte[] newContent = new byte[content*2];
System.arrayCopy(content, 0, newContent, 0, content.length);
record.fromStream(newContent);
record.setDirty();
record.save();
```

`ORecordBytes` class can work with Java Streams:

```
ORecordBytes record = new ORecordBytes().fromInputStream(in);
record.toOutputStream(out);
```

Pros:

- Fast and compact solution

Cons:

- Slightly complex management

While running in distributed mode ORecordBytes is not supported yet. See https://github.com/orientechnologies/orientdb/issues/3762 for more information.

Large content: split in multiple ORecordBytes

OrientDB can store up to 2Gb as record content. But there are other limitations on network buffers and file sizes you should tune to reach the 2GB barrier.

However managing big chunks of binary data means having big `byte[]` structures in RAM and this could cause a Out Of Memory of the JVM. Many users reported that splitting the binary data in chunks it's the best solution.

Continuing from the last example we could handle not a single reference against one `ORecordBytes` record but multiple references. A One-To-Many relationship. For this purpose the `LINKLIST` type fits perfect because maintains the order.

To avoid OrientDB caches in memory large records use the massive insert intent and keep in the collection the RID, not the entire records.

Example to store in OrientDB the file content:

```
database.declareIntent( new OIntentMassiveInsert() );

List<ORID> chunks = new ArrayList<ORID>();
InputStream in = new BufferedInputStream( new FileInputStream( file ) );
while ( in.available() > 0 ) {
  final ORecordBytes chunk = new ORecordBytes();

  // READ REMAINING DATA, BUT NOT MORE THAN 8K
  chunk.fromInputStream( in, 8192 );

  // SAVE THE CHUNK TO GET THE REFERENCE (IDENTITY) AND FREE FROM THE MEMORY
  database.save( chunk );

  // SAVE ITS REFERENCE INTO THE COLLECTION
  chunks.add( chunk.getIdentity() );
}

// SAVE THE COLLECTION OF REFERENCES IN A NEW DOCUMENT
ODocument record = new ODocument();
record.field( "data", chunks );
database.save( record );

database.declareIntent( null );
```

Example to read back the file content:

```
record.setLazyLoad(false);
for (OIdentifiable id : (List<OIdentifiable>) record.field("data")) {
    ORecordBytes chunk = (ORecordBytes) id.getRecord();
    chunk.toOutputStream(out);
    chunk.unload();
}
```

Pros:

- Fastest and compact solution

Cons:

- More complex management

While running in distributed mode ORecordBytes is not supported yet. See https://github.com/orientechnologies/orientdb/issues/3762 for more information.

Conclusion

What to use?

- Have you short binary data? Store them as document's field

- Do you want the maximum of performance and better use of the space? Store it with `ORecordBytes`
- Have you large binary objects? Store it with `ORecordBytes` but split the content in multiple records

Web Applications

The database instances are not thread-safe, so each thread needs a own instance. All the database instances will share the same connection to the storage for the same URL. For more information look at Java Multi threads and databases.

Java WebApp runs inside a Servlet container with a pool of threads that work the requests.

There are mainly 2 solutions:

- Manual control of the database instances from **Servlets** (or any other server-side technology like Apache Struts Actions, Spring MVC, etc.)
- Automatic control using **Servlet Filters**

Manual control

Graph API

```
package com.orientechnologies.test;
import javax.servlet.*;

public class Example extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException
  {
    OrientBaseGraph graph = new OrientGraph("plocal:/temp/db", "admin", "admin");

    try {
     // USER CODE

    } finally {
      graph.shutdown();
    }
  }
}
```

Document API

```
package com.orientechnologies.test;
import javax.servlet.*;

public class Example extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException
  {
    ODatabaseDocumentTx database = new ODatabaseDocumentTx("plocal:/temp/db").open("admin", "admin");

    try {
     // USER CODE

    } finally {
      database.close();
    }
  }
}
```

Automatic control using Servlet Filters

Servlets are the best way to automatise database control inside WebApps. The trick is to create a Filter that get a reference of the graph and binds it in the current ThreadLocal before to execute the Servlet code. Once returned the ThreadLocal is cleared and graph instance released.

JaveEE Servlets

Create a Filter class

Filter with Graph API

In this example a new graph instance is created per request, opened and at the end closed.

```
package com.orientechnologies.test;
import javax.servlet.*;

public class OrientDBFilter implements Filter {

  public void doFilter(ServletRequest request, ServletResponse response,
          FilterChain chain) {
      OrientBaseGraph graph = new OrientGraph("plocal:/temp/db", "admin", "admin");
      try{
        chain.doFilter(request, response);
      } finally {
        graph.shutdown();
      }
  }
}
```

Filter with Document API

In this example a new graph instance is created per request, opened and at the end closed.

```
package com.orientechnologies.test;
import javax.servlet.*;

public class OrientDBFilter implements Filter {

  public void doFilter(ServletRequest request, ServletResponse response,
          FilterChain chain) {
      ODatabaseDocumentTx database = new ODatabaseDocumentTx("plocal:/temp/db").open("admin", "admin");
      try{
        chain.doFilter(request, response);
      } finally {
        database.close();
      }
  }
}
```

Register the filter

Now we've create the filter class it needs to be registered in the **web.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
         http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
         version="2.4">
  <filter>
    <filter-name>OrientDB</filter-name>
    <filter-class>com.orientechnologies.test.OrientDBFilter</filter-class>
  </filter>
    <filter-mapping>
      <filter-name>OrientDB</filter-name>
      <url-pattern>/*</url-pattern>
    </filter-mapping>
    <session-config>
      <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```

JDBC Driver

The JDBC driver for OrientDB allows to connect to a remote server using the standard and consolidated way of interacting with database in the Java world.

Include in your projects

To be used inside your project, simply add the dependency to your pom:

```
<dependency>
  <groupId>com.orientechnologies</groupId>
  <artifactId>orientdb-jdbc</artifactId>
  <version>ORIENTDB_VERSION</version>
</dependency>
```

NOTE: to use SNAPSHOT version remember to add the Snapshot repository to your `pom.xml` .

How can be used in my code?

The driver is registered to the Java SQL DriverManager and can be used to work with all the OrientDB database types:

- memory,
- plocal and
- remote

The driver's class is `com.orientechnologies.orient.jdbc.OrientJdbcDriver` . Use your knowledge of JDBC API to work against OrientDB.

First get a connection

```
Properties info = new Properties();
info.put("user", "admin");
info.put("password", "admin");

Connection conn = (OrientJdbcConnection) DriverManager.getConnection("jdbc:orient:remote:localhost/test", info);
```

Then execute a Statement and get the ResultSet:

```
Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT stringKey, intKey, text, length, date FROM Item");

rs.next();

rs.getInt("@version");
rs.getString("@class");
rs.getString("@rid");

rs.getString("stringKey");
rs.getInt("intKey");

rs.close();
stmt.close();
```

The driver retrieves OrientDB metadata (@rid,@class and @version) only on direct queries. Take a look at tests code to see more detailed examples.

Advanced features

Connection pool

By default a new database instance is created every time you ask for a JDBC connection. OrientDB JDBC driver provides a Connection Pool out of the box. Set the connection pool parameters before to ask for a connection:

```
Properties info = new Properties();
info.put("user", "admin");
info.put("password", "admin");

info.put("db.usePool", "true"); // USE THE POOL
info.put("db.pool.min", "3");   // MINIMUM POOL SIZE
info.put("db.pool.max", "30");  // MAXIMUM POOL SIZE

Connection conn = (OrientJdbcConnection) DriverManager.getConnection("jdbc:orient:remote:localhost/test", info);
```

Spark compatibility (from 2.2.7)

Apache Spark allows reading and writing of DataFrames from JDBC data sources. The driver offers a compatibility mode to enable load of data frame from an OrientDb's class or query.

```
Map<String, String> options = new HashMap<String, String>() {{
    put("url", "jdbc:orient:remote:localhost/sparkTest");
    put("user", "admin");
    put("password", "admin");
    put("spark", "true"); // ENABLE Spark compatibility
    put("dbtable", "Item");
}};

SQLContext sqlCtx = new SQLContext(ctx);

DataFrame jdbcDF = sqlCtx.read().format("jdbc").options(options).load();
```

Custom drivers

OrientDB JDBC driver is compatible with most of the tools that support the JDBC standard. Even if we have tested the OrientDB JDBC driver against the most popular BI/Reporting tools, some tool could use a feature not supported. If you have problems with your tool and the OrientDB JDBC driver, please create an issue.

For some tool, instead, in order to use OrientDB JDBC driver, you need an additional connector:

QlickView: https://www.tiq-solutions.de/en/solutions/qlik-solutions/jdbc-connector/

JPA

There are two ways to configure OrientDB JPA

Configuration

The first - do it through /META-INF/persistence.xml Following OrientDB properties are supported as for now:

javax.persistence.jdbc.url, javax.persistence.jdbc.user, javax.persistence.jdbc.password, com.orientdb.entityClasses

You can also use *<class>* tag

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
    xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
    <persistence-unit name="appJpaUnit">
        <provider>com.orientechnologies.orient.object.jpa.OJPAPersistenceProvider</provider>

        <!-- JPA entities must be registered here -->
        <class>com.example.domain.MyPOJO</class>

        <properties>
            <property name="javax.persistence.jdbc.url" value="remote:localhost/test.odb" />
            <property name="javax.persistence.jdbc.user" value="admin" />
            <property name="javax.persistence.jdbc.password" value="admin" />
            <!-- Register whole package.
                             See com.orientechnologies.orient.core.entity.OEntityManager.registerEntityClasses(String) for mor
e details -->
            <property name="com.orientdb.entityClasses" value="com.example.domains" />
        </properties>
    </persistence-unit>
</persistence>
```

Programmatic

The second one is programmatic:

Guice example

```
com.google.inject.persist.jpa.JpaPersistModule.properties(Properties)
```

```
/**
 * triggered as soon as a web application is deployed, and before any requests
 * begin to arrive
 */
@WebListener
public class GuiceServletConfig extends GuiceServletContextListener {
    @Override
    protected Injector getInjector() {
        return Guice.createInjector(
                        new JpaPersistModule("appJpaUnit").properties(orientDBProp),
                        new ConfigFactoryModule(),
                        servletModule);
    }

    protected static final Properties orientDBProp = new Properties(){{
        setProperty("javax.persistence.jdbc.url", "remote:localhost/test.odb");
        setProperty("javax.persistence.jdbc.user", "admin");
        setProperty("javax.persistence.jdbc.password", "admin");
        setProperty("com.orientdb.entityClasses", "com.example.domains");
    }};

    protected static final ServletModule servletModule = new ServletModule() {
        @Override
        protected void configureServlets() {
            filter("/*").through(PersistFilter.class);
            // ...
    };
}
```

Native example

```
// OPEN THE DATABASE
OObjectDatabaseTx db = new OObjectDatabaseTx ("remote:localhost/petshop").open("admin", "admin");

// REGISTER THE CLASS ONLY ONCE AFTER THE DB IS OPEN/CREATED
db.getEntityManager().registerEntityClasses("foo.domain");
```

DB properties, that were passed programmatically, will overwrite parsed from XML ones

Note

Config parser checks persistence.xml with validation schemes (XSD), so configuration file must be valid.

1.0, 2.0 and 2.1 XSD schemes are supported.

JMX

Read Cache

JMX bean name: `com.orientechnologies.orient.core.storage.cache.local:type=O2QCacheMXBean`

It has following members:

- `usedMemory` , `usedMemoryInMB` , `usedMemoryInGB` which is amount of direct memory consumed by read cache in different units of measurements
- `cacheHits` is percent of cases when records will be downloaded not from disk but from read cache
- `clearCacheStatistics()` method may be called to clear cache hits statics so we always may start to gather cache hits statistic from any moment of time
- `amSize` , `a1OutSize` , `a1InSize` is the size of LRU queues are used in 2Q algorithm

Write Cache

JMX bean name: `com.orientechnologies.orient.core.storage.cache.local:type=OWOWCacheMXBean,name=<storage name>,id=<storage id>`

Write cache alike read cache is not JVM wide, it is storage wide, but one JVM may run several servers and each server may contain storage with the same name, that is why we need such complex name.

JMX bean of write cache has following members:

- `writeCacheSize` , `writeCacheSizeInMB` , `writeCacheSizeInGB` provides size of data in different units which should be flushed to disk in background thread
- `exclusiveWriteCacheSize` , `exclusiveWriteCacheSizeInMB` , `exclusiveWriteCacheSizeInGB` provides size of data which should be flushed to disk but contained only in write cache

More about memory model and data flow

At first when we read page we load it from disk and put it in read cache. Then we change page and put it back to read cache and write cache, but we do not copy page from read to write cache we merely send pointer to the same memory to write cache. Write cache flushes "dirty write page" in background thread. That is what property "writeCachSize" shows us amount of data in dirty pages which should be flushed. But there are very rare situations when page which is rarely used still is not flushed on disk and read cache has not enough memory to keep it. In such case this page is removed from read cache , but pointer to this page still exists in write cache, that is what property "exclusiveWriteCacheSize" shows us. Please note that this value is more than 0 only during extremely high load.

The rest properties of write cache JMX bean are following:

- `lastFuzzyCheckpointDate`
- `lastAmountOfFlushedPages`
- `durationOfLastFlush`

Gremlin API

Gremlin is a language specialized to work with Property Graphs. Gremlin is part of TinkerPop Open Source products. For more information:

- Gremlin Documentation
- Gremlin WiKi
- OrientDB adapter to use it inside Gremlin
- OrientDB implementation of TinkerPop Blueprints

To know more about Gremlin and TinkerPop's products subscribe to the Gremlin Group.

Get Started

Launch the **gremlin.sh** (or gremlin.bat on Windows OS) console script located in the **bin** directory:

```
> gremlin.bat

         \,,,/
         (o o)
-----oOOo-(_)-oOOo-----
```

Open the graph database

Before playing with Gremlin you need a valid OrientGraph instance that points to an OrientDB database. To know all the database types look at Storage types.

When you're working with a local or an in-memory database, if the database does not exist it's created for you automatically. Using the remote connection you need to create the database on the target server before using it. This is due to security restrictions.

Once created the **OrientGraph** instance with a proper URL is necessary to assign it to a variable. Gremlin is written in Groovy, so it supports all the Groovy syntax, and both can be mixed to create very powerful scripts!

Example with a local database (see below for more information about it):

```
gremlin> g = new OrientGraph("plocal:/home/gremlin/db/demo");
==>orientgraph[plocal:/home/gremlin/db/demo]
```

Some useful links:

- All Gremlin methods
- All available steps

Working with local database

This is the most often used mode. The console opens and locks the database for exclusive use. This doesn't require starting an OrientDB server.

```
gremlin> g = new OrientGraph("plocal:/home/gremlin/db/demo");
==>orientgraph[plocal:/home/gremlin/db/demo]
```

Working with a remote database

To open a database on a remote server be sure the server is up and running first. To start the server just launch **server.sh** (or server.bat on Windows OS) script. For more information look at OrientDB Server

```
gremlin> g = new OrientGraph("remote:localhost/demo");
==>orientgraph[remote:localhost/demo]
```

Working with in-memory database

In this mode the database is volatile and all the changes will be not persistent. Use this in a clustered configuration (the database life is assured by the cluster itself) or just for test.

```
gremlin> g = new OrientGraph("memory:demo");
==>orientgraph[memory:demo]
```

Use security

OrientDB supports security by creating multiple users and roles associated with certain privileges. To know more look at Security. To open the graph database with a different user than the default, pass the user and password as additional parameters:

```
gremlin> g = new OrientGraph("memory:demo", "reader", "reader");
==>orientgraph[memory:demo]
```

Create a new Vertex

To create a new vertex, use the **addVertex()** method. The vertex will be created and a unique id will be displayed as the return value.

```
g.addVertex();
==>v[#5:0]
```

Create an edge

To create a new edge between two vertices, use the **addEdge(v1, v2, label)** method. The edge will be created with the label specified.

In the example below two vertices are created and assigned to a variable (Gremlin is based on Groovy), then an edge is created between them.

```
gremlin> v1 = g.addVertex();
==>v[#5:0]

gremlin> v2 = g.addVertex();
==>v[#5:1]

gremlin> e = g.addEdge(v1, v2, 'friend');
==>e[#6:0][#5:0-friend->#5:1]
```

Save changes

OrientDB assigns a temporary identifier to each vertex and edge that is created. To save them to the database stopTransaction(SUCCESS) should be called

```
gremlin> g.stopTransaction(SUCCESS)
```

Retrieve a vertex

To retrieve a vertex by its ID, use the **v(id)** method passing the Record ID as an argument (with or without the prefix '#'). This example retrieves the first vertex created in the above example.

```
gremlin> g.v('5:0')
==>v[#5:0]
```

Get all the vertices

To retrieve all the vertices in the opened graph use **.V** (V in upper-case):

```
gremlin> g.V
==>v[#5:0]
==>v[#5:1]
```

Retrieve an edge

Retrieving an edge is very similar to retrieving a vertex. Use the *e(id)* method passing the Record ID as an argument (with or without the prefix '#'). This example retrieves the first edge created in the previous example.

```
gremlin> g.e('6:0')
==>e[#6:0][#5:0-friend->#5:1]
```

Get all the edges

To retrieve all the edges in the opened graph use **.E** (E in upper-case):

```
gremlin> g.E
==>e[#6:0][#5:0-friend->#5:1]
```

Traversal

The power of Gremlin is in traversal. Once you have a graph loaded in your database you can traverse it in many different ways.

Basic Traversal

To display all the outgoing edges of the first vertex just created append the **.outE** at the vertex. Example:

```
gremlin> v1.outE
==>e[#6:0][#5:0-friend->#5:1]
```

To display all the incoming edges of the second vertex created in the previous examples append the **.inE** at the vertex. Example:

```
gremlin> v2.inE
==>e[#6:0][#5:0-friend->#5:1]
```

In this case the edge is the same because it's the outgoing edge of 5:0 and the incoming edge of 5:1.

For more information look at the Basic Traversal with Gremlin.

Filter results

This example returns all the outgoing edges of all the vertices with label equal to 'friend'.

```
gremlin> g.V.outE('friend')
==>e[#6:0][#5:0-friend->#5:1]
```

Close the database

To close a graph use the **shutdown()** method:

```
gremlin> g.shutdown()
==>null
```

This is not strictly necessary because OrientDB always closes the database when the Gremlin console quits.

Create complex paths

Gremlin allows you to concatenate expressions to create more complex traversals in a single line:

```
v1.outE.inV
```

Of course this could be much more complex. Below is an example with the graph taken from the official documentation:

```
g = new OrientGraph('memory:test')

// calculate basic collaborative filtering for vertex 1
m = [:]
g.v(1).out('likes').in('likes').out('likes').groupCount(m)
m.sort{a,b -> a.value <=> b.value}

// calculate the primary eigenvector (eigenvector centrality) of a graph
m = [:]; c = 0;
g.V.out.groupCount(m).loop(2){c++ < 1000}
m.sort{a,b -> a.value <=> b.value}
```

Passing input parameters

Some Gremlin expressions require declaration of input parameters to be run. This is the case, for example, of bound variables, as described in JSR223 Gremlin Script Engine. OrientDB has enabled a mechanism to pass variables to a Gremlin pipeline declared in a command as described below:

```
Map<String, Object> params = new HashMap<String, Object>();
params.put("map1", new HashMap());
params.put("map2", new HashMap());
db.command(new OCommandSQL("select gremlin('
current.as('id').outE.label.groupCount(map1).optional('id').sideEffect{map2=it.map();map2+=map1;}
')")).execute(params);
```

GremlinPipeline

You can also use native Java GremlinPipeline like:

```
new GremlinPipeline(g.getVertex(1)).out("knows").property("name").filter(new PipeFunction<String,Boolean>() {
  public Boolean compute(String argument) {
    return argument.startsWith("j");
  }
}).back(2).out("created");
```

For more information: Using Gremlin through Java

Declaring output

In the simplest case, the output of the last step (https://github.com/tinkerpop/gremlin/wiki/Gremlin-Steps) in the Gremlin pipeline corresponds to the output of the overall Gremlin expression. However, it is possible to instruct the Gremlin engine to consider any of the input variables as output. This can be declared as:

```
Map<String, Object> params = new HashMap<String, Object>();
params.put("map1", new HashMap());
params.put("map2", new HashMap());
params.put("output", "map2");
db.command(new OCommandSQL("select gremlin('
current.as('id').outE.label.groupCount(map1).optional('id').sideEffect{map2=it.map();map2+=map1;}
')")).execute(params);
```

There are more possibilities to define the output in the Gremlin pipelines. So this mechanism is expected to be extended in the future. Please, contact OrientDB mailing list to discuss customized outputs.

Conclusions

Now you've learned how to use Gremlin on top of OrientDB. The best place to go in depth with this powerful language is the Gremlin WiKi.

Javascript

OrientDB supports server-side scripting. All the JVM languages are supported. By default JavaScript is installed.

Scripts can be executed on the client and on the server-side. On the client-side, the user must have READ privilege against the `database.command` resource. On the server-side, the scripting interpreter must be enabled. It is disabled by default for security reasons.

In order to return the result of a variable, put the variable name as last statement. Example:

```
var r = db.query('select from ouser');
for( var i = 0; i < r.length; ++ r ){
  print(r);
}
r
```

Will return the resultset.

See also

- SQL-batch

Usage

Via Java API

Executes a command like SQL but uses the class `OCommandScript` passing in the language to use. JavaScript is installed by default. Example:

```
db.command( new OCommandScript("Javascript", "print('hello world')") ).execute();
```

Via console

JavaScript code can be executed on the client-side, the console, or server-side:

- Use `js` to execute the script on the **client-side** running it in the console
- use `jss` to execute the script on the **server-side**. This feature is disabled by default. To enable it look at Enable Server side scripting.

Note: if you want to call a server-side Javascript function that was previously created in studio remember to add `commit()` after your insert or update operations. For example if you want to execute this function:

```
var g = orient.getGraph();
g.command('sql', 'insert into Person(name) values ("Luca")');
g.commit();
```

you can than use this command in console: `select functionName()`

Since the semi-colon `;` character is used in both console and JavaScript languages to separate statements, how can we execute multiple commands on the console and with JavaScript?

The OrientDB console uses a reserved keyword `end` to switch from JavaScript mode to console mode.

Example:

```
orientdb> connect remote:localhost/demo admin admin; js; for( i = 0; i < 10; i++ ){ db.query('select from MapPoint') };end; ex
it
```

This line connects to the remote server and executes 10 queries on the console. The `end` command switches the mode back to the OrientDB console and then executes the console `exit` command.

Below is an example to display the results of a query on the server and on the client.

1. connects to the remote server as `admin`
2. executes a query and assigns the result to the variable `r`, then displays it server-side and returns it to be displayed on the client side too
3. exits the console

Interactive mode

```
$ ./console.sh
OrientDB console v.1.5 www.orientechnologies.com
Type 'help' to display all the commands supported.

orientdb> connect remote:localhost/demo admin admin
Connecting to database [remote:localhost/demo] with user 'admin'...OK

orientdb> jss;var r = db.query('select from ouser');print(r);r

---+---------+--------------------+--------------------+--------------------+--------------------
  #| RID     |name                |password            |status              |roles
---+---------+--------------------+--------------------+--------------------+--------------------
  0|     #4:0|admin               |{SHA-256}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8A81F6F2AB448A918|ACTIVE
   |[1]
  1|     #4:1|reader              |{SHA-256}3D0941964AA3EBDCB00CCEF58B1BB399F9F898465E9886D5AEC7F31090A0FB30|ACTIVE
   |[1]
  2|     #4:2|writer              |{SHA-256}B93006774CBDD4B299389A03AC3D88C3A76B460D538795BC12718011A909FBA5|ACTIVE
   |[1]
---+---------+--------------------+--------------------+--------------------+--------------------
Script executed in 0,073000 sec(s). Returned 3 records

orientdb> exit
```

Batch mode

The same example above is executed in batch mode:

```
$ ./console.sh "connect remote:localhost/demo admin admin;jss;var r = db.query('select from ouser');print(r);r;exit"
OrientDB console v.1.0-SNAPSHOT (build 11761) www.orientechnologies.com
Type 'help' to display all the commands supported.
Connecting to database [remote:localhost/demo] with user 'admin'...OK

---+---------+--------------------+--------------------+--------------------+--------------------
  #| RID     |name                |password            |status              |roles
---+---------+--------------------+--------------------+--------------------+--------------------
  0|     #4:0|admin               |{SHA-256}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8A81F6F2AB448A918|ACTIVE
   |[1]
  1|     #4:1|reader              |{SHA-256}3D0941964AA3EBDCB00CCEF58B1BB399F9F898465E9886D5AEC7F31090A0FB30|ACTIVE
   |[1]
  2|     #4:2|writer              |{SHA-256}B93006774CBDD4B299389A03AC3D88C3A76B460D538795BC12718011A909FBA5|ACTIVE
   |[1]
---+---------+--------------------+--------------------+--------------------+--------------------
Script executed in 0,099000 sec(s). Returned 3 records
```

Examples of usage

Insert 1000 records

```
orientdb> js;for( i = 0; i < 1000; i++ ){ db.query( 'insert into jstest (label) values ("test'+i+'")' ); }
```

Create documents using wrapped Java API

```
orientdb> js new com.orientechnologies.orient.core.record.impl.ODocument('Profile').field('name', 'Luca').save()

Client side script executed in 0,426000 sec(s). Value returned is: Profile#11:52{name:Luca} v3
```

Enable Server-side scripting

For security reasons server-side scripting is disabled by default on the server. To enable it change the enable field to `true` in the **orientdb-server-config.xml** file under the Server Side Script Interpreter Plugin component and set "Javascript" between the supported languages:

```
<!-- SERVER SIDE SCRIPT INTERPRETER. WARNING! THIS CAN BE A SECURITY HOLE: ENABLE IT ONLY IF CLIENTS ARE TRUSTED, TO TURN ON S
ET THE 'ENABLED' PARAMETER TO 'true' -->
  <handler class="com.orientechnologies.orient.server.handler.OServerSideScriptInterpreter">
    <parameters>
      <parameter name="enabled" value="true" />
      <parameter name="allowedLanguages" value="SQL,Javascript" />
    </parameters>
  </handler>
```

For more information look at Server Side Script Interpreter Plugin.

NOTE: this will allow clients to execute any code inside the server. Enable it only if clients are trusted.

Javascript API

This API has been deprecated, please use HTTP calls from the browser by using the HTTP RESTful protocol.

This driver wraps the most common use cases in database usage. All parameters required by methods or constructor are Strings. This library works on top of HTTP RESTful protocol.

> **Note**: Due to cross-domain XMLHttpRequest restriction this API works, for now, only placed in the server deployment. To use it with cross-site look at Cross-site scripting.

The full source code is available here: **oriendb-api.js**.

See also

- Javascript-Command

Example

```
var database = new ODatabase('http://localhost:2480/demo');
databaseInfo = database.open();
queryResult = database.query('select from Address where city.country.name = \'Italy\'');
if (queryResult["result"].length == 0){
  commandResult = database.executeCommand('insert into Address (street,type) values (\'Via test 1\',\'Tipo test\')');
} else {
  commandResult = database.executeCommand('update Address set street = \'Via test 1\' where city.country.name = \'Italy\'');
}
database.close();
```

API

ODatabase object

ODatabase object requires server URL and database name:

Syntax: `new ODatabase(http://:/)`

Example:

```
var orientServer = new ODatabase('http://localhost:2480/demo');
```

Once created database instance is ready to be used. Every method return the operation result when it succeeded, null elsewhere.
In case of null result the database instance will have the error message obtainable by the getErrorMessage() method.

Open

Method that connects to the server, it returns database information in JSON format.

Browser Authentication

Syntax: `<databaseInstance>.open()`
Note: This implementation asks to the browser to provide user and password.

Example:

```
orientServer = new ODatabase('http://localhost:2480/demo');
databaseInfo = orientServer.open();
```

Javascript Authentication

Syntax: `<databaseInstance>.open(username,userpassword)`

Example:

```
orientServer = new ODatabase('http://localhost:2480/demo');
databaseInfo = orientServer.open('admin','admin');
```

Return Example:

```
{"classes": [
    {
      "id": 0,
      "name": "ORole",
      "clusters": [3],
      "defaultCluster": 3, "records": 3,
      "properties": [
        {
        "id": 0,
        "name": "mode",
        "type": "BYTE",
        "mandatory": false,
        "notNull": false,
        "min": null,
        "max": null,
        "indexed": false
      },
        {
        "id": 1,
        "name": "rules",
        "linkedType": "BYTE",
        "type": "EMBEDDEDMAP",
        "mandatory": false,
        "notNull": false,
        "min": null,
        "max": null,
        "indexed": false
      }
  ]},
],
"dataSegments": [
    {"id": -1, "name": "default", "size": 10485760, "filled": 1380391, "maxSize": "0", "files": "[${STORAGE_PATH}/default.0.od
a]"}
  ],

"clusters": [
    {"id": 0, "name": "internal", "type": "PHYSICAL", "records": 4, "size": 1048576, "filled": 60, "maxSize": "0", "files": "[
${STORAGE_PATH}/internal.0.ocl]"},
],

"txSegment": [
    {"totalLogs": 0, "size": 1000000, "filled": 0, "maxSize": "50mb", "file": "${STORAGE_PATH}/txlog.otx"}
  ], "users": [
    {"name": "admin", "roles": "[admin]"},
    {"name": "reader", "roles": "[reader]"},
    {"name": "writer", "roles": "[writer]"}
  ],

  "roles": [
    {"name": "admin", "mode": "ALLOW_ALL_BUT",
      "rules": []
    },
    {"name": "reader", "mode": "DENY_ALL_BUT",
      "rules": [{
        "name": "database", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.cluster.internal", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.cluster.orole", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.cluster.ouser", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.class.*", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.cluster.*", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.query", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.command", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.hook.record", "create": false, "read": true, "update": false, "delete": false
        }]
    },
],

  "config":{
    "values": [
      {"name": "dateFormat", "value": "yyyy-MM-dd"},
      {"name": "dateTimeFormat", "value": "yyyy-MM-dd hh:mm:ss"},
      {"name": "localeCountry", "value": ""},
      {"name": "localeLanguage", "value": "en"},
      {"name": "definitionVersion", "value": 0}
    ],
    "properties": [
    ]
  }
}
```

Query

Method that executes the query, it returns query results in JSON format.

Syntax: `<databaseInstance>.query(<queryText>, [limit], [fetchPlan])`

Limit and fetchPlan are optional.

Simple Example:

```
queryResult = orientServer.query('select from Address where city.country.name = \'Italy\'');
```

Return Example:

```
{ "result": [{
      "@rid": "12:0", "@class": "Address",
      "street": "Piazza Navona, 1",
      "type": "Residence",
      "city": "#13:0"
    }, {
      "@rid": "12:1", "@class": "Address",
      "street": "Piazza di Spagna, 111",
      "type": "Residence",
      "city": "#13:0"
    }
  ]
}
```

Fetched Example: fetching of all fields except "type"

```
queryResult = orientServer.query('select from Address where city.country.name = \'Italy\'',null,'*:-1 type:0');
```

Return Example 1:

```
{ "result": [{
      "@rid": "12:0", "@class": "Address",
      "street": "Piazza Navona, 1",
      "city":{
        "@rid": "13:0", "@class": "City",
        "name": "Rome",
        "country":{
          "@rid": "14:0", "@class": "Country",
          "name": "Italy"
        }
      }
    }, {
      "@rid": "12:1", "@version": 1, "@class": "Address",
      "street": "Piazza di Spagna, 111",
      "city":{
        "@rid": "13:0", "@class": "City",
        "name": "Rome",
        "country":{
          "@rid": "14:0", "@class": "Country",
          "name": "Italy"
        }
      }
    }
  ]
}
```

Fetched Example: fetching of all fields except "city" (Class)

```
queryResult = orientServer.query('select from Address where city.country.name = \'Italy\'',null,'*:-1 city:0');
```

Return Example 2:

```
{ "result": [{
       "@rid": "12:0", "@class": "Address",
       "street": "Piazza Navona, 1",
       "type": "Residence"
     }, {
       "@rid": "12:1", "@version": 1, "@class": "Address",
       "street": "Piazza di Spagna, 111",
       "type": "Residence"
    }
  ]
}
```

Fetched Example: fetching of all fields except "country" of City class

```
queryResult = orientServer.query('select from Address where city.country.name = \'Italy\'',null,'*:-1 City.country:0');
```

Return Example 3:

```
{ "result": [{
      "@rid": "12:0", "@class": "Address",
      "street": "Piazza Navona, 1",
      "type": "Residence",
      "city":{
          "@rid": "13:0", "@class": "City",
          "name": "Rome"
      }
    }
  ]
}
```

Execute Command

Method that executes arbitrary commands, it returns command result in text format.

Syntax: `<databaseInstance>.executeCommand(<commandText>)`

Example 1 (insert):

```
commandResult = orientServer.executeCommand('insert into Address (street,type) values (\'Via test 1\',\'Tipo test\')');
```

Return Example 1 (created record):

```
Address@14:177{street:Via test 1,type:Tipo test}
```

Example 2 (delete):

```
commandResult = orientServer.executeCommand('delete from Address where street = \'Via test 1\' and type = \'Tipo test\'');
```

Return Example 2 (records deleted):

```
{ "value" : 5 }
```

Note: Delete example works also with update command

Load

Method that loads a record from the record ID, it returns the record informations in JSON format.

Syntax: `.load(, [fetchPlan]);

Simple Example:

```
queryResult = orientServer.load('12:0');
```

Return Example:

```
{
"@rid": "12:0", "@class": "Address",
      "street": "Piazza Navona, 1",
      "type": "Residence",
      "city": "#13:0"
    }
```

Fetched Example: all fields fetched except "type"

```
queryResult = orientServer.load('12:0', '*:-1 type:0');
```

Return Example 1:

```
{
"@rid": "12:0", "@class": "Address",
      "street": "Piazza Navona, 1",
      "city":{
         "@rid": "13:0",
         "name": "Rome",
         "country":{
         "@rid": "14:0",
             "name": "Italy"
          }
      }
    }
```

Class Info

Method that retrieves information of a class, it returns the class informations in JSON format.

Syntax: `<databaseInstance>.classInfo(<className>)`

Example:

```
addressInfo = orientServer.classInfo('Address');
```

Return Example:

```
{ "result": [{
      "@rid": "14:0", "@class": "Address",
      "street": "WA 98073-9717",
      "type": "Headquarter",
      "city": "#12:1"
    }, {
      "@rid": "14:1", "@class": "Address",
      "street": "WA 98073-9717",
      "type": "Headquarter",
      "city": "#12:1"
    }
  ]
}
```

Browse Cluster

Method that retrieves information of a cluster, it returns the class informations in JSON format.

Syntax: `<databaseInstance>.browseCluster(<className>)`

Example:

```
addressInfo = orientServer.browseCluster('Address');
```

Return Example:

```
{ "result": [{
      "@rid": "14:0", "@class": "Address",
      "street": "WA 98073-9717",
      "type": "Headquarter",
      "city": "#12:1"
    }, {
      "@rid": "14:1", "@class": "Address",
      "street": "WA 98073-9717",
      "type": "Headquarter",
      "city": "#12:1"
    }
  ]
}
```

Server Information

Method that retrieves server informations, it returns the server informations in JSON format.
Note: Server information needs root username and password.

Syntax: `<databaseInstance>.serverInfo()`

Example:

```
serverInfo = orientServer.serverInfo();
```

Return Example:

```
{
  "connections": [{
    "id": "64",
    "id": "64",
    "remoteAddress": "127.0.0.1:51459",
    "db": "-",
    "user": "-",
    "protocol": "HTTP-DB",
    "totalRequests": "1",
    "commandInfo": "Server status",
    "commandDetail": "-",
    "lastCommandOn": "2010-12-23 12:53:38",
    "lastCommandInfo": "-",
    "lastCommandDetail": "-",
    "lastExecutionTime": "0",
    "totalWorkingTime": "0",
    "connectedOn": "2010-12-23 12:53:38"
    }],
  "dbs": [{
    "db": "demo",
    "user": "admin",
    "open": "open",
    "storage": "OStorageLocal"
    }],
  "storages": [{
    "name": "temp",
    "type": "OStorageMemory",
    "path": "",
    "activeUsers": "0"
    }, {
    "name": "demo",
    "type": "OStorageLocal",
    "path": "/home/molino/Projects/Orient/releases/0.9.25-SNAPSHOT/db/databases/demo",
    "activeUsers": "1"
    }],
    "properties": [
      {"name": "server.cache.staticResources", "value": "false"
      },
      {"name": "log.console.level", "value": "info"
      },
      {"name": "log.file.level", "value": "fine"
      }
    ]
}
```

Schema

Method that retrieves database Schema, it returns an array of classes (JSON parsed Object).

Syntax: `<databaseInstance>.schema()`

Example:

```
schemaInfo = orientServer.schema();
```

Return Example:

```
{"classes": [
    {
      "id": 0,
      "name": "ORole",
      "clusters": [3],
      "defaultCluster": 3, "records": 3,
      "properties": [
        {
        "id": 0,
        "name": "mode",
        "type": "BYTE",
        "mandatory": false,
        "notNull": false,
        "min": null,
        "max": null,
        "indexed": false
      },
        {
        "id": 1,
        "name": "rules",
        "linkedType": "BYTE",
        "type": "EMBEDDEDMAP",
        "mandatory": false,
        "notNull": false,
        "min": null,
        "max": null,
        "indexed": false
      }
  ]},
]
}
```

getClass()

Return a schema class from the schema.

Syntax: `<databaseInstance>.getClass(<className>)`

Example:

```
var customerClass = orientServer.getClass('Customer');
```

Return Example:

```
{
  "id": 0,
  "name": "Customer",
  "clusters": [3],
  "defaultCluster": 3, "records": 3,
  "properties": [
    {
      "id": 0,
      "name": "name",
      "type": "STRING",
    },
    {
      "id": 1,
      "name": "surname",
      "type": "STRING",
    }
  ]
}
```

Security

Roles

Method that retrieves database Security Roles, it returns an array of Roles (JSON parsed Object).

Syntax: `<databaseInstance>.securityRoles()`

Example:

```
roles = orientServer.securityRoles();
```

Return Example:

```
{ "roles": [
    {"name": "admin", "mode": "ALLOW_ALL_BUT",
      "rules": []
    },
    {"name": "reader", "mode": "DENY_ALL_BUT",
      "rules": [{
        "name": "database", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.cluster.internal", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.cluster.orole", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.cluster.ouser", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.class.*", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.cluster.*", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.query", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.command", "create": false, "read": true, "update": false, "delete": false
        }, {
        "name": "database.hook.record", "create": false, "read": true, "update": false, "delete": false
        }]
    }
  ]
}
```

Users

Method that retrieves database Security Users, it returns an array of Users (JSON parsed Object).

Syntax: `<databaseInstance>.securityUsers()`

Example:

```
users = orientServer.securityUsers();
```

Return Example:

```
{ "users": [
    {"name": "admin", "roles": "[admin]"},
    {"name": "reader", "roles": "[reader]"},
    {"name": "writer", "roles": "[writer]"}
  ]
}
```

close()

Method that disconnects from the server.

Syntax: `<databaseInstance>.close()`

Example:

```
orientServer.close();
```

Change server URL

Method that changes server URL in the database instance.
You'll need to call the open method to reconnect to the new server.

Syntax: `<databaseInstance>.setDatabaseUrl(<newDatabaseUrl>)`

Example:

```
orientServer.setDatabaseUrl('http://localhost:3040')
```

Change database name

Method that changes database name in the database instance.
You'll need to call the open method to reconnect to the new database.

Syntax: `<databaseInstance>.setDatabaseName(<newDatabaseName>)`

Example:

```
orientServer.setDatabaseName('demo2');
```

Setting return type

This API allows you to chose the return type, Javascript Object or JSON plain text. Default return is Javascript Object.

Important: the javascript object is not always the evaluation of JSON plain text: for each document (identified by its Record ID) the JSON file contains only one expanded object, all other references are just its Record ID as String, so the API will reconstruct the real structure by re-linking all references to the matching javascript object.

Syntax: `orientServer.setEvalResponse(<boolean>)`

Examples:

```
orientServer.setEvalResponse(true);
```

Return types will be Javascript Objects.

```
orientServer.setEvalResponse(false);
```

Return types will be JSON plain text.

Cross-site scripting

To invoke OrientDB cross-site you can use the `query` command in GET and the JSONP protocol. Example:

```
<script type="text/javascript" src='http://127.0.0.1:2480/query/database/sql/select+from+XXXX?jsoncallback=var datajson='></sc
ript>
```

This will put the result of the query `select from XXXX</code>` into the *<code>datajson</code>* variable.

Errors

In case of errors the error message will be stored inside the database instance, retrievable by getErrorMessage() method.

Syntax: `<databaseInstance>.getErrorMessage()`

Example:

```
if (orientServer.getErrorMessage() != null){
        //write error message
}
```

OrientJS Driver

OrientDB supports all JVM languages for server-side scripting. Using the OrientJS module, you can develop database applications for OrientDB using the Node.js language. It is fast, lightweight and uses the binary protocol, with features including:

- Intuitive API, based on the Bluebird promise library.
- Fast Binary Protocol Parser
- Distributed Support
- Multi-Database Access, through the same socket.
- Connection Pooling
- Migration Support

This page provides basic information on setting up OrientJS on your system. For more information on using OrientJS in developing applications, see

- Server API
- Database API
- Class API
- Index API
- Function API
- Queries
- Transactions
- Events

Installation

In order to use OrientJS, you need to install it on your system. There are two methods available to you in doing this: you can install it from the Node.js repositories or you can download the latest source code and build it on your local system.

Both methods require the Node.js package manager, NPM, and can install OrientJS as either a local or global package.

Installing from the Node.js Repositories

When you call NPM and provide it with the specific package name, it connects to the Node.js repositories and downloads the source package for OrientJS. It then installs the package on your system, either at a global level or in the `node_modules` folder of the current working directory.

- To install OrientJS locally in `node_modules`, run the following command:

  ```
  $ npm install orientjs
  ```

- To install OrientJS globally, run the following command as root or an administrator:

  ```
  # npm install orientjs -g
  ```

Once NPM finishes the install, you can begin to develop database applications using OrientJS.

Building from the Source Code

When building from source code, you need to download the driver directly from GitHub, then run NPM against the branch you want to use or test.

1. Using Git, clone the package repository, then enter the new directory:

```
$ git clone https://github.com/orientechnologies/orientjs.git
$ cd orientjs
```

2. When you clone the repository, Git automatically provides you with the current state of the `master` branch. If you would like to work with another branch, like `develop` or test features on past releases, you need to check out the branch you want. For instance,

```
$ git checkout develop
```

3. Once you've selected the branch you want to build, call NPM to handle the installation.

    - To install OrientJS locally, run the following command:

    ```
    $ npm install
    ```

    - To install OrientJS globally, instead run this command as root or an administrator:

    ```
    # npm install -g
    ```

4. Run the tests to make sure it works:

```
$ npm test
```

By calling NPM without an argument, it builds the package in the current working directory, here your local copy of the OrientJS GitHub repository.

Support

OrientJS aims to support version 2.0.0 and later. It has been tested on both versions 2.0.x and 2.1 of OrientDB. While it may work with earlier versions, it does not currently support them directly.

Bonsai Structure

Currently, OrientJS does not support the tree-based Bonsai Structure feature in OrientDB, as it relies on additional network requests.

What this means is that, by default, the result of `JSON.stringify(record)` on a record with 119 edges would behave very differently from a record with more than 120 edges. It can lead to unexpected results that may not appear at any point during development, but which will occur when your application runs in production

For more information, see Issue 2315. Until this issue is addressed it is **strongly recommended** that you set the `RID_EMBEDDED_TO_SBTREEBONSAI_THRESHOLD` to a very large figure, such as 2147483647.

Maximum Call Stack Size Exceeded

There is a maximum call stack size issue currently being worked on. This occurs in queries over large record-sets (as in, in the range of 150,000 records), results in a `RangeError` exception.

For more information, see Issue 116.

OrientJS Server API

With OrientJS installed on your system, you can now begin to develop applications around it. In your code, this starts with initializing the client connection between your application and the OrientDB Server, which is managed through the OrientJS Server API.

> By convention, the variable on which you initialize the Server API is called `server`. This designation is arbitrary and used only for convenience.

Initializing the Server API

In order to use the OrientDB with your Node.js application, you first need to initialize the Server API. This provides your application with the tools it needs in interacting with the OrientDB server to find and access the necessary database objects.

```
var OrientDB = require('orientjs');

var server = OrientDB({
   host:       'localhost',
   port:       2424,
   username:   'root',
   password:   'root_passwd'
});
```

Your application uses this code to initialize the `server` variable. It then connects to OrientDB using `root` and `root_passwd` as the login credentials.

When you're done with the server, you need to close it to free up system resources.

```
server.close();
```

Using Tokens

OrientJS supports tokens through the Network Binary Protocol. To use it in your application, define the `useToken` configuration parameter when you initialize the server connection.

For instance,

```
var server = OrientDB({
   host:     'localhost',
   port:     2424,
   username: 'root',
   password: 'root_passwd',
   useToken: true
});
```

Using Distributed Databases

Beginning in version 2.1.11 of OrientJS, you can develop Node.js applications to use distributed database instances. To do so, define the `servers` configuration parameter when you initialize the server connection. OrientJS uses these as alternate hosts in the event that it encounters an error while connecting to the primary host.

When using this method, the only requisite is that at least one of these servers must be online when the application attempts to establish its first connection.

```
var server = OrientDB({
   host:     '10.0.1.5',
   port:     2424,
   username: 'root',
   password: 'root_passwd',
   servers:  [{host: '10.0.1.5', port: 2425}]
});

var db = server.use({
   name:     'GratefulDeadConcerts',
   username: 'admin',
   password: 'admin_passwd'
});
```

The only prerequisite when executing this code is that at least one of the hosts, primary or secondary, be online at the time that it first runs. Once it has established this initial connection, OrientJS keeps the list up to date with those servers in the cluster that are currently online. It manages this through the push notifications that OrientDB sends to connected clients when changes occur in cluster membership.

> **NOTE**: Distributed Database support in OrientJS remains experimental. It is recommended that you don't use it in production.

Working with Server

Once you have initialized the `server` variable in your application, you can use it to manage and otherwise work with the OrientDB Server, such as listing the current databases and creating new databases on the server. It also provides methods to initialize the OrientJS Database API variables, which are covered in more detail elsewhere.

Listing Databases

Using the Server API you can list the current databases on the Server using the `server.list()` method. This returns a list object, which you can store as a variable and use elsewhere, or you can pass it to the `console.log()` function to print the number of databases available on the server.

```
var dbs = server.list()
   .then(
      function(list){
         console.log('Databases on Server:', list.length);
      }
   );
```

Here, the database list is set to the variable `dbs`. It then uses the `.length` operator to find the number of entries in the list. You might use the `server.list()` method in your application logic in determining whether you need to create a database before attempting to connect to one.

Creating Databases

In the event that you need to create a database on OrientDB, you can do so through the Server API using the `server.create()` method. Similar to `server.use()`, this method initializes an OrientJS Database API variable, which you can then use elsewhere to work with the specific databases on your server.

```
var db = server.create({
   name:    'BaseballStats',
   type:    'graph',
   storage: 'plocal'
   })
   .then(
      function(create){
         console.log('Created Database:', create.name);
      }
   );
```

Using this code creates a database for baseball statistics on the OrientDB Server. Within your application, it also initializes a Database API variable called `db`. Then it logs a message about what it did to the console.

Using Databases

In the event that the database already exists on your server, you can set an instance using the `server.use()` method. Similar to `server.create()`, this method initializes an OrientJS Database API variable, which you can then use elsewhere to work with the specific databases on your server.

```
var db = server.use('BaseballStats')
console.log('Using Database:', db.name);
```

Using this code initializes an instance of the `BaseballStats` database within your application called `db`. You can now use this instance to work with that database.

Closing Connections

When you're finished with the Server API, you need to close the client connection through the `server.close()` method. When you call this method, OrientJS sends a message to OrientDB that it no longer needs these resources, allowing the Server itself to reallocate them to other processes.

```
server.close()
```

OrientJS Database API

Once you have initialized the Server API, you can begin to interact with the databases on the OrientDB Server. In your code, this starts with initializing a database instance for your application, which is managed through the OrientJS Database API.

> By convention, the variable on which you initialize the Database API is called `db`. This designation is arbitrary and used only for convenience.

Initializing the Database API

In order to work with an OrientDB database in your Node.js application, you need to initialize an instance of the Database API. This provides your application with the tools it needs to access, manipulate and otherwise interact with specific databases. The initialization process is handled through the Server API, which is here called `server`, by convention.

For instance, with an existing database,

```
var db = server.use('BaseballStats')
console.log('Using Database:'  + db.name);
```

Using this code, your application would attempt to connect to the `BaseballStats` database. When it succeeds, it logs a message to the console to tell the user which database they're on.

When you're done with the database, be sure to close the instance,

```
db.close();
```

Using Credentials

In the above example, the Database API attempts to load the database using the default credentials, that is the `admin` user with the password `admin`. That's fine for a database that collects baseball statistics, but may prove problematic in the event that you want to store sensitive information. You can pass additional objects to the `server.use()` method to use credentials when you initialize the database instance.

```
var db = server.use({
   name:     'SmartHomeData',
   username:     'smarthome_user',
   password: 'smarthome_passwd'
});
```

Here, you have a database for logging information from various smart home devices. By passing in the `user` and `password` arguments, you can now use application specific credentials when interacting with the database. This allows you to implement better security practices on the database, (for instance, by restricting what `smarthome_user` can do).

Using Standalone Databases

Beginning in version 2.1.11 of OrientJS, you can initialize a database instance without needing to connect to the OrientDB Server. This is managed through the `ODatabase` class.

For instance, here the application connects to the `GratefulDeadConcerts` database and queries the vertex class, returns its length, then closes the database instance.

```
var ODatabase = require('orientjs').ODatabase;
var db = new ODatabase({
   host:     'localhost',
   port:     2424,
   username: 'admin',
   password: 'admin',
   name:     'GratefulDeadConcerts'
});

db.open().then(function() {
   return db.query('SELECT FROM V LIMIT 1');
}).then(function(res){
   console.log(res.length);
   db.close().then(function(){
      console.log('closed');
   });
});
```

Working with Databases

Methods are tied to the variable you initialize the Database API to: `db.<method>`. These can also be used to define and call other API's in the OrientJS driver.

Using the Record API

Methods tied to the Record API are called through the `db.record` object. These methods allow you to access and manipulate records directly through the Record ID. For instance, say you want to get the data from the record #1:1:

```
var rec = db.record.get('#1:1');
```

For more information on the Record API and other methods available, see Record API.

Using the Class API

Methods tied to the Class API are called through the `db.class` object. These methods allow you to create, access and manipulate classes directly through their class names. For instance, say you wanted to create a class in a baseball database for players,

```
var Player = db.class.create('Player', 'V');
```

For more information on the Class API and other methods available, see Class API.

Using the Index API

Methods tied to the Index API are called through the `db.index` object. These methods allow you to create and fetch index properties for a given class. For instance, say you want create an index on the `Player` class in your baseball database for the players' names,

```
var indexName = db.index.create({
   name: 'Player.name',
   type: 'fulltext'
});
```

For more information on the Index API and other methods, see Index API. For more information on indices in general, see Indexes.

Using the Function API

Using the Database API, you can access the Function API through the `db.createFn()` method. This allows you to create custom functions to operate on your data. For instance, in the example of a database for baseball statistics, you might want a function to calculate a player's batting average.

```
db.createFn("batAvg", function(hits, atBats){
   return hits / atBats;
});
```

For more information, see Function API.

Querying the Database

Unlike the above operations, querying the database does not require that you call a dedicated API. You can call these methods on the Database API directly, without setting or defining an additional object, using the `db.query()` method.

The `db.query` method executes an SQL query against the opened database. You can either define these values directly in SQL, or define arbitrary parameters through additional arguments.

For instance, using the baseball database, say that you want to allow users to retrieve statistical information on players. They provide the parameters and your application sorts and displays the results.

```
var targetAvg = 0.3;
var targetTeam = 'Red Sox';

var hitters = db.query(
   'SELECT name, battavg FROM Player WHERE battavg >= :ba AND team = :team',
   {params: {
      ba: targetAvg,
      team: targetTeam
    },limit: 20 }
).then(
   function(players){
      console.log(players);
   }
);
```

Here, the variables `targetAvg` and `targetTeam` are defined at the start, then the query is run against the `Player` class for Red Sox players with batting averages greater than or equal to .300, printing the return value to the console.

There are a number of more specialized query related methods supported by the Database API. For more information, see the Query guide.

Transactions

The Database API supports transactions through the `db.let()` and `db.commit()` methods. These allow you to string together a series of changes to the database and then commit them together. For instance, here is a transaction to add the player Shoeless Joe Jackson to the database:

```
var trx = db.let('player',
   function(p){
      p.create('VERTEX', 'Player')
         .set({
            name:      'Shoeless Joe Jackson',
            birthDate: '1887-07-16',
            deathDate: '1951-12-05',
            batted:    'left',
            threw:     'right'
         })
   }).commit()
   .return('$player').all();
```

For more information, see Transactions.

Events

You may find it useful to run additional code around queries outside of the normal operations for your application. For instance, as part of a logging or debugging process, to record the queries made and how OrientDB or your application performed in serving the data.

Using the `db.on()` method, you can set custom code to run on certain events, such as the beginning and ending of queries. For instance,

```
db.on("endQuery", function(obj){
   console.log("DEBUG QUERY:", obj);
});
```

For more information, see Events.

Closing the Database

When you initialize a database instance, OrientDB begins to reserve system resources for the client to access and manipulate data through the server. In order to free up these resources when you're done, you need to close the open database instance, using the `db.close()` method.

```
db.close();
```

OrientJS Record API

Once you have initialized a database instance, you can fetch and manipulate records directly through their Record ID's. Unlike many Database API features, you don't need to initialize a record object in order to manipulate the data.

> By convention, the variable on which you initialize the Database API is called `db`. This designation is arbitrary and used only for convenience.

Working with Records

Methods tied to the Record API are accessible through the Database API: `db.record.<method>` and take Record ID's as an argument.

Getting Records

Using the Record API, you can fetch records by their Record ID's using the `db.record.get()` method by their Record ID. For instance,

```
var rec = db.record.get('#1:1')
   .then(
      function(record){
         console.log('Loaded Record:', record);
       }
   );
```

Deleting Records

Using the Record API, you can delete records by their Record ID's using the `db.record.delete()` method. For instance,

```
db.record.delete('#1:1');
```

Creating Raw Binary Records

There are several methods available to you in creating records on the database. Using the Record API, you can create records through raw binary data, writing directly into OrientDB. For instance,

```
// Initialize Buffer
var binary_data = new Buffer(...);

// Define Type and Cluster
binary_data['@type'] = 'b';
binary_data['@class'] = 'Player';

// Create Record
var data = db.record.create(binary_data)
   .then(
      function(record){
         console.log('Created Record ID:', binary_data['@rid']);
      }
   );
```

Here, you initialize the `binary_data` variable as a new `Buffer()` instance. Then you set the type and cluster on which it's stored, (`@class` in this case refers to the cluster). Finally, you create the record in OrientDB, printing its Record ID to the console.

Updating a record

Using the Record API, you can updated records using previusly fetched objects wit db.record.update() method. For instance,

```
db.record.get('#5:0')
    .then(function(record){
      record.surname = 'updated'
      db.record.update(record)
         .then(function(){
          console.log("Updated");
      })
   })
```

OrientJS Class API

Once you have initialized a database instance, you can use it to access and manipulate classes in OrientDB, through the OrientJS Class API. This allows you to work with the database schema, in the event that you want to use a schema.

Methods for the Class API are accessible through the `db.class` object.

- **Working with Classes** Using these methods you can operate on the class itself: listing those available, creating, extending and updating classes, as well as fetching the object itself.
- **Working with Properties** Using these methods, you can operate on class properties: listing those available, creating, removing and updating the properties.
- **Working with Records** Using these methods, you can operate on records through the Class API: listing and creating records on the class, as an alternative to the OrientJS Query Builder.

Working with Classes

With OrientJS, you can access and manipulate classes on your OrientDB database, through the Class API. This allows you to perform various operations on the classes directly from within your Node.js application.

Listing Classes

Using the Class API, you can retrieve all current classes on the database in a list. You might do this in checking that the database has been set up or to otherwise ensure that it includes the classes your application requires to operate.

```
db.class.list()
   .then(
      function(classes){
         console.log('There are '
         + classes.length
         + ' classes in the db:',
         classes);
      }
   );
```

Here, the class list is set to the variable `classes` , which in turn uses the `length` operator to find the number of entries in the list.

Creating Classes

You can also create classes through the Class API. You may find this especially useful when setting up a schema on your database. For instance, on the `BaseballStats` database, you might want to create a class to log statistical information on particular players.

```
db.class.create('Player')
   .then(
      function(player){
         console.log('Craeted class: ' + player.name);
      }
   );
```

This creates the class `Player` in the database and a cluster `player` in which to store the data.

> To further define the schema, you need to create properties. For information on how to manage this through the Class API, see Working with Properties.

Extending Classes

Sometimes you may want to extend existing classes, such as ones that you've created previously or to create custom vertex and edge classes in a graph database. You can manage this through the Class API by passing additional arguments to the `db.class.create()` method.

For instance, in the above example you created a class for storing data on players. Consider the case where you want instances of `Player` to be vertices in a Graph Database.

```
db.class.create('Player', 'V')
   .then(
      function(player){
         console.log('Craeted Vertex Class: ' + player.name);
      }
   );
```

This creates `Player` as an extension of the vertex class `V` .

Getting Classes

In order to work with the class, you may need to retrieve the class object from OrientDB. With OrientJS this is handled by the `db.class.get()` method.

```
var player = db.class.get('Player')
   .then(
      function(player){
         console.log('Retrieved class: ' + player.name);
      }
   );
```

Updating Classes

In certain situations, you may want to update or otherwise change a class after creating it. You can do so through the `db.class.update()` method.

For instance, above there were two examples on how to create a class for baseball players, one using the default method and one creating the class as an extension of the vertex class `V`. By updating the class, you can add the super-class to an existing class, removing the need to create it all over again.

```
db.class.update({
   name: 'Player',
   superClass: 'V'
}).then(
   function(player){
      console.log(
         'Updated Class: ' + player.name
         + ' to extend ' + player.superClass
      );
   }
);
```

Working with Properties

With OrientJS, you can access and manipulate properties on your OrientDB database, through the Class API. This allows you to perform various operations on the classes directly from within your Node.js application.

The examples below use a database of baseball statistics, assuming that you've already created a class for players and initialized it in your code. For instance,

```
db.class.get('Player').then(function(Player){
   Player.property...
});
```

Methods that operate on properties use the `class.property` object, such as `Player.property.list()` .

Listing Properties

In the event that you need to check the schema or are unfamiliar with the properties created on a class, you can retrieve them into a list using the `class.property.list()` method.

For instance, say you want a list of properties set for the class `Player` :

```
Player.property.list()
   .then(
      function(properties){
         console.log(
            Player.name + 'class has the following properties: ',
            properties
         );
      }
   );
```

Creating Properties

Using OrientJS, you can define a schema for the classes from within your application by creating properties on the class with the `class.property.create()` method. For instance,

```
Player.property.create({
   name: 'name',
   type: 'String'
}).then(
   function(property){
      console.log("Created Property: " + property.name);
   }
);
```

This adds a property to the class where you can give the player's name when entering data. This is fine if you only have a few properties to create, but in the case of the example, there are a large number of values you might assign to a given player, such as dates of birth and death, team, batting averages, and so on. You can set multiple properties together by passing `class.property.create()` an array.

```
Player.property.create([
   {name: 'dateBirth',
    type: 'Date'},
   {name: 'dateDeath',
    type: 'Date'},
   {name: 'team',
    type: 'String'}
   {name: 'battingAverage',
    type: 'Float'}
]).then(
   function(property){
      console.log("Created Property: " + property.name);
   }
);
```

Deleting Properties

In the event that you find you have set a property on a class that you want to remove, you can do so from within your application using the `class.property.drop()` method. For instance, sat that you decide that you want to handle teams as a distinct class rather than a field in `Player`.

```
Player.property.drop('team').then(function(){
   console.log('Property dropped.');
});
```

Renaming Properties

In the event that you want to rename a property through OrientJS, you can do so using the `class.property.rename()` method.

```
Player.property.rename('battingAverage', 'batAvg').then(function(p){
   console.log('Property renamed');
});
```

Working with Records

With OrientJS you can access and manipulate records on your OrientDB database, through the Class API. This allows you to perform various operations on the data in a given class directly from within your Node.js application.

The examples below use a database of baseball statistics, assuming that you've already created a class for players and initialized it in your code. For instance,

```
db.class.get('Player').then(function(Player){
   Player...
});
```

Listing Records

With the Class API, you can retrieve all records on the current class into an array using the `db.class.list()` method. You might do this in operating on records or in collecting them for display in a web application.

```
Player.list()
   .then(
      function(player){
         console.log('Records Found: ' + records.length());
      }
   );
```

Here, the current list of players in the database are set to the variable `records`, which in turn uses the `length` operator to find the number of entries in the database.

Creating Records

You can also create records on OrientDB through the `db.class.create()` method. This creates the class in the database and then retrieves it into a variable for further operations.

```
Player.create({
   name:      "Ty Cobb",
   birthDate: "1886-12-18",
   deathDate: "1961-7-17",
   batted:    "left",
   threw:     "right"
}).then(
   function(player){
      console.log('Created Record: ' player.name);
   }
);
```

Here, you create an entry for the player Ty Cobb, including information on his dates of birth and death, and the sides he used when batting and throwing, which you might later collate with other players to show say the difference in batting averages and RBI's between left-side batters right-side throwers and left-side batters left-side throwers between the years 1910 and 1920.

OrientJS Index API

Once you have initialized a database instance, you can create and fetch indices from OrientDB, using the OrientJS Index API. This allows you to work with and take advantage of indexing features in OrientDB directly from within your application.

> By convention, the variable on which you initialize the Database API is called `db`. This designation is arbitrary and used only for convenience.

Working with Indices

Methods tied to the Index API are accessible through the Database API: `db.index.<method>`. In OrientDB, indices are set as properties on the class they index.

Creating Indexes

Using the Index API, you create an index property on a given class through the `db.index.create()` method. For instance,

```
var indexName = db.index.create({
   name: 'Player.name',
   type: 'fulltext'
}).then(
   function(index){
      console.log('Created Index: ' + index.name);
   }
);
```

In the baseball statistics database, you may find yourself often searching for players by name. The above example creates a Full Text index on the `name` property of `Player` using the SB-Tree indexing algorithm.

> For more information on indices in OrientDB, see Indexes.

Getting Indexes

In the event that you would like to operate on an index object directly from within your application, you can fetch using the `db.index.get()` method. You can then use the `get()` method on the return object to search for values. For instance,

```
var indexName = db.index.get('Player.name');
var cobb = indexName.get('Ty Cobb');
```

Here, on the class `Player`, you retrieve the index on the `name` property, defining it as `indexName`, then using the `get()` method you search for the player entry on Ty Cobb.

OrientJS Function API

In certain use cases you may find yourself in a situation where the available functions and methods supplied by OrientJS are not sufficient to meet your needs. To manage this, you can create your own, custom functions to use in your application.

Creating Custom Functions

The Function API is accessible through the Datatbase API. Once you initialize a database, you can create custom functions through the `db.createFn()` method.

For instance, consider the example database of baseball statistics. You might want a function to calculate a player's batting average from arguments providing hits and times at bat.

```
db.createFn("batAvg",
   function(hits, atBats){
      return hits / atBats;
   }
);
```

In the event that you don't want to define a name for the function, it defaults to `Function#name` :

```
db.createFn(function nameOfFunction(arg1, arg2){
   return arg1 + arg2;
});
```

Queries in OrientJS

Where the features described in previous chapters operate through their own API's, OrientJS handles queries as a series of methods tied to the Database API. Once you've initialized the variable for your database, (here called `db`), you can execute queries through it to get data from OrientDB.

There are two methods available in issuing queries to the database. You can query it directly in SQL. Alternatively, you can use the Query Builder methods to construct your query in Node.js.

Working with Queries

When querying the database directly, you prepare a string containing OrientDB SQL, then pass it to either the `db.query()` or `db.exec()` methods. For instance, going back to the baseball database, say that you want a list of players with a batting average of at least .300 that played for the Red Sox.

```
db.query(
   'SELECT name, ba FROM Player '
   + 'WHERE ba >= 0.3 AND team = "Red Sox"'
).then(function(hitters){
   console.log(hitters)
});
```

Using Parameters

In the event that you want to query other teams and batting averages, such as the case where you want your application to serve a website, you can use previously defined variables, (here, `targetBA` and `targetTeam`), to set these parameters. You can also add a limit to minimize the amount of data your application needs to parse.

> Do not enclose param placeholders in single/double quotes in query string. Query parameters are properly encoded by their type. Given example below, the executed query will be `SELECT name, ba FROM Player WHERE ba >= 0.3 AND team = "Red Sox" LIMIT 20`

```
var targetBA = 0.3;
var targetTeam = 'Red Sox';

db.query(
   'SELECT name, ba FROM Player '
   + 'WHERE ba >= :ba AND team = :team',
   {params:{
            ba: targetBA,
            team: targetTeam
           },
    limit: 20
   }
).then(function(hitters){
   console.log(hitters)
});
```

Using Fetching Strategies

While the `query()` method supports most OrientDB SQL statements, there are some limitations in regards to Fetching Strategies.

The `query()` method does not support Fetching Strategies directly.For instance, say that you want to query the player records in your baseball database with baseball card images. From the OrientDB Console, you might issue this command:

```
orientdb> SELECT name, OUT("hasCard") AS cards
          FROM Player FETCHPLAN *:1
```

This would return a result-set with the player's name in one colmn and an array from the `Card` class with all properties on that class, (as expected). If instead, you passed this query in OrientJS through the `query()` method, you would meet with different results:

```
db.query(
   'SELECT name, OUT("hasCard") AS cards '
   + 'FROM Player FETCHPLAN *:1')
   .then(function(results){
   console.log(results)
})
```

Using this code, instead of fetching and unpacking the relationship connecting the `Player` and `Card` class, the `card` column would only contain an array of Record ID's, ignoring the Fetch Plan. The reason is that it expects to parse Fetch Plans as part of a parameter option, rather than with the query itself. To use Fetching Strategies with OrientJS, pass it as a parameter:

```
var results = db.query(
   'SELECT name, OUT("hasCard") AS cards '
   + 'FROM Player',
   {fetchPlan: 'hasCard:1'})
    .then(function(results){
   console.log(results)
})
```

Query Builder

Rather than writing out query strings in SQL, you can alternatively use the OrientJS Query Builder to construct queries using a series of methods connected through the Database API.

Method	Description
`create()`	Creates vertices and edges.
`delete()`	Removes vertices, edges, and records.
`fetch()`	Defines a Fetching Strategy.
`insert()`	Adds records.
`liveQuery()`	Executes a Live Query.
`select()`	Fetches records by query.
`transform()`	Modifies records in transit.
`traverse()`	Fetches records by traversal.
`update()`	Modifies records on database.

OrientJS - `create()`

Creation queries in OrientJS are those used in creating vertex and edge records on a Graph Database. Given the added complexity of regular or lightweight edges running between the vertices, adding records is a little more complicated than the `insert()` method you might use otherwise.

The creation query method is comparable to the `CREATE VERTEX` and `CREATE EDGE` commands on the OrientDB Console.

Working with Creation Queries

In OrientJS, creating vertices and edges uses the `create()` method. The examples below operate on a database of baseball statistics, which has been initialized on the `db` variable.

Creating Vertices

Create an empty vertex on the `Player` vertex class:

```
var player = db.create('VERTEX', `Player`).one();
console.log('Created Vertex:', player);
```

Create a vertex with properties:

```
var player = db.create('VERTEX', 'Player')
   .set({
      name:      'Ty Cobb',
      birthDate: '1886-12-18',
      deathDate: '1961-7-17',
      batted:    'left',
      threw:     'right'
   }).one().then(
      function(player){
         console.log('Created Vertex: ' + player.name);
      }
   );
```

Creating Edges

Creating edges to connect two vertices follows the same pattern as creating vertices, with the addition of `from()` and `to()` methods to show the edge direction.

For instance, consider an edge class `PlaysFor` that you use to connect player and team vertices. Using it, you might create a simple edge that simply establishes the connection:

```
var playsFor = db.create('EDGE', 'PlaysFor')
   .from('#12:12').to('#12:13').one();
```

This creates an edge between the player Ty Cobb, (#12:12), and the Detroit Tigers, (#12:13). While this approach may be useful with players that stay with the same team, many don't. In order to account for this, you would need to define properties on the edge.

```
var playsFor = db.create('EDGE', 'PlaysFor')
   .from('#12:12').to('#12:13')
   .set({
      startYear: "1905",
      endYear: "1926",
   }).one();
```

Now, whenever you build queries to show the players for a team, you can include conditionals to only show what teams they played for in a given year.

OrientJS - `delete()`

Deletion queries in OrientJS are those used in removing records from the database. It can also account for edges between vertices, updating the graph to maintain its consistency.

The deletion query method is comparable to `DELETE` , `DELETE VERTEX` and the `DELETE EDGE` statements.

Working with Deletion Queries

In OrientJS, deletions use the `delete()` method. The examples below operate on a database of baseball statistics, which has been initialized on the `db` variable.

Deleting Vertices

With Graph Databases, deleting vertices is a little more complicated than the normal process of deleting records. You need to tell OrientDB that you want to delete a vertex to ensure that it takes the additional steps necessary to update the connecting edges.

For instance, you find that you have two entries made for the player Shoeless Joe Jackson, #12:24 and #12:84 you decide to remove the extra instance within your application.

```
db.delete('VERTEX','Player')
   .where('@rid = #12:84').one()
   .then(
      function(del){
         console.log('Records Deleted: ' + del);
      }
   );
```

Deleting Edges

When deleting edges, you need to define the vertices that the edge connects. For instance, consider the case where you have a bad entry on the `playsFor` edge, where you have Ty Cobb assigned to the Chicago Cubs. Ty Cobb has a Record ID of #12:12, the Chicago Cubs #12:54.

```
db.delete('EDGE', 'PlaysFor')
   .from('#12:12').to('#12:54')
   .scalar()
   .then(
      function(del){
         console.log('Records Deleted: ' + del);
      }
   );
```

Deleting Records

In order to delete records in a given class, you need to define a conditional value that tells OrientDB the specific records in the class that you want to delete. When working from the Console, you would use the `WHERE` clause. In OrientJS, set the `where()` method.

```
db.delete().from('Player')
   .where('@rid = #12:84').limit(1).scalar()
   .then(
      function(del){
         console.log('Records Deleted: ' + del);
      }
   );
```

OrientJS - `fetch()`

In OrientDB, sometimes the default behavior of the `SELECT` command is not sufficient for your needs. For instance, when your application connects to a remote server, using a fetching strategy can limit the number of times it needs to connect to the remote server.

> For more information, see Fetching Strategies.

Working with Fetching Strategies

Using Fetching Strategies in OrientJS operates through the `fetch()` method tied into a larger query.

Using Depth Levels

When traversing relationships to a vertex, queries can return a great deal more data than you need for the operation. By defining the depth level, you can limit the number of traversals the query crosses in collecting the data.

For instance, say you want a list of active users on your database. Using this method, you can limit the size of the load by only traversing to the fifth level.

```
var users = db.select().from('OUser')
   .where({
     status: 'ACTIVE'
   }).fetch({
      role: 5
   }).all()
   .then(
      function(data){
         console.log('Active Users: ' + users);
      }
   );
```

OrientJS - `insert()`

Insertion queries in OrientJS are those that add records of a given class into the database. The insertion query method is comparable to the `INSERT` commands on the OrientDB Console.

Working with Insertion Queries

In OrientJS, inserting data into the database uses the `insert()` method. For instance, say that you want to add batting averages, runs and runs batted in for Ty Cobb.

```
db.insert().into('Player')
   .set({
     ba:  0.367,
     r:   2246,
     rbi: 1938
   }).where('name = "Ty Cobb"').one().then(function(player){
      console.log(player)
   });
```

Raw Expressions

with set

```
db.insert().into('Player')
   .set({
     uuid : db.rawExpression("format('%s',uuid())"),
     ba:  0.367,
     r:   2246,
     rbi: 1938
   }).where('name = "Ty Cobb"').one().then(function(player){
      console.log(player)
   });
```

Generated query

```
INSERT INTO Player SET uuid = format('%s',uuid()), ba = 0.367, r = 2246, rbi = 1938
```

OrientJS - `liveQuery()`

When using traditional queries, such as those called with `db.query()` and `db.select()` you only get data that is current at the time the query is issued. Beginning in version 2.1, OrientDB now supports Live Queries, where in issuing the query you tell OrientDB you want it to push affecting changes to your application.

You can execute Live Queries using the `db.liveQuery()` method with a `LIVE SELECT` statement passed as its argument.

> IMPORTANT: From OrientDB 2.2.x in order to make live query work the token must be enabled. See here

Understanding Live Queries

Traditional queries provide you with information that is current at the time the query is issued. In most cases, such as well pulling statistical data on long-dead ball players like Ty Cobb, this behavior is sufficient to your needs. But, what if about when you need real time information.

For instance, what if in addition to historical data you also want your application to serve real-time information about baseball games as they're being played.

With the traditional query, you would have to reissue the query within a set interval to update the application. Live Queries allow you to register events, so that your application performs addition operations in the event of an `INSERT`, `DELETE`, or `UPDATE`.

For example, say that you have a web application that uses the baseball database. The application serves the current score and various other stats for the game. Whenever your back-end system inserts new records for the game, you can execute a function to update the display information.

> For more information on event handlers in OrientJS, see Events.

Working with Live Queries

In OrientJS, Live Queries are called using the `db.liveQuery()` method. This is similar to `db.query()` in that you use it to issue the raw SQL of a `LIVE SELECT` statement. Unlike `db.query()`, you can assign event handlers to `db.liveQuery` using the `on()` method.

For instance,

```
db.liveQuery('LIVE SELECT FROM V')
   .on('live-update', function(data){
      var myRecord = data.content;
   });
```

This would assign to the variable `myRecord` object data in response to each update made to the class `V`. For instance,

```
db.update('#12:97')
   .set({
      ba: 0.321
   }).one();
```

When your application runs this method, it also sets the `myRecord` variable to

```
var myRecord = {
   content: {
      @rid: $12:97,
      ba: 0.321
   },
   operation: update
};
```

You can then use this information in your code to determine what your application should do in response.

Running Events on Insert

When you pass the `liveQuery().on()` method the string `live-insert`, it executes the function argument whenever an insert operation is performed on the class.

For instance, say that you want to pass statistics on new plays to your game monitoring application:

```
db.liveQuery('LIVE SELECT FROM Game '
             + WHERE game_id = "201606-001"')
   .on('live-insert', function(data){
      var gameStat = data.content;
          if (gameStat.score != currentScore){
         var score = updateScore(gameStat.score);
         console.log("Updated Score:", score);
      }
   });
```

Here, the Live Query registers an event to `INSERT` events on the `Game` class where the `game_id` property equals the one currently being played. Whenever the back-end service inserts data on a new play, the application checks for changes in the score. When either team scores, it passes the score to an `updateScore` function, which we might use to trigger a notification event in the web browser to let the user know their team scored run or runs.

Running Events on Delete

When you pass the `liveQuery().on()` method the string `live-delete`, it executes the function argument whenever a delete operation is performed on the class.

For instance, for the baseball database, say that you only calculate player batting averages after each game instead of on the fly. You might set up an event to process new batting averages whenever the stored values are deleted.

```
db.liveQuery('LIVE SELECT rid, name, ba FROM Player`)
   .on('live-delete' function(data.content){
      var player = data.content;
            var newBA = genBatAvg(player.rid);
      db.update(player.rid)
         .set({
            ba: newBa
         }).one()
         then(
            function(update){
               console.log("Updated BA: " update.name);
            }
         );
   });
```

Now, whenever your application issues a `delete()` statement against the player's batting average, it triggers this function to generate a new batting average and runs the `update()` method to apply the changes to the player's record. Once this operation is complete, all queries made against this class will use the new data.

Running Events on Updates

When you pass the `liveQuery().on()` method the string `live-update`, it executes the function argument whenever an update operation is performed on the class.

For instance, say that as a security precaution, you would like your application to email you whenever certain key changes are made to the `OUser` class, such as an update that escalates user privileges. You might use something like this,

```
db.liveQuery('LIVE SELECT FROM OUser')
   .on('live-update', function(data){
      var report = reportChange(data.content);
      console.log("Report:", report);
   });
```

When your application contains this code, any changes on the database that issue `UPDATE` queries to the `OUser` class are passed to the `reportChange()` function, which you can use to determine whether or not the changes are legitimate and if they're not, what you want it to do or who it should notify about suspicious activity.

OrientJS - `select()`

Selection queries in OrientJS are those used to fetch data from the database, so that you can operate on it with your application. The method is comparable to the `SELECT` command in the OrientDB Console.

Working with Selection Queries

In OrientJS, fetching data from the database uses the `select()` method. The examples below operate on a database of baseball statistics, which has been initialized on the `db` variable.

Selecting Records

Use the `select()` method to fetch records from the database. For instance, say you want a list of all players with a batting average of .300.

```
var hitters = db.select().from('Player')
   .where({
      ba: 0.3
   }).all()
   .then(
      function(select){
         console.log('Hitters:', select);
      }
   );
```

Searching Records

Using the `containsText()` method, you can search records for specific strings. For instance, say that a user would like information on a particular player, but does not remember the player's full name. So, they want to fetch every player with the string `Ty` in their name.

```
var results = db.select().from('Player')
   .containsText({
      name: 'Ty'
   }).all()
   .then(
      function(select){
         console.log('Found Players:', select);
      }
   );
```

Using Expressions

In the event that you would like to return only certain properties on the class or use OrientDB functions, you can pass the expression as an argument to the `select()` method.

For instance, say you want to know how many players threw right and batted left:

```
var count = db.select('count(*)').from('Player')
   .where({
      threw:  'right',
      batted: 'left'
   }).scalar()
   .then(
      function(select){
         console.log('Total:', select);
      }
   );
```

Returning Specific Fields

Similar to expressions, by passing arguments to the method you can define a specific field that you want to return. For instance, say you want to list the teams in your baseball database, such as in building links on a directory page.

```
var teams = db.select('name')
   .from('Teams').all()
   .then(
      function(select){
         console.log('Loading Teams:', select);
      }
   );
```

Specifying Default Values

Occasionally, you may encounter issues where your queries return empty fields. In the event that this creates issues for how your application renders data, you can define default values for certain fields as OrientJS sets them on the variable.

For instance, in the example of the baseball database, consider a case where for some historical players the data is either currently incomplete or entirely unavailable. Rather than serving null values, you can set default values using the `defaults()` method.

```
var player = db.select('name').from('Player')
   .defaults({
      throws: 'Unknown',
      bats: 'Unknown'
    }).all();
```

OrientJS - `transform()`

When working with queries in OrientJS, you may sometimes need to operate on data before setting it to the variable. For instance, in cases where the data stored in the database is not in a format that suits your needs. Transformation allows you to define functions within your application to update or alter the data as it comes in from the database.

Working with Transformation Queries

In OrientJS, transformation operations are controlled through the `transform()` method as part of a larger query that fetches the data on which you want to operate.

Transforming Fields

Most of the time, you may find that the data in the database is sufficient to your needs, save for the case of certain fields. Using this method, you can define transformation functions for individual fields.

For instance, OrientDB stores the user status as a string in all caps, which you may find unsuitable as a return value to display on the user profile.

```
var user = db.select('name').from('OUser')
   .where({
      status: 'ACTIVE'
   }).transform({
      status: function(status){
         return status.toLowerCase();
      }
   }).limit(1).one()
   .then(
      function(select){
         console.log('User Status Transformed:', select.status);
      }
   );
```

In the `transform()` method, the `toLowerCase()` method operates on the return status to convert `ACTIVE` to `active`.

Transforming Records

On occasion, you may need to operate on the entire record rather than individual fields.

```
var user = db.select('name').from('OUser')
   .where({
      status: 'ACTIVE'
   }).transform(
      function(record){
         return new User(record);
      }
   ).limit(1).one()
   .then(
      function(select){
         console.loog(
            'Is the user an instance of User?',
            (user.instanceof User)
         );
      }
   );
```

OrientJS - `traverse()`

Traversal queries in OrientJS are those used to traverse records, crossing relationships. This works in both Graph and Document Databases. The method is comparable to the `TRAVERSE` command in the OrientDB Console.

> Bear in mind, in many cases you may find it sufficient to use the `db.select()` method, which can result in shorter and faster queries. For more information, see `TRAVERSE` versus `SELECT`.

Working with Traversal Queries

In OrientJS, traversal operations use the `traverse()` method. The example below uses a database of baseball statistics, which has been initialized on the `db` variable.

Traverse All Records

The simplest use of the method is to traverse all records connected to a particular class. For instance, say you want to see see all records associated with the Boston Red Sox, which has a Record ID of #12:45.

```
var records = db.traverse()
   .where({name: 'Boston Red Sox'})
   .all()
   .then(
      function(trav){
         console.log('Found Records:', trav);
      }
   );
```

OrientJS - `update()`

Update queries in OrientJS are those used in changing or otherwise modifying existing records in the database. The method is comparable to the `UPDATE` command on the OrientDB Console.

Working with Update Queries

In OrientJS, updating records works through the `update()` method. The examples below operate on a database of baseball statistics, which has been initialized on the `db` variable.

Updating Records

When the record already exists in the database, you can update the content to modify the current values stored on the class.

Consider the case where you want to update player data after a game. For instance, last night the Boston Red Sox played and you want to update the batting average for the player Tito Ortiz, who has a Record ID #12:97. You've set the new batting average on the variable `newBA` and now you want to pass the new data to OrientDB.

```
db.update('#12:97')
   .set({
      ba: newBA
   }).one()
   .then(
      function(update){
         console.log('Records Updated:', update);
      }
   );
```

Putting in Map Entries

When working with map fields, you sometimes need to put entries into map properties. In the OrientDB Console, you can do this with the `PUT` clause, in OrientJS you can do so through the `put()` method.

```
db.update('#1:1')
   .put('mapProperty', {
      key: 'value',
      foo: 'bar'
   })
   .scalar()
   .then(
      function(update){
         console.log('Records Updated:', update);
      }
   );
```

Transactions in OrientJS

Rather than writing out SQL commands to manage transactions, you can use the OrientJS Transaction Builder to construction transactions using a series of methods connected through the Database API.

Working with the Transaction Builder

Transactions are built through a string of `let()` methods. With each, define functions that execute commands on the database. When the operation is complete, call the `commit()` method to commit your changes to the database.

Methods used in transactions include,

- **`let()`** Method takes as arguments a name and a function that defines one operation in the transaction. You can reference the name elsewhere in the transaction using the `$` symbol, (for instance, `$name`).
- **`commit()`** Method commits the transaction to the database.
- **`return()`** Method defines the transaction return value.

For instance, this transaction adds vertices for Ty Cobb and the Detroit Tigers to the database, then defines an edge for the years in which Cobb played for the Tigers.

```
var trx = db.let('player', function(p){
      p.create('vertex', 'Player')
         .set({
            name:      'Ty Cobb',
            birthDate: '1886-12-18',
            deathDate: '1961-7-17',
            batted:    'left',
            threw:     'right'
         })
   })
   .let('team', function(t){
      t.create('vertex', 'Team')
         .set({
            name: 'Tigers',
            city: 'Detroit',
            state: 'Michigan'
         })
   })
   .let('career', function(c){
      c.create('edge', 'playsFor')
         .from('$player')
         .to('$team')
         .set({
            startYear: '1905',
            endYear:   '1926'
         })
   })
   .commit().return('$edge').all()
   .then(
      function(results){
         console.log(results);
      }
    );
```

Working with Batch Scripts

In addition to the standard transactions, you can also execute raw batch scripts, without using the transaction builder. These are the equivalent of SQL Batch scripting in the OrientDB Console.

```
db.query('begin;'
   + 'let $t0 = SELECT FROM V LIMIT 1;'
   + 'return $t0'
   ,{class: 's'}).then(function(res){
      console.log(res);
   });
```

OrientJS Events

On occasion, you may find the need to run additional code around queries outside of the standard query methods. In OrientJS you can program events to run whenever queries begin and end.

You may find this useful in implementing logging functionality or profiling your system, or in initiating housekeeping tasks in response to certain types of queries.

Working with Events

The method to create events is accessible through the Database API, using `db.on()`. The second argument defines the function it executes, the first the event that triggers the function.

Running Events on Query Start

When you pass the `db.on()` method the string `beginQuery`, it executes the function argument whenever a query starts. So, for instance, if your application executes the following query,

```
var query = db.select('name, status').from('OUser')
   .where({"status": "active"})
   .limit(1)
   .fetch({"role": 1})
   .one();
```

The function in `beginQuery` event receives an object that contains data on the query it's executing, this is similar to the data set on the `obj` variable below:

```
var obj = {
   query: 'SELECT name, status FROM OUser'
          + 'WHERE status = :paramstatus0 LIMIT 1',
   mode: 'a',
   fetchPlan: 'role:1',
   limit: -1,
   params: {
      params: {
         paramstatus0: 'active'
      }
   }
}
```

For instance, say that there is something wrong with your application and for the purposes of debugging, you want to log all queries made to the database.

```
db.on("beginQuery", function(obj){
   console.log('DEBUG: ', obj);
});
```

Running Events on Query End

In addition to running your event code at the start of the query, by passing the `endQuery` string to the `db.on()` method, you can define a function to execute after the query finishes.

The function in `endQuery` events receives an object containing data about how the query ran, similar to the data set on the `obj` variable below:

```
var obj = {
   {
      "err": errObj,
      "result": resultObj,
      "perf": {
         "query": timeInMs
      },
      "input" : inputObj
   }
}
```

Where `inputObj` is the data used for the `beginQuery` event.

For instance, when debugging you might use this event to log errors and performance data to the console:

```
db.on("endQuery", function(obj){
   console.log("DEBUG: ", obj);
});
```

Running Events on Live Queries

Beginning in version 2.1, OrientDB introduces Live Queries, which provides support for `INSERT`, `DELETE`, and `UPDATE` events to OrientJS applications.

Unlike other events, these are not set to the Database API itself, but rather through a `LIVE SELECT` query that determines what records you want the application to monitor.

For more information and examples, see `liveQuery()`.

PyOrient

OrientDB supports all JVM languages for server-side scripting. Using the PyOrient module, you can develop database applications for OrientDB using the Python language.

It works with version 1.7 and later of OrientDB.

Installation

In order to use PyOrient, you need to install it on your system. There are two methods available to you in doing this: you can install it from the Python Package Index (PyPI) or you can download the latest source code from GitHub and build it on your local system.

You'll need to have Python in desired version, (2 or 3), as well as the Python package management tool pip installed on your system as a prerequisite.

Installing from PyPI

When you call pip and provided it with the specific package name, it connects to the Python Package Index and downloads the current source package for PyOrient. It then runs the setup script to install the package on your system, either at a global or user level. Installing it at a system-level requires that you run pip as root or with sudo and installs the package as a system-level Python library, (for instance `/usr/lib/python3.4/site-packages/pyorient`). Installing it at a local-level does not require root privileges and installs it as a user-level Python library, (for instance `~/.local/lib/python3.4/site-packages/pyorient`).

- To install PyOrient locally, run the following command:

  ```
  $ pip install pyorient --user
  ```

- To install PyOrient globally, instead run this command:

  ```
  $ sudo pip install pyorient
  ```

When pip runs, it calls the default Python interpreter `python` . Due to backwards compatibility issues, in some situations you may find this behavior undesirable. For instance, on many Linux distributions and Mac OS X, 2.7 is the default installation of Python. If yo install PyOrient using the above command and then attempt to develop an application that uses Python 3.5, you'll find the `pyorient` module unavailable.

In the event that you don't know what version you have installed, you can run this command to see:

```
$ python --version
Python 2.7.11
```

To install PyOrient for a specific version of Python, call pip as a module for the desired Python interpreter. For instance,

```
$ python3 -m pip install pyorient --user
```

Building from Source

In the event that you would like to contribute to the PyOrient Project or for whatever reason would prefer to work from a development release of PyOrient, you can clone the repository from GitHub and install it on your system, again either at a local- or system-level.

You can clone the repository using the following command:

```
$ git clone https://github.com/orientechnologies/pyorient
```

Then from within that directory you can install PyOrient using the setup script:

- To install PyOrient locally, run the following command:

  ```
  $ python setup.py install --user
  ```

- To install PyOrient globally, instead run this command:

  ```
  $ sudo python setup.py install
  ```

Notice that these commands utilize the default Python interpreter, whichever version that happens to be on your system. This may create issues if you intend to use PyOrient with a version of Python that is not your system default. To run the installation, substitute the interpreter in the above commands with the version that you want to use. For instance,

```
$ python3 setup.py install --user
```

Running Tests

When working with development releases, it is advisable that you run tests after installing PyOrient. If you would like to contribute code to the project, it's required that you also develop tests for your contributions, to avoid issues in future development.

Testing PyOrient requires that you have Apache Maven and Nose installed on your system. To run the test, first bootstrap OrientDB by running the following command from the `pyorient/` directory:

```
$ ./ci/start-ci.sh
```

Running this command downloads the latest version of OrientDB into a local directory then makes some changes to its configuration file and installs a database for use in the test. Once it's done, you can run the test:

```
$ nosetests
```

Using PyOrient

Once you have PyOrient installed you can begin to use it in your applications. You can load it through the standard Python import system. There are two modules available to you:

- **PyOrient Client** The base module provides a Python wrapper to the OrientDB Binary Protocol. To use it, add the following line to your application:

  ```
  import pyorient
  ```

- **PyOrient OGM** The Object-Graph Mapper for PyOrient provides a higher-level Object Oriented Pythonic interface for Graph Databases in OrientDB. To use it, add the following line to your application:

  ```
  import pyorient.ogm
  ```

You may need one or both for your application, depending upon your particular needs.

PyOrient Client

The base module in PyOrient provides a Python wrapper for the OrientDB Binary Protocol. Using this wrapper, you can initialize a client instance within your application, then operate on the OrientDB Server through this instance.

In order to use the PyOrient Client, you need to import the base module into your application.

```
import pyorient
```

Initializing the Client

In order to use the PyOrient Client, you need to initialize a client instance in your application. By convention, this instance is called `client`, but you can use any object name you prefer.

To use the client, first initialize a the client object, then connect to the OrientDB Server:

```
client = pyorient.OrientDB("localhost", 2424)
session_id = client.connect("admin", "admin_passwd")
```

Here, you initialize the `client` object to connect to OrientDB through the localhost interface on port 2424. Then, you establish a connection with the Server, using the username `admin` and the password `admin_passwd`.

Server Shutdown

From within your application, you can shut down the OrientDB Server. The user that initiates the shutdown must have shutdown permissions in the `orientdb-server-config.xml` configuration file, (for instance, the `root` user on the OrientDB Server).

```
client.shutdown('root', 'root_passwd')
```

Working with the Client

Within your application, once you have initialized the PyOrient Client this object provides you with an interface in working databases on the OrientDB Server.

- `command()` This method issues SQL commands.
- `batch()` This method issues batch commands.
- `data_cluster_add()` This method creates new clusters on the database.
- `data_cluster_count()` This method counts the numbers of records in an array of clusters.
- `data_cluster_data_range()` This method retrieves all records in the given cluster.
- `data_cluster_drop()` This method removes a cluster from the database.
- `db_count_records()` This method counts the number of records on a database.
- `db_create()` This method creates databases.
- `db_drop()` This method removes databases.
- `db_exists()` This method determines if a database exists.
- `db_list()` This method lists databases on the server.
- `db_open()` This method opens a database on the server.
- `db_reload()` This method reloads the database on the client.
- `db_size()` This method returns the size of the database.
- `get_session_token()` ` This method returns the client session token.
- `query()` This method issues synchronous queries to the database.
- `query_async()` This method issues asynchronous queries to the database.
- `record_create()` This method creates records on the database.
- `record_delete()` This method removes records from the database.
- `record_load()` This method retrieves records from the database.

- `record_update()` This method updates records on the database.
- `set_session_token()` This method enables and loads a token for the client session.
- `tx_commit()` THis method initializes a control object for transactions.

PyOrient Client - `command()`

This method allows you to issue SQL commands to an open OrientDB database.

Sending Commands

There are several methods available in issuing queries and commands to OrientDB. Using the `command()` method calls the `OCommandSQL` Java class in OrientDB. This allows you issue commands to OrientDB through your application as you would from the Console.

Syntax

```
client.command(<sql-command>)
```

- **`<sql-command>`** Defines the command you want to issue.

Example

Going back to the example of a smart home database, consider the use case of logging environmental information to the database for later analysis. In each room in your house, you set up a series of small Arduino devices to monitor light, sound levels, pollen count and so on. Every fifteen minutes, your application needs to pull data off each device and log on the database for later graphing and analysis.

```
for sensor in pollen_sensors:
     client.command(
      "INSERT INTO PollenSensor "
      "('device_id', 'read_time', 'read') "
      "VALUES('%s', '%s', %s')"
      % (sensor.get_id()
         time.now(),
         sensor.get_data()))
```

Here, the application iterates through an array of pollen sensors. For each entry, it issues an `INSERT` statement to OrientDB, retrieving the identifier and data from the sensor object and using the Python `time` module to timestamp the entry.

PyOrient Client - `batch()`

This methods allows you to execute a series of SQL commands as a batch operation.

Executing Batch Operations

Normally, when you execute a series of commands in PyOrient, the operations are run in sequence. If your application is connecting to a remote server, this means that the PyOrient client is connecting over the network for each commands, which can introduce latency issues for your application.

Batch operations allow you to group several commands together, then issue them together in a single operation.

Syntax

```
client.batch(<batch>)
```

- `<batch>` Defines the batch commands.

Example

Consider the example of a smart home system that uses OrientDB as its back-end database. You have created and installed various environmental sensors around the house. Every fifteen minutes, your applications takes readings from these sensors and adds them to the database. Given that there may be several dozen sensors in the house, you may find it advantageous to update OrientDB through a batch operation.

```
# Initialize Batch Commands Array
batch_cmds = ['begin']

# Set Time
time_now = time.now()

# Add Commands to Array
for sensor in sensors:
   node = sensor.node
   zone = sensor.zone
   read = read_sensor(sensor)
   command = ("let %s = CREATE VERTEX Node"
              "SET node ='%s', zone = '%s',"
          "read = '%s', time = '%s"
          % (node, node, zone, time_now, read))
   batch_cmds.append(command)

# Add Batch Commit
batch_cmds.append('commit retry 100;')

# Join with Semicolons
cmd = ';'.join(batch_cmds)

# Execute Commands
results = client.batch(cmd)
```

Here, you have an array of record objects for each sensor in the house. Iterating over that array, you extract the node name and zone, then take a reading from the sensor and use it in defining a `CREATE VERTEX` batch command. Once you have a command for each sensor, it joins the batch commands with a commit message, creating a string object to pass to `batch()`.

PyOrient Client - `data_cluster_add()`

This method creates new clusters on the connected OrientDB Server.

Creating Clusters

In OrientDB, the cluster is the place in memory or on disk where the database stores its records. Using PyOrient, you can create new clusters from within your application using the `data_cluster_add()` method. New cluster can be created as physical (that is, on disk), or in memory.

Syntax

```
client.data_cluster_add(<cluster-name>, <cluster-type>)
```

- **`<cluster-name>`** Defines the cluster name.
- **`<cluster-type>`** Defines the cluster type:
  - *`pyorient.CLUSTER_TYPE_PHYSICAL`* Creates a physical cluster.
  - *`pyorient.CLUSTER_TYPE_MEMORY`* Creates an in-memory cluster.

> For more information, see Clusters.

Example

Consider the example of a database for a smart home management application. When the application runs for the first time, you'll need to initialize OrientDB with any clusters the application requires to operate. For instance, say you want your various environmental sensors to use different clusters for different rooms in the house:

```
rooms = ['livingRoom', 'masterBedroom', 'guestBedroom',
         'kitchen', 'bathroom', 'porch']

# Create a Cluster for Each Room
for room in rooms:

   client.data_cluster_add(room, pyorient.CLUSTER_TYPE_PHYSICAL)

   logging.info('Created Physical Cluster: %s' % room)
```

Here, your application loops over a list of cluster names and creates each instance as a physical cluster on the OrientDB Server.

PyOrient Client - `data_cluster_count()`

This method returns the number of records in an array of clusters.

Counting Cluster Records

Using the `data_cluster_count()` method, you can retrieve a count of the number of records stored on a given cluster. You might find this useful for maintenance tasks or when you need a more granular record count than what the `db_count_records()` method provides.

Syntax

```
client.data_cluster_count([<cluster-ids>])
```

- **`<cluster-ids>`** Defines an array of Cluster ID's.

> For more information, see Clusters.

Example

For instance, consider the example of an application to manage smart home devices. In each room you create a physical cluster to store data from environmental sensors. For the web application, you want a count of the number of sensors in an array of rooms.

```
# Retrieve Record Count by Zone
def zone_count(client, cluster_ids, zone_name):

   # Fetch Record Count
   count = client.data_cluster_count(cluster_ids)

   # Log Findings
   logging.info("Found %s records for %s"
               % (str(count), zone_name))

   # Return
   return count
```

Where records are organized in clusters by room, for display purposes rooms are grouped together into zones. This function receives the PyOrient Client, an array of Cluster ID's for a given zone and the zone name. Using the Cluster ID's, it fetches a record count for that zone, uses the logging module to report its findings, then passes the count back as a return value.

PyOrient Client - `data_cluster_data_range()`

This method returns a range of Record ID's for the given cluster.

Retrieving Cluster Records

Using the `data_cluster_data_range()` method, you can retrieve all records stored in a particular cluster. You may find this particularly useful in implementations that organize similar records by storing them in dedicated clusters.

Syntax

```
client.data_cluster_data_range(<cluster-id>)
```

- **`<cluster-id>`** Defines an integer for the Cluster ID.

> For more information, see Clusters.

Example

Consider the example of a smart home management application that maintains records using OrientDB. You have built a series of Arduino or Micro Python devices to monitor environmental conditions around the house, (that is, light, temperature, pollen levels, and so on), and have created a class in OrientDB to store data from these sensors. To better organize this data, for each room in your house you have a dedicated cluster to store records from these sensors.

```
# Retrieve Sensor Data
def get_sensor(client, cluster_id):

   # Retrieve Data
   data = client.data_cluster_data_range(cluster_id)
   return data
```

Here, the function receives the `client` object and an integer that indicates the Cluster ID for the room you want to access. It then calls the `data_cluster_data_range()` method with these arguments to retrieve all records in that cluster.

PyOrient Client - `data_cluster_drop()`

This method removes a cluster from the database.

Removing Clusters

Occasionally, you may find the need to remove clusters from your database. You can do so through the PyOrient Client using the `data_cluster_drop()` method.

Syntax

```
client.data_cluster_drop(<cluster-id>)
```

- **`<cluster-id>`** Defines the ID of the cluster.

> For more information, see Clusters.

Example

Consider the example of the smart home application. Environmental sensors for each room write their data to a dedicated cluster on OrientDB. Every so often users will move and take the system with them to a new house. The new house may not have the same number of rooms as the old, which opens up the need for the application to delete clusters in setting the system back up:

```
# Remove Cluster
def remove_cluster(client, name, cluster_id):

   # Remove Cluster
   client.data_cluster_drop(cluster_id)

   # Log Event
   logging.info("Removed Cluster %s for %s"
               % (str(cluster_id), name))
```

Here, the function receives the PyOrient Client, the room name and the Cluster ID for sensor nodes in that room as arguments. It removes the cluster, then reports on what it did to the logging module.

PyOrient Client - `db_count_records()`

This method allows you to get the number of records in an opened OrientDB database.

Counting Records

There are two methods available in determining the size of the opened database. `db_size()` returns the size of the database, while `db_count_records()`, which returns the number of records in the database. It takes no arguments and returns a long value.

Syntax

```
client.db_count_records()
```

Example

You might use this method as a basic sanity check in your application. For instance, you might use `db_exists()` to determine whether the database was created, then you can use the `db_count_records()` to determine whether or not the database is empty.

```
# Check that Database exists and contains Data
if client.db_exists("tinkerhome"):
   assert client.db_count_records() != 0
```

PyOrient Client - `db_create()`

Creates a database on the connected OrientDB Server.

Creating Databases

In the event that a database does not exist already, you can create one from within your application, using the `db_create()` method. This method requires one argument, but can take two others.

Syntax

```
client.db_create(<name>, <database-type>, <storage-type>)
```

- `<name>` Defines the database name.
- `<database-type>` Defines the database type, (optional):
  - `pyorient.DB_TYPE_DOCUMENT` Creates a Document Database.
  - `pyorient.DB_TYPE_GRAPH` Creates a Graph Database.
- `<storage-type>` Defines the storage type (optional):
  - `pyorient.STORAGE_TYPE_PLOCAL` Uses PLocal storage type.
  - `pyorient.STORAGE_TYPE_MEMORY` Uses Memory storage type.

Only the database name is required. By default the method creates a Document Database using the PLocal storage type.

Example

Say that your application collects and analyzes data from various custom built smart home devices installed around the house. When it first runs it finds that it needs to initialize a database on OrientDB to store the data it collects.

```
try:
   client.db_create(
      "tinkerhome",
       pyorient.DB_TYPE_GRAPH,
       pyorient.STORAGE_TYPE_PLOCAL)
   logging.info("TinkerHome Database Created.")
except pyorient.PYORIENT_EXCEPTION as err:
   logging.critical(
      "Failed to create TinkerHome DB: %"
      % err)
```

Here, PyOrient attempts to create the database `tinkerhome` on OrientDB. If the create command fails, it logs it as a critical error.

PyOrient Client - `db_drop()`

This method removes a database from the connected OrientDB server.

Dropping Databases

In certain situations, you may want to remove a full database from the OrientDB Server. For instance, if you create a temporary database in memory for certain operations or if you want to provide the user with the ability to uninstall the database with the application, without removing OrientDB itself.

Syntax

```
client.db_drop(<db-name>)
```

- `<db-name>` Defines the database to remove.

Example

Consider the example of the smart home database. In developing your application, you may want to frequently reset the database to a clean state. This allows you to move forward with a new implementation without having to worry about an obsolete schema causing problems for you.

```
# Reset Database
def reset_db(client, name):

   # Remove Old Database
   client.db_drop(name)

   # Create New Database
   client.db_create(name,
      pyorient.DB_TYPE_GRAPH,
      pyorient.STORAGE_TYPE_PLOCAL)
```

Here, the function receives the client and the database name. It deletes the given database and creates a new one with the same name.

PyOrient Client - `db_exists()`

This method checks whether or not the given database exists on the connected OrientDB Server. In the event that you need the database to exist and be of a specific storage type, you can pass a second argument to check this as well.

Checking Database Existence

There are two methods available in determining whether a database exists on the OrientDB Server. You can use the `db_list()` method to find all the databases on the server, then search the results or, in the event that you know the database name, you can check it directly with `db_exists()`.

Syntax

```
client.db_exists(<name>, <storage-type>)
```

- **`<name>`** Defines the database you want to find.
- **`<storage-type>`** Defines the storage type you want to find, (optional):
  - *`pyorient.STORAGE_TYPE_PLOCAL`* Checks for PLocal database.
  - *`pyorient.STORAGE_TYPE_MEMORY`* Checks for Memory database.

By default, it searches for a PLocal database type.

Example

Consider the example of the database for your smart home application. Since the application requires a database to operate, you might use this method as a basic check when the application starts. For instance,

```
# Check Database
if client.db_exists("tinkerhome"):
   # Open Database
   client.db_open("tinkerhome", "admin", "admin_passwd")

else:
   # Create Database
   client.db_create(
      "tinkerhome",
      pyorient.DB_TYPE_GRAPH,
      pyorient.STORAGE_TYPE_PLOCAL
   )
```

Here, the application checks if the `tinkerhome` database exists. If it does, it opens the database on the client. If it doesn't, it creates a PLocal Graph Database on that name.

PyOrient Client - `db_list()`

This method lists the databases available on the connected OrientDB Server.

Listing Databases

There are two methods available in determining what databases are available on the OrientDB Server. The first is check whether an individual database is present, using `db_exists()`. In the event that you don't know the database name or you want to operate on all databases, you can use `db_list()` to fetch information on all databases on the server.

Syntax

```
client.db_list()
```

Example

When you create a database without specifying the username and password, OrientDB defaults to the user `admin` and the password `admin`. While this behavior is fine during development, it poses certain security risks in production. You might use `db_list()` as part of a basic security check to ensure that all databases on your server have non-default login credentials. For instance,

```
# List Databases
database_list = client.db_list().__getattr__('databases')

# Check for Default Login
for i in database_list:
   try:
      client.db_open(i, "admin", "admin")
      print("Default Credentials found on: %s" % i)
      client.db_close()
   except:
      print("Non-default Credentials found on: %s" % i)
```

Here, PyOrient creates a dict object, where the key is the database name and the value the database path. The `__getattr__()` method ensures that you retrieve a dict instead of an OrientRecord object. You can test this using the Python console,

```
>>> client.db_list()
<pyorient.otypes.OrientRecord object at 0x7f89c4211e10>

>>> client.db_list().__getattr__('databases')
{'tinkerhome': 'plocal:/data/tinkerhome',
'GratefulDeadConcerts': 'plocal:/data/GratefulDeadConcerts'}
```

In the code, once this is set PyOrient then loops through the dict, checking each key, (that is, each database name), to test the default login credentials. It then prints whether the login succeeds or fails to stdout. This lets you know which databases need updated credentials.

PyOrient Client - `db_open()`

This method opens a database on the OrientDB Server.

Opening Databases

When you have the name of the database that you want to use, as well as the relevant authentication credentials, you can open this database within your client.

Syntax

```
client.db_open(<name>, <username>, <user-passwd>, <db-type>, <client-id>)
```

- **`<name>`** Defines the database you want to open.
- **`<username>`** Defines the database user name.
- **`<user-passwd>`** Defines the database user password.
- **`<db-type>`** Defines the database type, (optional).
  - *`pyorient.DB_TYPE_DOCUMENT`* Opens it as a Document Database.
  - *`pyorient.DB_TYPE_GRAPH`* Opens it as a Graph Database.
- **`<client-id>`** With distributed deployments, use this argument to define the distributed node.

Example

In the case of the example smart home database, say that after various sanity checks you're ready to start working on the database itself. You might use a line like the one below to open the database, so that you can start using it:

```
client.db_open("tinkerhome", "nodeuser", "node_passwd")
```

This opens the `tinkerhome` database on the PyOrient Client. Commands issued through this object now operate on that database.

PyOrient Client - `db_reload()`

This method reloads the open database, refreshing cluster information from OrientDB.

Reloading Databases

On occasion, you may encounter or implement deployments where multiple clients are interacting with the OrientDB server at a given time. In cases such as this, you may need to reload the database, updating cluster information on the `client` object.

Syntax

```
client.db_reload()
```

Example

For instance, consider the example of the smart home application database. You have a web interface that provides control functionality and a back-end systems that update the database with information taken from physical devices. Periodically, you may need the web interface to reload its client connection.

```
class WebInterface():

   def __init__(self, client):
      self.client = client

   def reload(self):
      self.client.db_reload()
```

Here, the class receives the PyOrient Client and initializes it as a class variable. You can then periodically call the method periodically to reload the database on the client, updating cluster information.

PyOrient Client - `db_size()`

This method returns the size of the open database.

Getting the Database Size

Using the `db_size()` method, you can determine the size of a given database. It does not receive any arguments, but rather checks the database currently open on the client. To open a database, see `db_open()`.

Syntax

```
client.db_size()
```

Example

You might use `db_size()` as a sanity check in your application. For instance, you might have your application check the database size in order to determine whether or not it has been properly initialized:

```
client.db_open("tinkerhome", "admin", "admin_passwd")

# Fetch Size
size = client.db_size()

assert size != 0
```

Here, the application opens the `tinkerhome` database, then checks it's size. It uses an `assert` statement to ensure that the database is not empty.

PyOrient Client - `get_session_token()`

In certain use cases, you may want to maintain a client connection across several sessions. For instance, in a web application you might set an identifier for a shopping cart or use sessions to maintain a local history of the user's interactions with the site.

Beginning in version 27, PyOrient provides support for token-based sessions. Using the `get_session_token()` method, your client can fetch a session ID from the client, which you can then use in reattaching to old sessions.

Getting Session Tokens

Retrieving session tokens from OrientDB is a process closely tied with setting them on the client using the `set_session_token()` method. In order to fetch the session token, you first need to enable the feature through this method.

Syntax

```
client.get_session_token()
```

The method returns either the current session token or, in the event that the feature is disabled, an empty string.

Enabling Session Tokens

Working with session tokens requires that you configure the OrientDB Server to use them. You can manage this through the `set_session_token()`. When you pass this parameter a token it reattaches to that session. When you pass it a boolean value, it enables or disables the feature.

```
# Initialize the Client
client = pyorient.OrientDB("localhost", 2424)

# Enable Token-based Authentication
client.set_session_token(True)
```

Retrieving Session Tokens

Once you have enabled this feature, you can fetch a token for the current session and set it to the variable.

```
# Fetch Session Token
sessionToken = client.get_session_token()
```

In the event that something went wrong or if for some reason session tokens are disabled on OrientDB when you call this method, it returns an empty string. To avoid problems later, you can implement a check using an `assert` statement to ensure that the token was properly set:

```
# Ensure that Server allows Session Tokens
assert sessionToken != ''
```

With the session token set on the variable `sessionToken`, you can reattach to your current session later using the `set_session_token()` method.

PyOrient Client - `query()`

This method issues a synchronous query to the open OrientDB database.

Querying Records

In the event that you're more comfortable working in SQL, you can query the OrientDB Server directly from within your PyOrient application using the `query()` method. This method operates similar to the `record_load()`, `record_update()` and `create_record()` methods, depending on the SQL you use.

> In the event that you want to make an asynchronous query, see the `query_async()` method.

Syntax

```
client.query(<query>, <limit>, <fetch-plan>)
```

- **`<query>`** Defines the SQL you want to send to OrientDB.
- **`<limit>`** Defines the number of results you want returned, defaults to all results.
- **`<fetch-plan>`** Defines the fetching strategy you want to use.

Example

Consider the example of the smart home database, where your application is logging readings from various Arduino or Micro Python sensors to OrientDB. In building the web interface for this system, say that you want to pull all pollen data recorded to the database.

```
data = client.query("SELECT FROM Sensors "
                    "WHERE sensorType = 'Pollen'",
                    100)
```

Here, your application queries the `Sensors` class for all records that pertain to pollen sensors. It then sets the return value to the `data` object. You can now pass this object to a Django or Flask method in rendering it to HTML to use with the web interface.

Querying Records with Fetch Plans

In OrientDB, Fetching Strategies allow you to refine the process the given client uses in retrieving data from the server. You may find this useful when retrieving a record that contains several layers of linked records. With a normal query, the client would need to resubmit the request for each additional layer of linked records. Using a fetch plan, OrientDB with collect all connecting records and send them to PyOrient together.

For instance, in the example of a smart home database say that you want to retrieve all records from sensors in your bedroom. In the event that you have trouble sleeping, this might help in determining the cause of the problem, allowing you to say connect room temperature or light conditions with your waking up in the middle of the night.

```
data = client.query("SELECT FROM Sensors "
                    "WHERE room = 'bedroom'",
                    100, "*:-1")
```

Here, the fetching strategy, (given as the third argument) retrieves all connecting records to the bedroom sensors. It sets the return data to an object, which you can then pass to other methods for analysis or presentation. You can fine tune how many layers of connecting records OrientDB collects. For more information, see Fetching Strategies.

PyOrient Client - `query_async()`

This method is runs queries against the open OrientDB database, and runs the given callback function for each record the query returns.

Querying the Database

When you issue a query using the standard `query()` method, PyOrient issues the query to OrientDB and then waits for the database to return its result-set. In the case of particularly long running queries, you may want your application to perform some additional checks as it collects records from the query.

Using the `query_async()` method, each individual record PyOrient receives from OrientDB triggers the given callback function or method.

When working with the standard `query()` method, PyOrient issues the query to OrientDB then sets the subsequent result-set as its return value. In the time between the issuing of the query and receiving the return value, the application waits. For most use cases, given OrientDB's performance, this behavior is fine. But, in the case of particularly long running queries, you may find it undesirable.

In the event that you would like to perform some additional checks or operations while the query runs, you can use the `query_async()` method. This allows you to perform addition operations on the data as it processes, such as logging or initiating certain calculations in advance of the final result.

Syntax

```
client.query_async(<query>, <limit>, <fetch-plan>, <callback>)
```

- **`<query>`** Defines the SQL query.
- **`<limit>`** Defines a limit for the result-set.
- **`<fetch-plan>`** Defines a Fetching Strategy
- **`<callback>`** Defines the callback function or method.

Example

Consider the example of a smart home system that uses OrientDB for back-end storage. Say that your application logs data on various environmental sensors. Given a few dozen sensors and a set of records added at least every fifteen minutes, after a few months this can grow into a very large database.

For instance, say that you have a class used in analyzing data for a web front-end. Using `query_async()` you can call a logging method to report on the records you've retrieved before setting the return value.

```
# Data Analyzer Class
class DataAnalyzer():

   # Initialize Class
   def __init__(self, client, nodetype):

      # Init Class Variables
      self.client = client
      self.record_count = 0

      # Retrieve Data
      query = "SELECT FROM Nodes WHERE nodetype = '%s' % nodetype
      self.data = self.client.query_async(query, "*:-1", self.log_query)

      # Report Count
      logging.info("Records Retrieved: %s" % self.record_count)

   # Log Record Returns
   def log_query(self, record):
       logging.debug("Loading Record: %s" % record._rid)
       self.record_count += 1
```

Here, when you initialize the `DataAnalyzer` class, you pass to it the PyOrient client and a string indicating the type of sensor nodes you want to analyzie. The class queries OrientDB for data on these node types. For each record `query_async()` retrieves, it calls the `log_query()` class method.

For each record retrieved, it calls the logging module to create a debugging event. It also increments the internal counter `record_count`. Once the query is complete, it calls logging again to issue an informational event reporting the number of records the query retrieved.

PyOrient Client - `record_create()`

This method creates a record on the open OrientDB database, using the given dict.

> **Warning**: Prior to version 2.0 of OrientDB, some users encountered issues with `record_create()` and `record_update()` in PyOrient. When developing applications for older versions of OrientDB, it is recommended that you avoid these methods.

Creating Records

When you want to add a completely new record to the database, use the `record_create()` method. This method takes two arguments, the ID of the cluster you want to add the data to, and a dict object that represents the data you want to add.

Syntax

```
client.record_create(<cluster_id>, <data>)
```

- **`<cluster_id>`** Integer that defnes the cluster to use.
- **`<data>`** Dict that defines the data to add.

The dict object you pass to this method uses the following format:

```
data = {
   @<class>: {
      <property>: <value>
   }
}
```

- **`<class>`** Defines the class name.
- **`<property>`** Defines the property you want to add, if any.
- **`<value>`** Defines the value you want to assign the property.

Example

For instance, consider the use case of a smart home database. Your application is connected to several basic security devices set up around the exterior doors to your house. Whenever someone approaches one of these doors, they trigger a motion sensor that initializes the event.

When setting up a system like this, you might want to record the event with webcams, log whether the visitor rang the doorbell, knocked or did nothing, and determine whether the interior or exterior locks were used in openning the door. Once the application finishes recording the event, it needs to log the data to OrientDB, (such as an offsite server, in the event of burglary).

```
event = {
   @DoorSecurityEvent: {
      "time_start": timestamp_start,
      "time_end": timestamp_end,
      "webcam_file": "/path/to/file.ogg,
      "doorbell": True,
      "knock": False,
      "open": True,
      "open_type": "InnerLock"
   },
}
client.record_create(cluster_id, event)
```

Here, your application records a postal service delivery. It sets the cluster ID, as well as the starting and ending timestamp variables as they occur, then builds the dict object, then it passes the Cluster ID and the dict object to the `record_cluster()`, to log the delivery to OrientDB.

PyOrient Client - `record_delete()`

This method removes records from the open OrientDB database.

Deleting Records

When you want to remove records from the open OrientDB database, you can do so using the `record_delete()` method. To do so, you'll need the cluster that contains the record and its Record ID.

Syntax

```
client.record_delete(<cluster-id>, <record-id>)
```

- **`<cluster-id>`** Defines the cluster that contains the record.
- **`<record-id>`** Defines the record to remove.

Example

Consider the example of the smart home database. In developing the web interface, you want to implement a feature that allows the user to delete sensor nodes from the database. In the event that you find certain nodes are damaged or you move the system to a new house and would like the monitors to display fresh sensor data.

```
def remove_records(client, cluster_id record_list):

   # Iterate through Record ID's
   for record in record_list:

      # Delete Record
      client.record_delete(cluster_id, record.__rid)
```

Here, the function receives the client, cluster ID and an array of records as arguments. It then iterates over each record in the array and deletes it using the `record_delete()` method.

PyOrient Client - `record_load()`

This method retrieves the given Record ID from an open OrientDB database.

Loading Records

In PyOrient, a record load is similar to a `query()` in that it fetches record from the database for the application to operate on. But, unlike a query, `record_load()` fetches a record using its Record ID.

Syntax

```
client.record_load(<rid>, <fetch-plan>, <callback>)
```

- **`<rid>`** Defines the Record ID of the record you want to load.
- **`<fetch-plan>`** Defines a Fetching Strategy to use.
- **`<callback>`** Defines a function to use as callback with your Fetch Plan.

Example

For instance, consider the use case of an application that manages a series of smart devices scattered about your home. You log data from these devices to OrientDB.

Given that your application is logging all this data, you might want to extract it to use elsewhere. With an array of Record ID's, you can iterate through, using this method to retrieve the data from OrientDB and pass it on as a return value. For example,

```
# Process Data
def compile_data(client, rid_array):

   # Initialize Variable
   data = {}

   # Iterate through Record ID's
   for rid in rid_array:

      # Log Data
      data[rid] = client.record_load(rid)

   # Return Data
   return data
```

Here, the function receives the `client` interface and array of Record ID's, it iterates through them, storing the data in the `data` dict, which it then returns for analysis elsewhere in your application.

Loading Records with Cache

In addition to the standard record loads, where PyOrient queries the database and returns the single requested record, you can also implement a Fetch Plan through this method. When you need to traverse connected records, this feature allows you to take advantage of early loads, ensuring that the client only needs to make one request, rather than sending a series of additional requests for the connected records.

Consider the example of a smart home database. In each room of your house you have installed a series of Arduino- or Micro Python-based devices to record environmental conditions, such as light and noise levels, pollen count, smoke and carbon monoxide detectors and so on. Every fifteen minutes your application checks in with these devices and takes an average of their readings, logging its findings to OrientDB.

In building a web interface to serve as a control panel, you might want to extract this data by room or zone to generate graphs illustrating how conditions have changed over time. For example, you might find it useful to combine the senor readings for your bedroom with a sleep monitor.

```
zone = Zone('bedroom')

client.db_record_load('#14:33', '*:-1', zone.add_sensors)
```

Here, we initialize a Python `Zone` class, which we've defined elsewhere as a bedroom instance. Once this is done, we ask PyOrient to all records associated with #14:33, (that is, the master bedroom). PyOrient then passes the results to the `add_sensors()` method defined on the `Zone` class. Once this is complete, we can use the `zone` class object in rendering a report for the control panel.

PyOrient Client - `record_update()`

This method updates the given Record ID with data from a dict argument.

> **Warning**: Prior to version 2.0 of OrientDB, some users encountered issues with `record_update()` and `record_create()` in PyOrient. When developing applications for older versions of OrientDB, it is recommended that you avoid these methods.

Updating Records

When you want to update an existing record on OrientDB, you can use the `record_update()` method. In order to update a record, you'll need its Record ID and version, which you can call from the record object.

Syntax

```
client.record_update(<record-id>, <data>, <version>)
```

- **`<record-id>`** Defines the Record ID of the record you want to update.
- **`<data>`** Defines the data you want to change in a dict.
- **`<versin>`** Defines the version of the record you want to update.

The dict object you pass to this method uses the following format:

```
data = {
   @<class>: {
      <property>: <value>
   }
}
```

- **`<class>`** Defines the class name.
- **`<property>`** Defines the property you want to add, if any.
- **`<value>`** Defines the value you want to assign the property.

Example

Consider the example of a database for a smart home application. Say that you have a series of appliances that you have modified with your own custom electronics. For instance, you might add a Raspberry Pi with a touchscreen to the refrigerator, to display weather information while idle and provide an app for generating grocery lists.

Whenever you open the grocery list, your application checks in with OrientDB for the current shopping list, then updates the database with the new entries. Then, say you have guests coming over and want to buy more eggs:

```
# Fetch Grocery List
eggs = client.query(
   "SELECT FROM ShoppingList "
   "WHERE type = 'grocery'"
   "AND item = 'eggs'")

data = {
   "@ShoppingList": {
      "quantity": 24
   }
}
client.record_update(eggs._rid, data, eggs._version)
```

Here, PyOrient uses the `query()` method to retrieve an object instance from the database, it then updates the `ShoppingList` with the new quantity of eggs.

PyOrient Client - `set_session_token()`

In certain use cases, you may want to maintain a client connection across several sessions. For instance, in a web application you might set an identifier for a shopping cart or use sessions to maintain a local history of the user's interactions with the site.

Beginning in version 27, PyOrient provides support for token-based sessions. Using the `set_session_token()` method, your client can tokens and, using one, reattach to an existing session.

Setting Session Tokens

There are two operations possible with this method. It determines which by the type of the value you pass it as an argument.

Syntax

```
client.set_session_token(<enable|session-id>)
```

- **`<enable>`** When the method receives a boolean value, it enables or disables token-based authentication in OrientDB.
- **`<session-id>`** When the method receives the identifier for an existing session, it reattaches to that session.

In practice, you need to use both in your application. First to enable token-based authentication and then to reattach to an existing session. When this feature is enabled, you can fetch the current session token using the `get_session_token()` method.

Enabling Session Tokens

Working with session tokens requires that you configure the OrientDB Server to use them. You can manage this through the `set_session_token()`. When you pass this parameter a token it reattaches to that session. When you pass it a boolean value, it enables or disables the feature.

```
# Initialize the Client
client = pyorient.OrientDB("localhost", 2424)

# Enable Token-based Authentication
client.set_session_token(True)
```

Once you have enabled this feature, you can fetch a token for the current session and set it to the variable, using the `get_session_token()` method.

```
# Fetch Session Token
sessionToken = client.get_session_token()
```

With the session token fetched and stored on the variable, `sessionToken`, you now have it available to use later in reattaching to this session.

Setting Session Tokens

With session tokens enabled and one fetched and set to a variable using the `get_session_token()` method, you can use the token to reattach to an existing session.

```
# Reattach to Session
if sessionToken != '':
   client.set_session_token(sessionToken)
else:
   # Open Database
   client.db_open("tinkerhome", "admin", "admin_passwd")

   # Enable Session Tokens
   client.set_session_token(True)

   # Fetch Session Token
   sessionToken = client.get_session_token()
   assert sessionToken != ''

# Query Database
record = client.query(
   "SELECT status FROM Nodes WHERE zone = 'living-room'")
```

Here, your application first checks whether the `sessionToken` variable is an empty string. If it finds that it is not an empty string, it uses the variable to reattach to an existing session. If it finds that it is an empty string, it opens the database, enables session tokens, to fetches the current session token to set it on the `sessionToken` variable for later. Afterwards, it queries the database for the status of all devices in the living room.

> In the above example note, when the client reattaches to an existing session, it doesn't need to reopen the database, given that the database is already open on that session.

PyOrient Client - `tx_commit()`

This method initializes a transaction control object.

Working with Transactions

Transactions represent units of work executed against the database, which are treated in a coherent and reliable way, independent of other transactions. They provide units of work, allowing for failure recovery and keeping the database consistent even in cases of system failure. They also provide isolation in cases where multiple clients concurrently access the database.

In PyOrient, the `tx_commit()` client method initializes a transaction control object. Using the control object, you can begin and operate on the transaction itself, controlling what gets added and when the application commits the changes.

Syntax

```
client.tx_commit()
```

Examples

Consider the example of a smart home system that uses OrientDB for its back-end storage. Say that you have a web application that runs on a touchscreen device built into or near the refrigerator. This application allows you to create and update grocery lists. Other applications can then call up the grocery list to automatically send SMS reminders about things family members need to pick up on their next trip to the supermarket.

The grocery list application handles updates using transactions, beginning the transaction when the user edits the list and committing it after they save. For instance, your application may build a transaction update like this,

```
# Define Cluster
cluster_id = 3

#  Create Record for Base Objects
record_base = {
   'store': "Davis Square Farmers' Market",
   'list': [],
}
record_position = client.record_create(cluster_id, record_base)

# Begin Transaction
tx = client.tx_commit()
tx.begin()


# Create Record in Transaction
record_1 = {
   'store': "McKinnon's Meat Market",
   'list': ['steak', 'eggs', 'sausage']
}
new_record = client.record_create(-1, record_1)

# Update Existing Record
record_2 = {
   'store': "Davis Square Farmers' Market",
   'list': ['garlic, 'green pepper', 'onion', 'celery'],
}
update_record = client.record_update(
      cluster_id,
      record_position._rid,
      record_2,
      record_position._version
)

# Attach Operations to Transaction
tx.attach(new_record)
tx.attach(update_record)

# Commit the Transaction
results = tx.commit()

assert results["#3:1"].store == "Davis Square Farmers' Market"
assert results["#3:2"].store == "McKinnon's Meat Market"
```

PyOrient Transactions - `attach()`

This method attaches operations to particular transactions.

Attaching Operations

When you initialize a transaction through the `tx_commit()` client method, the return object provides you with several methods used in controlling the transaction process. Using the `attach()` method, you can associate various client operations with the transaction.

You can then call `commit()` to apply the changes to OrientDB. In the event that you aren't satisfied with the changes, you can revert the database to an earlier state by calling `rollback()` .

Syntax

```
tx.attach(<record>)
```

- **`<record>`** Defines the PyOrient return object for a database operation you want to add to the transaction. For instance, the return object for the `record_create()` or `record_update` client methods.

Example

Consider the example of a smart home system that uses OrientDB for back-end storage. Say that your application uses the database to store configuration options, such as how often to take readings from environmental sensors or where it should stream the video feed from security cameras.

```
# Prepare Update
updated_config = {
"sensorReadInterval": 15
}
update = client.record_update(cluster_id, '#2:1', updated_config)

# Attach to Transaction
tx.attach(update)
```

Here, the configuration record, (that is, #2:1), is updated with a new value assigned to the `sensorInterval` property. When you commit the transaction, it updates the interval on environmental sensors to take readings every fifteen minutes.

PyOrient Transactions - `begin()`

This method initiates a transaction.

Beginning Transactions

When you initialize a transaction through the `tx_commit()` client method, the return object provides you with several methods used in controlling the transaction process. Using the `begin()` method, you can initiate a transaction on this object.

Beginning a transactions effectively saves the database state. In the event that there is a problem with operations made after this point, you can revert the database back to this point using the `rollback()` method. Alternatively, you can commit the changes to the database using the `commit()` method.

Syntax

```
tx.begin()
```

Example

For instance, say for a web application you want to create a series of new records on the database through a transaction. This ensures that, in the event that something goes wrong in the process, you can revert the database to its earlier state rather than committing an incomplete operation.

```
# Initialize Transaction Control Object
tx = client.tx_commit()

# Begin Transaction
tx.begin()

try:
   # Create Records
   for record in records:
      new = client.record_create(cluster_id, record)
      tx.attach(new)
except:
   tx.rollback()

# Commit Changes
tx.commit()
```

Here, your application has an array of records stored in the `records` object. It initializes a transaction object with the `tx_commit()` client method, then begins the transaction with `begin()`. Within a `try` statement, it loops through this array, using the `record_create()` method to create new records. After it creates the record, it attaches the operations to the open transaction.

In the event that there is a problem, Python executes the `except` statement, which calls `rollback()`, reverting the database to its earlier state. If it's able to create all the records in the array, it issues the `commit()` method to commit the changes to the database.

PyOrient Transactions - `commit()`

This method commits the open transaction.

Committing Transactions

When you initialize a transaction through the `tx_commit()` client method, the return object provides you with several methods used in controlling the transaction process. Using the `commit()` method, you close the transaction committing all modifications to the database.

Syntax

```
tx.commit()
```

Example

For instance, say for a web application you want to create a series of new records on the database through a transaction. This ensures that, in the event that something goes wrong in the process, you can revert the database to its earlier state rather than committing an incomplete operation.

```
# Initialize Transaction Control Object
tx = client.tx_commit()

# Begin Transaction
tx.begin()

try:
   # Create Records
   for record in records:
      new = client.record_create(cluster_id, record)
      tx.attach(new)
except:
   tx.rollback()

# Commit Changes
tx.commit()
```

Here, your application has an array of records stored in the `records` object. It initializes a transaction object with the `tx_commit()` client method, then begins the transaction with `begin()`. Within a `try` statement, it loops through this array, using the `record_create()` method to create new records. After it creates the record, it attaches the operations to the open transaction.

In the event that there is a problem, Python executes the `except` statement, which calls `rollback()`, reverting the database to its earlier state. If it's able to create all the records in the array, it issues the `commit()` method to commit the changes to the database.

PyOrient Transactions = `rollback()`

This method rolls the database back to an earlier state, removing any changes made on the open transaction.

Rolling Back Transactions

When you initialize a transaction through the `tx_commit()` client method, the return object provides you with several methods used in controlling the transaction process. Using the `rollback()` method, you can close the transaction, reverting all associated changes made to the database.

Example

For instance, say for a web application you want to create a series of new records on the database through a transaction. This ensures that, in the event that something goes wrong in the process, you can revert the database to its earlier state rather than committing an incomplete operation.

```
# Initialize Transaction Control Object
tx = client.tx_commit()

# Begin Transaction
tx.begin()

try:
   # Create Records
   for record in records:
      new = client.record_create(cluster_id, record)
      tx.attach(new)
except:
   tx.rollback()

# Commit Changes
tx.commit()
```

Here, your application has an array of records stored in the `records` object. It initializes a transaction object with the `tx_commit()` client method, then begins the transaction with `begin()`. Within a `try` statement, it loops through this array, using the `record_create()` method to create new records. After it creates the record, it attaches the operations to the open transaction.

In the event that there is a problem, Python executes the `except` statement, which calls `rollback()`, reverting the database to its earlier state. If it's able to create all the records in the array, it issues the `commit()` method to commit the changes to the database.

PyOrient OGM

Where the PyOrient Client is a wrapper for the Binary Protocol, the Object-Graph Mapper provides a higher-level object-oriented Pythonic interface for Graph databases in OrientDB. It is comparable to the use of ORM's with Relational databases.

The purpose of the OGM is to make interactions with large and complex Graph databases more understandable and easier to maintain. It bridges the gap between higher level object-oriented concepts in your application and the vertices and edges in your database.

Understanding the OGM

What the OGM does is that it maps Python objects to classes and properties in OrientDB. Your application can then operate on objects as it would normally, with PyOrient operating on the database in the background.

- When you have an existing OrientDB database schema, the PyOrient OGM can map the schema classes to Python classes in your application.
- When you have an existing Python application, the PyOrient OGM can build a database schema in OrientDB from the Python classes in your application.

> Whichever method you choose to adopt, once you have built the database and mapped its schema to your Python classes, you can execute queries against it. You can do this using the OrientDB console or from within your application.
>
> Currently, the mapper does not support `TRAVERSE` and it's support for Gremlin is functional, but limited.

Using the OGM

- **Database Connections**
- **Working with Schemas**
- **Working with Brokers**
- **Batch Transactions**
- **Scripting**

PyOrient OGM - Connection

In order to use the Object-Graph Mapper with your application, you first need to connect to a running OrientDB Server. In PyOrient, there are two interface classes used in this process and you can access them through an `import` statement:

```
# Module Imports
from pyorient.ogm import Graph, Config
```

- **`pyorient.ogm.Graph`** Wraps the lower-level PyOrient Client, that is, `pyorient.OrientDB()` and adds to it support for mapping Python classes to OrientDB database schemas and database schemas to Python Classes.
- **`pyorient.ogm.Config`** Provides an interface for defining how the OGM connects to OrientDB.

Connecting to OrientDB

When the PyOrient OGM connects to OrientDB, it uses the `pyorient.ogm.Config` class to define the specific database and credentials it uses in establishing the connection.

Syntax

```
Config.from_url(
      <database-url>,
      <user>,
      <password>,
      initial_drop = False)
```

- **`<database-url>`** Defines the database.
- **`<user>`** Defines the username.
- **`<password>`** Defines the user password.
- **`initial_drop`** Defines whether PyOrient should drop any existing databases that share this configuration.

For the database URL, you have the option of using one of several supported formats:

- Connecting to a PLocal database:
  - `localhost/<database-name>`
  - `plocal://localhost:<port>/<database-name>`
- Connecting to a Memory database:
  - `<database-name>`
  - `memory://localhost:<port>/<database-name>`

Example

For instance, say that you have a smart home system written in Python that uses OrientDB for back-end database storage. You might use something like this to set up the database connection for your application:

```
# Connect to Database
Config.from_url(
   'plocal://localhost:2424/smarthome',
   'root', 'root_passwd')
```

Connecting with Graph Object

In addition to basic connection described above, you can also pass the connection configuration when you initialize the `Graph` object. For instance,

```
graph = Graph(
      Config.from_url(
         'localhost/smarthome',
     'root', 'root_passwd'))
```

This initializes a `graph` instance of the `pyorient.ogm.Graph` class and defines how you want the OGM to connect to OrientDB. You can then use `graph` in your applications to access further PyOrient OGM methods.

PyOrient OGM - Schemas

By definition, an OGM maps objects in your application to classes in a Graph Database. In doing so, it defines a schema in the database to match classes and sub-classes in the application.

Building Schemas from Classes

PyOrient only maps classes to the database schema that belong to the registry on your PyOrient `Graph` object. There are two types of registries, one indicating a vertex, (or node), and the other an edge, (or relationship). Adding Python classes to the registries is handled through subclassing. For instance,

```
from pyorient.ogm import declarative

# Initialize Registries
Node = declarative.declarative_node()
Relationship = declarative.declarative_relationship()

# Create Vertex Class
class Person(Node):
   pass

# Create Edge Class
class Likes(Relationship):
   pass
```

Each call made to the `declarative_node()` and `declarative_relationship()` methods creates a new registry. The OGM preserves inheritance hierarchies of nodes and relationships. This registry is now accessible through `Node.registry` and `Relationship.registry` .

To create the corresponding classes in the OrientDB database schema, pass the registries to the `Graph` object, using the `create_all()` method. For instance,

```
# Initialize Schema
self.g.create_all(Node.registry)
self.g.create_all(Relationship.registry)
```

In the event that these classes already exist in your database schema, and you want to bind the Python objects too them rather than creating the schema anew, use the `include` method.

```
# Bind Schema
self.g.include(Node.registry)
self.g.include(Relationship.registry)
```

Building Classes from Schemas

Currently, PyOrient does not have a tool for automatically generating Python code from a database schema, but it come close. Using the `build_mapping()` method, you can generate a dictionary of Python classes, (that is, a registry). You can then pass the dictionary to the `include()` method on the PyOrient `Graph` class.

For instance,

```
from pyorient.ogm.declarative import declarative_node, declarative_relationship

# Initial Schema Objects
SchemaNode = declarative_node()
SchemaRelationship = delcarative_relationship()

# Retrive Schema from OrientDB
classes_from_schema = graph.build_mapping(
   SchemaNode,
   SchemaRelationship,
   auto_plural = True)

# Initialize Schema in PyOrient
graph.include(classes_from_schema)
```

Here, dynamically generated vertex classes have `SchemaNode` as their the top-level of their inheritance, while edge classes have `SchemaRelationship`. By setting the `auto_plural` parameter to `True`, the subsequent `include()` method automatically assigns brokers to `graph` object.

> In the event that you would like to use custom names for your brokers, (such as, brokers that use the actual plural nouns), you need to perform some kind of post processing on the dict returned to `classes_from_schema`. For instance, iterating through the broker names and adding "`s`" to the end of the string.

PyOrient OGM - Brokers

The mapping that the PyOrient OGM creates between Python classes and the OrientDB database schema happens on a few different levels. In Schemas you can see how it forms connections between classes in your application and database. Brokers handle the mapping between objects in a given Python class and the specific vertices and edges in the OrientDB database.

Using Brokers

When the OGM creates a mapping between Python classes and vertex or edges classes in the OrientDB schema, for each class it creates a broker to handle instances of that class. The broker provides you with an interface in working with various types of vertices and edges in your graph. They can also reduce coupling and hide the classes that you have mapped, to help focus on the interfaces that they expose.

For instance, consider the example of a smart home application that uses OrientDB as its back-end database. You might create a class in your application and on OrientDB to manage home-built Arduino or Micro Python environmental sensors.

```
# Initialize Graph Database
graph = pyorient.ogm.Graph(
   pyorient.ogm.Config.from_url(
      'smarthome', 'root', 'root'))
Node = pyorient.ogm.declarative.declarative_node()

# Sensor Control Class
class Sensor(Node):
   element_plural = 'sensors'
   name = String()
   zone = String()

# Create a 'sensors' bject and set the 'objects' attribute
graph.include(Node.registry)
```

In the example, you create a Python class for your sensors and register it with the OGM. Given that you already have sensor data on OrientDB, you use the `include()` method to map the registry to a vertex class in OrientDB.

When you create classes that map to vertex and edge classes in OrientDB, you must define a particular attribute: `element_plural` for a node and `label` for a relationship. PyOrient uses this class variable in setting the broker attribute for the class. Meaning that, in the example above, `Sensor.objects` and `graph.sensors` are synonymous. Thus,

```
# Create Smoke Alarm (Method 1)
g.sensors.create(
   name = 'smokeAlarm',
   zone = 'kitchen')

# Create Light Sensor (Method 2)
Sensor.objects.create(
   name = 'lightSensor',
   zone = 'kitchen')
```

Creating vertex classes without `element_plural` attribute or edge classes without the `label` attribute, cause PyOrient to not create a broker. Meaning, in such cases, the first method shown above won't work.

You can then query the results using the `query()` method:

```
results = graph.query(Sensor, name = 'smokeAlarm')
```

Manual Vertex and Edge Creation

The `create()` method show above hides whether you are creating a vertex or an edge in OrientDB. Using the manual create methods `create_vertex()` and `create_edge()`, you can optionally make your application more explicit. For instance,

```
graph.create_vertex(
   Sensor,
   name = 'smokeAlarm',
   zone = 'kitchen')
```

Working with Properties

In the example above, when the OGM maps the Python `Sensor` class to OrientDB, attributes on that class map to properties on the vertex, (that is, `name` and `zone`). OrientDB and PyOrient support a range of property types. Some basic examples are,

Numeric Types	Other
Boolean	String
Byte	Date
Integer	DateTime
Short	Binary
Long	Embedded
Float	
Double	
Decimal	

> For more information on types supported by OrientDB, see Types.

PyOrient OGM - Batch Operations

The Object-Graph Mapper provides basic support for transactions. This allows you to group several operations together and execute them in a batch on the OrientDB database. You may find this useful in ensuring concurrency as well as reducing the round-trip time on the given operation.

Using Batch Transactions

Transactions in PyOrient use a similar workflow to transactions in OrientDB. That is, you start by beginning the transaction, perform various operations as part of it, then commit those changes to the database.

Initializing Batch Transactions

Unlike standard operations with the OGM, batch transactions operate on a separate object from the OGM `Graph` class. Instead, you use the `batch()` method to initialize a control object for the transaction. You can then direct operations on OrientDB through this object, only committing it to OrientDB when you're ready.

```
# Initialize Batch Transaction
batch = graph.batch()
```

Performing Operations

Once you've initialized the control object through the `batch()` method, you can begin building the transaction. For instance, consider the example of a smart home application that uses OrientDB for back-end storage. While home built systems often have the database server on premises, let's say that you expect some homes to rely on a remote database accessed through the internet.

In situations like this, database initialization can prove difficult, since it would involve multiple network requests to OrientDB. Instead you can group this process into a batch transaction handled through a single network operation.

Initialize Classes

In preparing a batch operation to initialize the database, you might start by initializing the classes that you want to create:

```
# Initialize Sensor Class
class Sensor(Node):
   element_type = 'sensor'
   element_plural = 'sensors'

   name = String(multibyte = False, unique = True)
   node_type = String(multibyte = False)

# Initialize Zone Class
class Zone(Node):
   element_type = 'zone'
   element_plural = 'zones'

   name = String(multibyte = False, unique = True)
   zone_type = String(multibyte = False)

# Initialize Position Class
class SensorPosition(Relationship):
   label = 'position'
```

Here, you create three classes: two vertex classes for sensors and zones in the house, and an edge class for to position the sensors in particular zones. The special attribute `element_type` is redundant here, but it allows you to tell the mapper what name to use for the corresponding schema class.

Creating Vertices

With the classes created, you can create particular instances of these classes, then adding them to the batch transaction:

```
# Create Pollen Sensor
batch['pollen-sensor-1452'] = batch.sensors.create(
   name = 'ARDUINO-UNO-1432',
   node_type = 'pollen')

# Create Light Sensor
batch['light-sensor-259'] = batch.sensors.create(
   name = 'ARDUINO-UNO-259',
   node_type = 'light-sensor')

# Create Bedroom Zone
batch['master-bedroom'] = batch.zones.create(
   name = 'master',
   zone_type = 'bedroom'
```

This creates three vertices in the batch: one for an Arduino device built as a pollen sensor, one for a light sensor, and one for a zone for the master bedroom.

Creating Edges

With the vertices created, you can also create edges to connect them. In doing so, you have the option of two syntax methods for the operation.

```
# Place Pollen Sensor (Method 1)
batch[:] = batch.position.create(
   batch[:'pollen-sensor-1452'],
   batch[:'master-bedroom']).retry(20)

# Place Light Sensor (Method 2)
batch[:] = batch[:'light-sensor-259'](SensorPosition) > batch[:'master-bedroom']
```

In the example, notice that in both cases the code defines the relationship using slice syntax. You can do this in cases where you don't need to reference the edge later in the batch transaction.

Return Values

In the event that you would like to perform further operations on the code before committing the transaction, batches provide an optional return value. You can retrieve this value from the control object:

```
# Fetch Pollen Sensor Return Value
pollen = batch['$pollen-sensor-1452']
```

This retrieves the return value and sets it on the `pollen` variable.

Committing Batch Transactions

When you're finished with the batch transaction, you can commit it to OrientDB using the `commit()` method.

```
# Cmmmit Batch Transaction
batch.commit()
```

When you issue this method, PyOrient sends a network request to OrientDB, then executes each operation in the batch through that single request, without the need for the usual back and forth between the two.

PyOrient OGM - Scripts

In addition to the Client and the standard OGM methods shown above, you can also operate on the OrientDB database. This provides you with basic support for Gremlin graph traversal, through Groovy scripts.

Working with Scripts

In order to use scripts in your application, you first need to add them to the OGM `Graph` class. Once added, you can call them from within your application, using the `gremlin()` method.

For instance, say that you want to add a set of Groovy scripts from the `scripts/` directory. You might use something like this to do so:

```
# Module Imports
import pathlib
from pyorient.groovy import GroovyScripts

# Iterate through scripts/
for path in pathlib.Path('scripts/').iterdir():

   # Check if Grovvy Script
   if path.is_file() and path.suffix == '.groovy':

      # Add Script
      graph.scripts.add(GroovyScripts.from_file(str(path)))
```

Here, your application uses the `pathlib` module to iterate over each file and directory in the `scripts/` directory. For each instance, it checks whether the instance is a file with the `.groovy` extension. It then parses out functions found in these files and adds them to the `scripts` attribute.

Using Scripts

Scripts that you add to the `scripts` attribute are callable through the `gremlin()` method. For the example of the smart home application, let's say that you have a `compile_data()` function that compiles sensor data in building charts for the web interface. You give it the type of sensor you want to read and it iterates through each zone as data points for the charts.

```
pollen_chart = graph.gremlin('compile_data', 'pollen')
```

This runs the `compile_data()` function, passing the string `pollen` to it, to indicate that you would like to compile data from pollen sensors. It then returns the results to the `pollen_chart` variable.

> **NOTE**: In the event that you need namespacing, both the `add()` and `gremlin()` methods allow you to set `namespace` arguments. Using this, you can separate different functions with the same name.

OrientDB-NET

OrientDB provides driver support through a network binary protocol. This allows you to manage servers through various API's and drivers. You can access the database through a C#/.NET application using the OrientDB-NET.binary driver.

> The OrientDB .NET SDK is currently being rewritten as a modular SDK. If you would like to check it out and help drive the direction of the new version, see OrientDB.NET.Core.

Installation

The OrientDB-NET driver is not included by default in your OrientDB installation. In order to use it, you must install it separately on your system.

> For information on how to install OrientDB itself, see Installing OrientDB.

Installing from NuGet

NuGet is a package manager available for C#/.NET development environments. It is accessible through Visual Studio and through the `nuget` command-line interface on Windows, macOS and on Linux through Mono.

The package name is `OrientDB-Net.binary.Innov8tive`

- Using the package management console in Visual Studio, run the following command:

  ```
  PM> Install-Package OrientDB-Net.binary.Innov8tive
  ```

- Using the NuGet command-line application, run the following command from your project directory:

  ```
  $ nuget install OrientDB-Net-Innov8tive
  ```

SendFailure Errors

While most C#/.NET applications run on Microsoft Windows, it is now also possible to develop them to work with Linux and FreeBSD.

When using NuGet for the first time on these operating systems, you may encounter errors whenever it attempts to connect to the configured software repositories. For instance,

```
$ nuget install OrientDB-Net.binary.Innov8tive
WARNING: SendFailure (Error writing headers)
WARNING: An error occured while loading packages from
'https://www.nuget.org/api/v2': Error SendFailure
(Error writing headers)
Unable to find package 'OrientDB-NET.binary'
```

These errors occur in new installations of Mono or in similar cases where NuGet cannot locate the certificates it requires to securely download packages from Microsoft and the NuGet repositories.

You can download these certificates using the Mono Certificate Manager, through the following commands:

```
# certmgr -ssl -m https://go.microsoft.com
# certmgr -ssl -m https://nugetgallery.blob.core.windows.net
# certmgr -ssl -m https://nuget.org
```

You can now install packages through NuGet.

Using OrientDB-NET

Once you have installed OrientDB and the OrientDB-NET.binary driver, you can begin to develop your C#/.NET application. In order to utilize OrientDB-NET functions and classes, set the relevant namespace with the `using` directive.

```
using Orient.Client;
```

- **Server** Documents the `OServer` interface, used when operating on the OrientDB Server to manage server-level configuration as well as creating, removing and listing databases on the server.
- **Database** Documents the `ODatabase` interface, used when operating on particular OrientDB databases to manage clusters, operate on records and issue scripts, queries and commands.
- **Queries** Provides a guide to building queries.
- **Transaction** Documents the `OTransaction` interface, used when operating on OrientDB databases through transactions.

OrientDB-NET - `OServer`

This class provides an interface and methods for when you need to operate on the OrientDB Server from within your C#/.NET application.

Use this interface in cases where you need to retrieve or modify server configuration, create or remove databases or fetch information about those databases available on the server. If you want to operate on a specific database, use the `ODatabase` interface.

Initializing OServer

When you enable the `Orient.Client` namespace through the `using` directive, you can create a server interface for your application by instantiating the `OServer` class.

Syntax

```
OServer(    string hostName,
            int port,
            string userName,
            string userPasswd)
```

- **`hostname`** Defines the host that you want to connect to, such as `localhost` or the IP address on which OrientDB is running.
- **`port`** Defines the port you want to connect to, such as 2424.
- **`userName`** Defines the Server user name.
- **`userPasswd`** Defines the Server user password.

Once you have an instance of this class, you can begin to call methods on it to operate on the OrientDB Server.

Example

In the interest of abstraction, you might create a class with methods to handle common OrientDB Server operations. For instance, say you want a method that creates and returns a new `OServer` instance,

```
public static class Server
{
   private static string _hostname  = "localhost";
   private static int _port         = 2424;
   private static string _user      = "root";
   private static string _passwd    = "root_passwd";

   public static OServer Connect()
   {
      server = new OServer(_hostname, _port,
            _root, _root_passwd);
      return server;
   }

}
```

With this `Server` class, you can use the `Connect()` method to retrieve a new instance of `OServer`. You can then use this instance in various database operations.

Using OServer

Once you have created an instance of `OServer` within your application, you can use its methods in performing various operations on databases, including,

Method	Return Value	Description
`ConfigGet()`	`string`	Retrieves a configuration value for the given variable.
`ConfigList()`	`Dictionary<string, string>`	Retrieves the complete server configuration.
`ConfigSet()`	`bool`	Modifies the given configuration variable.
`CreateDatabase()`	`bool`	Creates a database on the server.
`Close()`	`void`	Disposes of the OServer connection.
`DatabaseExists()`	`bool`	Checks that database exists on server.
`Databases()`	`Dictionary<string, ODatabaseInfo>`	Returns information on all databases on server.
`Dispose()`	`void`	Disposes of the OServer connection.
`DropDatabase()`	`void`	Removes the given database.

Closing OServer

When you are finished using an instance server interface, you can close it to free up system resources. The `OServer` class provides two methods to close a server connection.

- `Close()`
- `Dispose()`

Given one is an alias, you can use whichever you find most familiar. For instance, say that you have creating an `OServer` instance in your application and set it on the `server` object. To dispose of this instance, call one of the above methods on that object.

```
// Close Server
server.Close();
```

OrientDB-NET - `ConfigGet()`

This method retrieves the value of the given configuration variable from the server. It returns the value as a string.

Retrieving Configuration Values

In certain situations you may want to retrieve values from various configuration variables from the OrientDB Server. You might find this useful when checking various settings to determine whether the server is ready for use by your application.

Syntax

```
string OServer.ConfigGet(string key)
```

- `key` Defines the configuration variable that you want to check.

This method returns a string of the current setting.

Example

For instance, say that you are developing a basic unit test to evaluate whether the file-type is properly set for the transaction log. You want the file-type set to classic-mode, but are worried that some databases in your distributed cluster may have the wrong configuration.

```
// FETCH TX.LOG FILETYPE
string txFileType = server.ConfigGet("tx.log.fileType");

Assert.AreEqual("classic", txFileType);
```

OrientDB-NET - `ConfigList()`

This method returns the current configuration of the OrientDB Server as a dictionary of key/value pairs.

Retrieving Server Configuration

In situations where you need to check multiple configuration variables on a server, you may find it beneficial to retrieve the entire server configuration set in one call rather than making multiple individual requests.

Syntax

```
Dictionary<string, string> OServer.ConfigList()
```

The return value is a dictionary of configuration variables and their current values.

Example

When debugging your application, you may sometimes find it useful to consult the OrientDB Server configuration, to help in checking whether the server was set up properly for your usage.

Consider the use-case where you are working with OrientDB in a distributed cluster with dozens of servers running in-memory. Whenever you add servers to your infrastructure, you need to evaluate several configuration variables before moving it to production.

```
using Orient.Client;
using System;
...

// WRITE SERVER CONFIGURATION TO CONSOLE
public Dictionary<string, string> ReportConfig(OServer server)
{
   // FETCH SERVER CONFIGURATION
   Dictionary<string, string> srvConfig = server.ConfigList();

   // WRITE HEADER
   Console.WriteLine("OrientDB Server Configuration");

   // LOOP OVER EACH CONFIG ENTRY
   foreach(KeyValuePair<string, string> entry in srvConfig)
   {

      // WRITE TO CONSOLE
      Console.WriteLine(" - {0}: {1}",
         entry.Key,
         entry.Value);
   }

   // RETURN CONFIGURATION FOR ADDITIONAL TESTING
   return srvConfig;
}
```

Here, the function receives an `OServer` interface as an argument. Using this interface, it retrieves the current server configuration and loops over it, printing each variable and value to the console. When it's done it returns the configuration dictionary, which you can then use to perform additional tests instead of making multiple calls to `ConfigGet()`.

OrientDB-NET - `ConfigSet()`

This method changes the value of the given configuration variable on the OrientDB Server. It returns a boolean value indicating whether the change was successful.

Setting Configuration Variables

You may find that you want to change the values on various configuration variables for the OrientDB Server from within your application. With this method you can do so through the `OServer` instance.

Syntax

```
bool OServer.ConfigSet(
   string key,
   string value)
```

- **`key`** Defines the configuration variable you want to change.
- **`value`** Defines the value you want to set on the variable.

This method returns a boolean value, indicating whether OrientDB applied your change.

Example

For instance, say that you have multiple OrientDB servers running in a distributed deployment. You notice certain performance issues relating to network connectivity and decide to begin optimizing the `network.retry` parameter by changing its current setting to 6.

```
// UPDATE network.retry PARAMETER
bool IsUpdated = server.ConfigSet("network.retry", 6);
Assert.IsTrue(IsUpdated);
```

OrientDB-NET - `CreateDatabase()`

This method creates a database on the connected OrientDB Server. It then returns a boolean value to indicate that the operation was successful.

Creating Databases

When you initialize your C#/.NET application, you may find it useful to provision an OrientDB database as part of the installation process. This ensures that you'll have a database ready when you first run the application. You can create databases on the OrientDB Server by calling the `CreateDatabase()` method on the `OServer` interface.

Syntax

```
bool OServer.CreateDatabase(
   string name,
   ODatabaseType type,
   OStorageType storage)
```

- **`name`** Defines the name of the database.
- **`type`** Defines the type of database you want to create, that is a Graph, Document or Object Database.
- **`storage`** Defines the type of storage you want to use. That is, physical or in-memory.

When the operation is complete, the method returns a boolean value indicated that the new database now exists.

Examples

For instance, imagine an application that utilizes a series of in-memory databases for various services that you want to provide. You might construction a method such as this to use when provisioning new servers:

```
using Orient.Client;
using System;
...

// PROVISION ORIENTDB SERVER
public void InitServer(OServer server, string[] names)
{
   // LOG OPERATION
   Console.WriteLine("Creating Databases:");

   // LOOP OVER DATABASE NAMES
   foreach(string name in names)
   {
      // CREATE DATABASE
      bool dbCheck = server.CreateDatabase(name,
         ODatabaseType.Graph,
         OStorageType.Memory);

      // REPORT CREATION
      if(dbCheck)
      {
         Console.WriteLine(" - SUCCESS: {0}", name);
      }
      else
      {
         Console.WriteLine(" - FAILURE: {0}", name);
      }
   }
}
```

OrientDB-NET - `DatabaseExists()`

This method determines whether or not a database exists already on the OrientDB Server. It returns a boolean value indicating what it finds.

Checking Databases

In order to check that databases exist on the server, you first need to create an `OServer` instance. Once you have it, you can call the `DatabaseExists()` method on it.

Syntax

```
bool OServer.DatabaseExists(
   string <name>,
   OStorageType <storage>)
```

- `<name>` Defines the name of the database you want.
- `<storage>` Defines the storage type you want, such as PLocal or Memory.

The method returns a boolean value, indicating whether or not it found the requested database on the server.

Example

For instance, say that you have a complex application that utilizes several databases in-memory on an OrientDB Server. The in-memory storage type is volatile and is lost in the event that the server shuts down or the host crashes. As such, you may want to create a basic test function to determine whether a series of databases exist on the server before you attempt operations. If the database doesn't exist, you'll need to create it.

```
using Orient.Client;
using System;
...

// CHECK THAT DATABASES EXIST
public void checkDatabases(OServer server, string[] databases)
{
   Console.WriteLine("Checking that databases exists...");

   // LOOP OVER EACH REQUIRED DATABASE
   foreach(string database in databases)
   {
      // DETERMINE IF DATABASE EXISTS
      bool dbExists = server.DatabaseExists(database,
         OStorageType.Memory);

      // CREATE DATABASE
      if(dbExists == false)
      {
         Console.WriteLine("Database {0} doesn't exist, creating...",
            database);
         // CREATE NONEXISTENT DATABASE
         server.CreateDatabase(database,
            ODatabaseType.Graph,
            OStorageType.Memory);
      }

      // REPORT IF DATABASE EXISTS
      else
      {
          Console.WriteLine("Database {0} exists already",
             database);
      }
   }
}
```

OrientDB-NET - `Databases()`

This method retrieves information on the current databases available on the OrientDB Server. The return value is a dictionary of string keys and `ODatabaseInfo` objects.

Retrieving Database Information

In certain situations you may need to operate on multiple databases on a given OrientDB Server, such as testing certain conditions on each database available. When the given databases are arbitrary, (that is, the names are not fixed by variables within your application), you can retrieve all databases on the given `OServer` instance using this method.

Syntax

```
Dictionary<string, ODatabaseInfo> OServer.Databases()
```

This method takes no arguments. The return value is a dictionary with string value keys and `ODatabaseInfo` object values. Each `ODatabaseInfo` object has three variables:

Variable	Type	Description
`DataBaseName`	`string`	The database name
`StorageType`	`OStorageType`	The storage type, (PLocal or Memory)
`Path`	`string`	The database URL

Examples

Consider the use case of a long running OrientDB Server that has be utilized by several applications, including yours. You intend to move your operation onto a fresh host, but want to back up those databases that were not part of your application.

```
using Orient.Client;
using System;
...

// CLEANUP SERVER
public void CleanServer(OServer server, string[] databaseNames,
   string host, int, port, string user, string userPasswd,
   string backupPath)
{
   // REPORT OPERATION
   Console.WriteLine("Cleaning OrientDB Server...");

   // FETCH DATABASE INFORMATION
   Dictionary<string, ODatabaseInfo> srvInfo = server.Databases();

   // LOOP OVER EACH DATABASE INSTANCE
   foreach(KeyValuePair<string, ODatabaseInfo> dbInfo in srvInfo)
   {
      string name = entry.Value.DataBaseName;

      // TEST IF DBNAME IN DATABASENAMES
      if(databaseNames.Contains(name))
      {
         // OPEN DATABASE
         ODatabase database = ODatabase(host, port, name,
            ODatabaseType.Graph, user, passwd);

         // FORMAT BACKUP COMMAND
         string backupCommand = String.Format(
            "BACKUP DATABASE {0}/{1}.zip -compressionLevel=2",
            backupPath, name);

         // RUN COMMAND
         database.Command(backupCommand);

         // CLOSE DATABASE
         database.Close();
      }
   }
}
```

OrientDB-NET - `DropDatabase()`

This method removes a database from the OrientDB Server. It returns no value.

Removing Databases

To remove a database from the server, you need to call the `DropDatabase()` method on the `OServer` instance.

Syntax

```
void OServer.DropDatabase(
   string <name>,
   OStorageType <storage>)
```

- `<name>` Defines the name of the database you want to remove.
- `<storage>` Defines the storage type of the database.

Example

For instance, say you have an application that requires you to periodically reset databases, removing all records then initializing a new instance.

```
public void cleanDatabases(OServer server, string[] databaseNames)
{
   // LOOP OVER DATABASE NAMES ARRAY
   foreach(string name in databaseNames)
   {
      // REMOVE EXISTING DATABASE
      server.DropDatabase(name, OStorageType.Memory);

      // CREATE NEW DATABASE
      bool createdDb = server.CreateDatabase(name,
         ODatabaseType.Graph, OStorageType.Memory);

      // TEST
      Assert.IsTrue(createDb);
   }
}
```

OrientDB-NET - `ODatabase`

This class provides an interface and methods for when you need to operate on OrientDB databases from within your C#/.NET application.

Use this interface in cases where you need to retrieve or manage clusters, operate on records or issue scripts, queries and commands to the database. If you want to operate on the OrientDB Server as a whole or create or delete databases on the server, use the `OServer` interface.

Initializing the Database

When you enable the `Orient.Client` namespace through the `using` directive, you can create a database interface for your application by instantiating the `ODatabase` class.

Syntax

```
// INITIALIZE DATABASE
ODatabase(  string host,
            int port,
            string name,
            ODatabaseType type,
            string user,
            string passwd)

// INITIALIZE DATABASE USING CONNECTION POOL
ODatabase(  string host,
            int port,
            string name,
            ODatabaseType type,
            string user,
            string passwd,
            string poolAlias)
```

- **`host`** Defines the hostname or IP address of the OrientDB Server.
- **`port`** Defines the port to use in connecting to the server.
- **`name`** Defines the database name.
- **`type`** Defines the database type, that is PLocal or Memory.
- **`user`** Defines the user name.
- **`passwd`** Defines the user password.
- **`poolAlias`** Defines the alias to use for the connection pool.

Example

In the interest of abstraction, you might create a method to handle common OrientDB database operations for your application.

```
using Orient.Client;
using System;
...

// OPEN DATABASE
public ODatabase openDatabase(string _host, int _port,
       string _dbName, string _user, string _passwd)
{

   // CONSOLE LOG
   Console.WriteLine("Opening Database: {0}", _dbname);

   // OPEN DATABASE
   ODatabase database = ODatabase(_host, _port, _dbName,
       ODatabaseType.Graph, _user, _passwd);

   // RETURN ODATABASE INSTANCE
   return database;
}
```

Using a Connection Pool

Normally, OrientDB-NET clients operate through a single network connection. When working with web applications or any situation where network bottlenecks are a concern, it is common to pool these connections to ensure better performance.

To use a connection pool pass a pool alias when you create the `ODatabase` interface.

```
ODatabase database = ODatabase("localhost", 2424,
   "microblog", ODatabaseType.PLocal,
   "guestUser", "guest_passwd",
   "pool123");
```

Using Connection Options

In addition to configuring the database interface through arguments passed to the `ODatabase` class, you can also create a configuration object independent of the class, through the `ConnectionOptions` class. You may find this useful when you need to initialize several database instances with similar configuration.

The `ConnectionOptions` class has seven parameters:

Parameter	Type	Description
`HostName`	`string`	Defines the hostname or IP address of the server hosting OrientDB
`UserName`	`string`	Defines the name of the database user
`Password`	`string`	Defines the password for database user
`Port`	`int`	Defines the port number for the connection
`DatabaseName`	`string`	Defines the name of the database to use
`DatabaseType`	`ODatabaseType`	Defnes the type of database, PLocal or Memory
`PoolAlias`	`string`	Defines the connection pool to use.

If you initialize this object without defining the connection pool, it sets it to `Default` .

```
// INITIALIZE CONNECTION OPTIONS
ConnectionOptions opts = ConnectionOptions();

opts.HostName = "localhost";
opts.UserName = "admin";
opts.Password = "admin_passwd";
opts.Port = 2727;
opts.DatabaseName = "microblog";
opts.DatabaseType = ODatabaseType.PLocal

// Initialize Database
ODatabase database = ODatabase(opts);
```

Using ODatabase

Once you have instantiated the `ODatabase` class, you can begin to operate on a particular database from within your C#/.NET application. OrientDB-NET supports a number of database-level operations that you can call through this interface. Additionally, you can issue queries, commands and scripts to the database for those operations that it does not support.

Method	Return Value	Description
`Close()`	`void`	Disposes of the database instance.
`Clusters()`	`OClusterQuery`	Creates clusters.
`Command()`	`OCommandResult`	Issues commands.
`CountRecords`	`long`	Counts records on database.
`Dispose()`	`void`	Disposes of the database instance.
`GetClusterIdFor()`	`short`	Retrieves ID for the given cluster name.
`GetClusterNameFor()`	`string`	Retrieves the name for the given Cluster ID.
`GetClusters()`	`List<OCluster>`	Retrieves clusters from database.
`Gremlin()`	`OCommandResult`	Executes Gremlin commands.
`Insert()`	`IOInsert`	Prepares insertion operations.
`JavaScript()`	`OCommamndQuery`	Prepares JavaScript commands.
`Query()`	`List<ODocument>`	Queries the database using SQL.
`Select()`	`OSqlSelect`	Prepares queries to execute.
`Size`	`long`	Retrieves the size of the database.
`SqlBatch()`	`OCommandQuery`	Executes batch queries.
`Update()`	`OSqlUpdate`	Prepares update statements.

Size Variables

In addition to the various methods, the `ODatabase` object also supports sizing variables, which you can use to determine the size of the database or the number of records it contains.

- To retrieve the size of the database, use the `Size` variable:

  ```
  long databaseSize = database.Size;
  ```

- To count the number of records on the database, use the `CountRecords` variable:

  ```
  long numberOfRecords = database.CountRecords;
  ```

Closing Databases

When you are finished using the database instance, you can close it to free up system resources. The `ODatabase` object provides two methods for closing databases: `Close()` and `Dispose()` . Given that one is an alias to the other, you can use whichever is more famiiliar to you.

```
// CLOSE DATABSE
database.Close()
```

Transactions

OrientDB provides support for transactions. The Insert(), Update() and Query() methods all provide support so that you can execute them as part of a transaction rather than on the database outright. For more information, see `OTransaction` .

OrientDB-NET - `Clusters()`

This method adds clusters to the OrientDB database. The return value is an `OClusterQuery` object.

Creating Clusters

To create clusters on the database, you need to call the `Clusters()` method on the `ODatabase` interface. In order to do so, you need to pass it an array of cluster names or ID's that you want to add to the database.

Syntax

```
// CREATING CLUSTERS BY NAME
OClusterQuery ODatabase.Clusters(params string[] clusterNames)

// CREATING CLUSTERS BY ID
OClusterQuery ODatabase.Clusters(params short[] clusterId)
```

- **`cluster-names`** Defines a series of strings (`params string[]`) indicating the names of the clusters you want to create.
- **`cluster-ids`** Defines a series of numbers (`params short[]`) indicating the ID's of the clusters you want to create.

Additional Methods

When you use this method, the return value is an `OClusterQuery` object, which provides additional methods that you may find useful:

- `Count()` This method returns a long value of the created clusters.
- `Range()` This method returns an `ODocument` object of the created clusters.

Examples

For instance, you might use this method in building a wrapper function to add clusters to the database. Using the wrapper you can extend the method with additional logic to log information to the console or perform further operations on the newly created clusters.

```
using Orient.Client;
using System;
...

public void createCluster(ODatabase database,
      params string[] clusters)
{
   // LOG TO CONSOLE
   Console.Write("Creating Clusters: {0}",
      String.Join(", ", clusters));

   // CREATE CLUSTERS
   OClusterQuery clusterQuery = database.Clusters(clusters);

   // FETCH COUNT
   long count = clusterQuery.Count();

   // LOG TO CONSOLE
   Console.Write("Created {0} clusters", count);
}
```

Similarly, you might want to create an arbitrary range of clusters by passing a `param` of short values to the same method.

```
// INITIALIZE CLUSTER ID's
params short[] clusterIds = [1, 2, 3, 4, 5]

// CREATE CLUSTERS
OClusterQuery query = database.Clusters(clusterIds);
```

OrientDB-NET - `Command()`

This method prepares or executes a command on the OrientDB database. The return value is an `OCommandResult` object.

Sending Commands

There are several methods available in issuing queries and commands to OrientDB through your C# application. This method allows you to issue SQL commands to the database.

For information on available commands, see SQL and Console commands.

Syntax

```
// EXECUTING COMMANDS
OCommandResult Command(     string <query>)
```

- `<query>` Defines an SQL statement to execute.
- `<command>` Defines a prepared command object.

Example

For instance, consider the use case of making internal data persistent across multiple operations. When your application is running, it operates a dictionary object. When it closes, it uploads data from the dictionary to OrientDB, so that it will have it ready when you run the app again.

```
using Orient.Client;
using System;

// SAVE DATA
public void Save(ODatabase database, Dictionary<string, string> data)
{
   // LOG OPERATION
   Console.WriteLine("Saving Data to OrientDB");

   // LOOP OVER DATA
   foreach(KeyValuePair<string, string> entry in data)
   {
      // LOG OPERATION
      Console.WriteLine(" - Saving: {0}", entry.Key);

      // BUILD QUERY
      string sqlQuery = String.Format("UPDATE Save SET {0} = {1}",
         entry.Key, entry.Value);

      // RUN COMMAND
      database.Command(sqlQuery);
   }
}
```

Here, the application loops over the dictionary, running an `UPDATE` statement for each variable in the data dictionary.

OrientDB-NET - `GetClusterIdFor()`

This method retrieves the default Cluster ID for a given class. The return value is a `short`.

Retrieving Cluster ID's

While cluster names are easier for people to understand and keep track of, you may occasionally find it more efficient and performant to work with cluster ID's in these operations. Using the `GetClusterIdFor()` method, you can retrieve the short ID for a given cluster.

It is comparable to the `GetClusterNameFor()` method.

Syntax

```
short ODatabase.GetClusterIdFor(  string <name>)
```

- `<name>` Defines the class name.

Example

For instance, if you prefer operating on clusters by their ID's rather than names, you may find it convenient to create a helper function to retrieve the Id's for a group of cluster names.

```
using Orient.Client;
using System;
...

// FETCH CLUSTER ID'S
public List<short> fetchClusterIds(ODatabase database, string[] clusterNames)
{
   // LOG OPERATION
   Console.WriteLine("Retrieving ID's for clusters: {0}",
      String.Join(", ", clusterNames));

   // INITIALIZE RETURN VARIABLE
   List<short> clusterIDs;

   // LOOP OVER NAMES
   foreach(string name in clusterNames)
   {
      // FETCH CLUSTER ID
      short clusterID = database.GetClusterIdFor(name);

      // APPEND ID TO RETURN VALUE
      clusterIDs.Add(clusterID)
   }

   // RETURN CLUSTER ID'S
   return clusterIDs;
}
```

OrientDB-NET - `GetClusterNameFor()`

This method retrieves the name of a cluster for the given Cluster ID. The return value is a string.

Retrieving Cluster Names

While Cluster ID's may prove easier and more performant for your application to operate on, their meaning and purpose is not always clear for the programmer or when writing to logs. To retrieve the cluster name from an ID, use the `GetClusterNameFor()` method on the `ODatabase` interface.

It is comparable to the `GetClusterIdFor()` method.

Syntax

```
string ODatabase.GetClusterNameFor(short <cluster-id>)
```

- **`<cluster-id>`** Defines the Cluster ID you want to query.

Example

For instance, if you prefer operating on clusters by their ID's rather than names, you may find it convenient to create a helper function to retrieve cluster names from a list of cluster ID's.

```
using Orient.Client;
using System;
...

// FETCH CLUSTER NMAES
public List<string> fetchClusterNames(ODatabase database, short[] clusterIDs)
{
   // LOG OPERATION
   Console.WriteLine("Retrieve Names for Clusters: {0}",
      String.Join(", ", clusterIDs));

  // INITIALIZE RETRUN VALUE
  List<string> clusterNames;

  // LOOP OVER ID'S
  foreach(short clusterID in clusterIDs)
  {
    // FETCH CLUSTER NAME
    string clusterName = database.GetClusterNameFor(clusterID);

    // APPEND NAME TO RETURN VALUE
    clusterNames.Add(clusterName);
  }

  // RETURN CLUSTER NAMES
  return clusterNames;
}
```

OrientDB-NET - `GetClusters()`

This method returns the clusters on the connected OrientDB database. The return value is a list of `OCluster` objects.

Retrieving Clusters

In cases where you need to operate on many, most or all clusters in a database, you may find it more efficient to retrieve all `OCluster` objects in a single call. You can do so using the `GetClusters()` method.

Syntax

```
List<OCluster> ODatabase.GetClusters(bool reload)
```

- **`reload`** Defines whether you want to reload the `ODatabase` instance before retrieving the clusters. Defaults to `false`.

Examples

For instance, as part of a logging operation, you might build a helper function to retrieve the available clusters from the database and to print their names to the console, before returning them for further operations.

```
using Orient.Client;
using System;

public List<OCluster> FetchClusters(ODatabase database,
    bool reload = false)
{
  // FETCH CLUSTERS
  List<OCluster> clusters = database.GetClusters(reload);

  // INITIALIZE CLUSTER NAMES LIST
  List<string> clusterNames;
  foreach(OCluster cluster in clusters)
  {
     // ADD CLUSTER NAME
     clusterNames.Add(cluster.Name);
  }

  // LOG TO CONSOLE
  Console.WriteLine("Retrieved Clusters: {0}",
    String.Join(', ', clusterNames));

  // RETURN CLUSTERS
  return clusters;
}
```

OrientDB-NET - `Gremlin()`

This method executes Gremlin scripts. The return value is an `OCommandResult` object.

Executing Gremlin Scripts

In cases where you have existing scripts or would prefer to operate on OrientDB using the Gremlin language, you can do so through `ODatabase` interface, using the `Gremlin()` method.

Syntax

```
OCommandResult Gremlin(string <query>)
```

- **`<query>`** Defines the command to execute

Example

In situations where you prefer to using Gremlin scripts or would like to use features implemented in Gremlin but which are not yet available to OrientDB-NET, you can call these internally from a string.

```
using Orient.Client;
using System;
...

// RETRIEVE VERTICES THROUGH GREMLIN
public OCommandResult fetchVertices(ODatabase database, string databaseName,
    string className)
{
  // LOG OPERATION
  Console.WriteLine("Gremlin Query: All Vertices in: {0}",
     className);

  // INITIALIZE SCRIPT
  string script = String.Format("
    g = new OrientGraph('plocal:/data/{0}');
    vertices = g.{1};
    g.close();
    return vertices;",
    databaseName, className);

  // EXECUTE SCRIPT
  OCommandResultSet resultSet = database.Gremlin(script);

  // RETURN RESULTS
  return resultSet;
}
```

In addition to building your Gremlin scripts within your application as strings, you can also retrieve the script from file. You may find this particularly useful in cases where you have a body of routine Gremlin scripts already prepared for your application, or when you want to work with developers who are familiar with JavaScript and Gremlin, but somewhat less so with the C#/.NET framework.

```
using Orient.Client;
using System;
...

// RUN GREMLIN SCRIPT FROM FILE
public OCommandResult runScript(ODatabase database, string path)
{
  // LOG OPERATION
  Console.WriteLine("Running Gremlin Script: {0}", path);

  // RETRIEVE SCRIPT
  string script = IO.File.ReadAllText(path);

  // RUN SCRIPT
  OCommandResult resultSet = database.Gremlin(script);

  // RETURN RESULT-SET
  return resultSet;
}
```

OrientDB-NET - `Insert()`

This method inserts records into the database.

Inserting Data

Using this method you can insert records into the database. By itself, it initializes an `IOInsert` object, which you can that operate on to further define the data you want to insert.

Syntax

```
IOInsert ODatabase.Insert()
   .Into(class)
   .Set(field, value)

IOInsert ODatabase.Insert()
   .Cluster(cluster)
   .Set(field, value)
```

- **`class`** Defines the class to use.
- **`cluster`** Defines the cluster to use.
- **`field`** Defines the field to set.
- **`value`** Defines the value to set on the field.

The above methods allow you to build the `IOInsert` object. You can then execute a processing command to run the query against the database. There are two such methods available to you,

- **`Run()`** Executes the insertion on the database and returns an `ODocument` object.
- **`ToString()`** Executes the insertion on the database and returns a string of the added record.

Example

For instance, say that you are developing an accounting application in C# and want to support migration. You receive a CSV file from a spreadsheet application and want to insert its records into OrientDB.

```
using Orient.Client;
using (TextFieldParser parser = new TextFieldParser("$HOME/2016-report.csv"))
{
   // INITIALIZE DATABASE
   ODatabase database = ODatabase("localhost", 2424, "account-app",
      ODatabaseType.PLocal, "user", "passwd");

   // INITIALIZE PARSER
   parser.TextFieldType = FieldType.Delmited;
   parser.SetDelmiters(",");

   // MIGRATE DATA
   while (!parser.EndOfData)
   {
      // INSERT ROW
      string[] fields = parser.ReadFields();
            ODocument test = database.Insert()
                 .Into("Account")
                 .Set("name", field[0])
         .Set("contact", field[1])
         .Set("status", field[2])
         .Run();
   }
}
```

OrientDB-NET - `JavaScript()`

This method prepares JavaScript queries to execute on the OrientDB database. The return value is an `OCommandQuery` object.

Querying with JavaScript

In cases where you have database operations scripted in JavaScript, you can execute these through OrientDB-NET using the `JavaScript()` method.

Syntax

```
OCommandQuery JavaScript(string <query>)
```

- **`<query>`** Defines the query to execute.

Example

In cases situations where you prefer to operate on the database using JavaScript or would like to use features available through JavaScript but which are not yet available with OrientDB-NET, you can use this method to execute JavaScript from a string.

```
using Orient.Client;
using System;

// RETRIEVE RECORDS FROM GIVEN CLASS
public OCommandResult FetchAllRecords(ODatabase database,
    string dbName, string className)
{
  // LOG OPERATION
  Console.WriteLine("Retrieving All Records: {0}",
    className);

  // CONSTRUCT SCRIPT
  string script = "
    var db = new ODatabase('http://localhost:2480/{0}');
    dbInfo = db.open();
    queryResult = db.Query('SELECT FROM {1}');
    db.close();
    return queryResult;",
    dbName, className);

  return database.JavaScript(script).Run();
}
```

In addition to building your JavaScript scripts from within your application as strings, you can also retrieve scripts from file. You may find this particularly useful inc ases where you have a body of routine JavaScript operations already prepared for your application, or when you want to work with developers who are familiar with JavaScript, but somewhat less so with the C#/.NET framework.

```
using Orient.Client;
using System;
...

// RUN JAVASCRIPT FILE
public OCommandResult JSQuery(ODatabase database, string filename)
{
  // LOG OPERATION
  Console.WriteLine("Run File: {0}", filename);

  // RETRIEVE SCRIPT
  string script = IO.File.ReadAllText(path);

  // RUN SCRIPT
  return database.JavaScript(script).Run();

}
```

OrientDB-NET - `Query()`

This method issues SQL queries against the OrientDB database. It returns a list of `ODocument` objects from the result-set.

Querying the Database

In some cases you may find features in OrientDB that are not yet available through OrientDB-NET. You can utilize these features by passing SQL statements for them through the `Query()` method. It returns a list of `ODocument` objects that you can operate on further.

It is comparable to the `Command()` method.

Syntax

```
// QUERY DATABASE
List<ODocument> ODatabase.Query(string SQL)

// QUERY DATABASE WITH FETCH PLAN
List<ODocument> ODatabase.Query(string SQL,
   string fetch-plan)
```

- **`SQL`** Defines the SQL statement to use.
- **`fetch-plan`** Defines the Fetching Strategy to use.

Example

In situations where you execute the same or very similar queries with some frequency or in cases where you need to run a query that has no comparable function available in OrientDB-NET, you can issue the SQL statement manually through this menthod.

```
using Orient.Client;
using System;
...

// FETCH MATCHING DOCUMENTS FROM CLASS
public List<ODocument> FetchRecords(ODatabase database,
    string className, Dictionary<string, string> conditions)
{
  // LOG OPERATION
  Console.WriteLine("Querying Class: {0}", className);

  // BUILD QUERY
  List<string> baseQuery = [
    String.Format('SELECT FROM {0}', className)];

  // CHECK FOR CONDITIONAL VALUES
  if(conditions.Count > 0)
  {
    // ADD WHERE
    baseQuery.Add('WHERE');

    // ADD CONDITIONS
    foreach(KeyValuePair<string, string> condition in conditions)
    {
      string entry = String.Format("{0}={1}",
        condition.Key, condition.Value);

      baseQuery.Add(entry);
    }

    // JOIN QUERY
    string query = String.Join(' ', baseQuery);

    // RUN QUERY
    return database.Query(query);
  }
}
```

In the event that you would like to execute the query with a fetching strategy, you can do so through the second argument.

```
using Orient.Client;
using System;

// FETCH ALL RECORDS WITH FETCHING STRATEGY
public List<ODocument> FetchAll(ODatabase database,
    string className, string fetchPlan)
{
  // LOG OPERATION
  Console.WriteLine("Fetching All Records from {0}",
    className);

  // BUILD QUERY
  string query = String.Format("SELECT FROM {0}", className);

  // RUN QUERY
  return database.Query(query, fetchPlan);
}
```

OrientDB-NET - `Select()`

This method creates an `OSqlSelect` object, which you can use in querying the database.

Querying the Database

Eventually, you'll want to access the data that you're storing on OrientDB. This method allows you to construct a query to use in retrieving documents from the database.

Syntax

```
// RETRIEVE LIST
List<ODocument> ODatabase.Select(params string[] projections)
   .From(target)
   .ToList(string fetchplan)

// RETRIEVE STRING
string ODatabase.Select(params string[] projections)
   .From(target)
   .ToString()
```

- **`projections`** Defines the columns you want returned.
- **`target`** Defines the target you want to operate on.
  - *`string target`* Where the target is a class or cluster.
  - *`ORID target`* Where the target is a Record ID.
  - *`OSqlSelect target`* Where the target is a nested `Select()` operation.
  - *`ODocument target`* Where the target is the return value from another database operation.
- **`fetch-plan`** Defines the Fetching Strategy you want to use. If you want to issue the query without a fetching strategy, execute the method without passing it arguments.

> Note that you can retrieve either a list of `ODocument` objects or a string value.

The base `Select()` method operates on an `OSqlSelect` object, which provides additional methods for conditional and grouping operations. For more information on these additional methods, see Queries

Examples

This method supports two return values: strings and lists. In cases where you find you use on form more often than others, you might build a helper function to save yourself time and typing.

```
using Orient.Client;
using System;

// FETCH ALL DOCUMENTS BY CLASS
public List<ODocument> ClassFetch(ODatabase database, string className)
{
  // LOG OPERATION
  Console.WriteLine("Fetching Documents from: {0}", className);

  // FETCH DOCUMENTS
  return database.Select().From(className).ToList();
}
```

For classes that contain particularly large number of records, you might find it more useful to issue the query with a fetching strategy.

```
public List<ODocument> ClassFetch(ODatabase database,
    string className, string fetchPlan)
{
  // LOG OPERATION
  Console.WriteLine("Fetching Documents from: {0}", className);

  return database.Select().From(className).ToList(fetchPlan);
}
```

> For more information on queries in OrientDB-NET, see Queries.

OrientDB-NET - `SqlBatch()`

This method prepares a `Batch` command. The return value is an `OCommandQuery` object.

Executing Batch Commands

OrientDB supports scripting arbitrary commands through a minimal SQL engine to batch commands together. Using `SqlBatch()` command, you can execute these scripts from within your C#/.NET application.

Syntax

```
OCommandQuery ODatabase.SqlBatch(string command)
```

- **`command`** Defines the command you want to execute.

Example

For instance, say you have a business application that tracks accounts in different regions and sales persons assigned to that region. Whenever someone joins the team, you need to update OrientDB to tell the application who they are and their assigned region.

```
public AddToSales(ODatabase database, string name, string region)
{
   string query = string.Format("begin
      let region = select from Region where name = {0}
      let employee = create vertex Sales set name = '{1}'
      let e = create edge Assignment from $employee to $account
      commit retry 100
      return $e", region, name);

   return database.SqlBatch(query);
}
```

OrientDB-NET - `Update()`

This method updates records on the database.

Updating Records

Using this method you can change or otherwise modify records on the database. By itself, it initializes an `OSqlUpdate` object, which you can then operate on to further define the records you want to change and what changes you would like to make.

Syntax

```
// UPDATING RECORDS
OSqlUpdate ODatabase.Update()
```

There are several methods available to the `OSqlUpdate` method. These allow you to build and execute the update against your database.

Defining the Target

When using this method, you can update by class, cluster or Record ID.

- **Class Updates**

  ```
  OSqlUpdate ODatabase.Update()
     .Class(string className)
  ```

- **Cluster Updates**

  ```
  OSqlUpdate ODatabase.Update()
     .Cluster(string clusterName)
  ```

- **Record Updates**

  ```
  // RECORD METHOD
  OSqlUpdate ODatabase.Update()
     .Record(ORID recordID)

  // RECORD ARGUMENT
  OSqlUpdate ODatabase.Update(ORID recordID)
  ```

For the sake of simplicity, syntax cases shown hereafter assume that you're updating a class.

Defining the Update

When using this method, you have a few options in defining the update that you want to make: whether you're adding or removing data, or changing records.

- **Setting Field**

  ```
  OSqlUpdate ODatabase.Update(<rid>)
     .Set(field, value)
  ```

- **Adding Fields**

  ```
  OSqlUpdate ODatabase.Update(<rid>)
     .Add(field, value)
  ```

- **Removing Fields**

  ```
  OSqlUpdate ODatabase.Update(<rid>)
     .Remove(field)
  ```

Executing Updates

Using the above methods you can build the update that you want, however OrientDB-NET does not execute the update on your database until you use an execution method:

- `Run()` Executes the update on the database and returns an integer.
- `ToString()` Executes the update on the database and returns a string.

Example

For instance, say that you routinely implement complicated updates on your database with various changes. You might wnat to build a helper function that loops through common variables in defining the update.

```
using Orient.Client;
using System;
...

// UPDATE OPERATION
public ODocument UpdateDatabase(ODatabase database, string className,
      Dictionary<string, string> changes)
{
   // LOG OPERATION
   Console.WriteLine("Updating Class: {0}", className);

   // INITIALIZE UPDATE
   OSqlUpdate update = database.Update().Class(classname);

   // DEFINE CHANGES
   foreach(KeyValuePair<string, string> setting in changes)
   {
      // APPLY CHANGES
      update.Set(setting.Key, setting.Value);
   }

   // RUN AND RETURN ODOCUMENT
   return update.Run();
}
```

OrientDB-NET - Building Queries

When querying a database on OrientDB through your C#/.NET application there are a few options available to you through the `ODatabase` methods. While you can issue SQL statements through `Query()` and `Command()`, there are also methods that allow you to build queries in C#: `Insert()`, `Select()`, and `Update()`

These queries support common conditional and grouping methods to organize the data before OrientDB returns it to your application.

- **Conditional Methods** These are in building `WHERE` clauses within OrientDB-NET.
- **Limiter Methods** These are used in limited or offsetting the result-set.
- **Sort Methods** These are used in sorting or otherwise ordering the result-set.

OrientDB-NET - Conditionals

In SQL, the `WHERE` clause defines conditions for returning data. When building queries in OrientDB-NET on `ODatabase` objects, there are a series of methods available in setting conditions on the query result-set.

There are two parts to a conditional statement, the field you want to set the condition on and the value you want from that field.

Setting the Field

In setting the field for a condition, there are three methods available to you.

- **`Where()`** Defines the initial field in the condition.
- **`And()`** Defines a condition where the previous cases and this both return true.
- **`Or()`** Defines a condition where either the previous cases or this return true.

For instance, say you want to retrieve blog entries from a database where the user is considered active and the entries are flagged as published.

```
List<ODocument> blogEntries = database.Select()
   .From('Blog')
   .Where('user-status').Equals<string>('active')
   .And('published').Equals<bool>(true)
   .ToList();
```

Setting the Value

Once you have defined the field for a condition, the second method is the value you want to check on that field. There are several methods available to use in checking the values.

- **`Equals<T>()`** Returns the record if the value of the field is equal to the given argument.
- **`NotEquals<T>()`** Returns the record if the value of the field is not equal to the given argument.
- **`Lesser<T>()`** Returns the record if the value of the field is less than the given argument.
- **`LesserEqual<T>()`** Returns the record if the value of the field is less than or equal to the given argument.
- **`Greater<T>()`** Returns the record if the value of the field is greater than the given argument.
- **`GreaterEqual<T>()`** Returns the record if the value of the field is greater than or equal to the given argument.
- **`Like<T>()`** Returns the record if the value of the field is similar to the given argument.
- **`In<T>()`** Returns the record if the value occurs in the given argument list.
- **`Lucene<T>()`** Returns true if the value occurs in the given argument, searched using the Lucene Engine.
- **`IsNull()`** Returns the record if the value of the field is `NULL`.
- **`Contains<T>()`** Returns the record if the value of the field contains the given argument.

For instance, say you want to retrieve blog entries for active users that have not set their profile picture.

```
List<string, ODocument> blogEntries = database.Select()
   .From('Blog')
   .Where('user-status').Equals<string>('active')
   .And('profile-picture').IsNull()
   .ToList();
```

OrientDB-NET - Limiters

Queries made against large databases can return many more records than you need for a particular operation. In OrientDB as with other databases, there are limiters available to determine how much data gets returned to you.

Limiters

There are three limiters available to you, mostly used with `Select()` queries:

- **`Between()`** This method takes two arguments, which are integers. It limits the return records to those that occur between them.
- **`Skip()`** This method takes one argument, which is an integer. It acts as offset, only returning the records that occur after this point.
- **`Limit()`** This mehtod takes one argument, which is an integer. It defines the maximum number of records to return.

For instance,

```
List<ODocument> blogEntries = database.Select()
   .From('Blog')
   .Limit(50)
   .ToList();
```

Here, you query the database for a list of the first fifty blog entries.

OrientDB-NET - Sort

When OrientDB returns a result-set to your application, it isn't always arranged in a manner that is convenient to your uses. Rather than putting your own resources to sorting results, you can add sort methods to the query to ensure that OrientDB returns the records in the order you want.

Sorting Records

There are three methods available to you in sorting the result-sets OrientDB sends you on `Select()` queries.

- **`OrderBy()`** This method takes a `params string[]` argument that defines the fields to use in sorting the result-set.
- **`Ascending()`** This method sorts the result-set in ascending order.
- **`Descending()`** This method sorts the result-set in descending order.

For instance,

```
List<ODocument> blogs = database.Select()
   .From("Blog")
   .OrderBy('date')
   .Descending()
   .Limit(10)
   .ToList();
```

Here, you have a web application that is rendering a series of blog entries. You would like the entries sorted by the date and display the ten most recent entries on the home page.

OrientDB-NET - OTransaction

Transactions allow you to organize a series of commands on the database into units of work. Once you have done the work that you want to do, you can then commit the transaction to the database to make it persistent or revert the database to an earlier state.

In OrientDB-NET, tranasctions are controlled through the `OTransaction` object, which you can access through the `ODatabase` interface.

For more information, see Transactions.

Initialize a Transaction

In order to initialize a transaction, you need to first create a transaction object to manage and identify the changes you are making. This is handled through the database interface.

```
OTransaction trx = ODatabase.Transaction;
```

This initializes an `OTransaction` object that you can then use in further operations, building the transaction before you commit it to the database.

Working with Transactions

Once you have the `OTransaction` interface initialized, you can call methods on this object to build the transaction. These are similar to the methods you would normally call on the `ODatabase` interface, but specific to the transaction.

Method	Description
`Add<T>()`	Adds an object, typed by the generic.
`AddEdge()`	Adds an edge.
`AddOrUpdate<T>()`	Adds a new object or updates an existing one.
`Commit()`	Commits changes made in the transaction to the database.
`Delete<T>()`	Removes a record.
`GetPendingObject<T>()`	Retrieves last object of the given type in the transaction.
`Reset()`	Closes the transaction and reverts all changes made to it.
`Update<T>()`	Updates records on the database.

`Commit()`

When you make changes to the database through an `OTransaction` object, these changes are not persistent. In order to make them persistent, you need to commit your changes to the database. Or, in the event that you aren't happy with the changes, you can revert the database to an earlier state, that is the state of the last commit or before you opened the transaction.

To commit your changes to the database, call the `Commit()` method on the transaction.

```
// COMMIT TRANSACTION
trx.Commit();
```

All changes made on the transaction are now committed to the database.

`Reset()`

When you make changes on the database that you are not happy with, such as in cases where there's a problem or issue with the database state, you can roll the data back to an earlier state by calling the `Revert()` method on the transaction.

```
// REVERT CHANGES ON TRANSACTION
trx.Revert();
```

All changes made on the transaction are removed. The database reverts to its earlier state.

OrientDB-NET - `Add<T>()`

This method adds records to the database. The new records remain part of the transaction and can either be removed or made persistent, through `Commit()` or `Revert()`.

Adding Records

In order to add records to the database, you need to initialize the objects and then pass them to the `Add<T>()` method.

Syntax

```
OTransaction.Add<T>(T typedObject)
```

Example

For instance, if you find yourself often adding records with complex information or changes made to multiple fields, you may find it useful to implement a helper function to simplify these operations.

```
using Orient.Client;
using System;
...

// ADD RECORDS TO THE DATABASE
public void AddRecords(OTransaction trx, List<Dictionary<string, string>>    records)
{
   // LOG OPERATION
   Console.WriteLine("Adding Records to Transaction");

   // LOOP OVER NEW RECORDS LIST
   foreach(Dictionary<string, string> record in records)
   {
      // INITIALIZE RECORD
      ODocument document = ODocument();

      // DEFINE RECORD CONTENTS
      foreach(KeyValuePair<string, string> field in record)
      {
         // DEFINE FIELD
         document.SetField<string>(field.Key, field.Value);
      }

      // ADD TO RECORD TO TRANSACTION
      trx.Add<ODocument>(document);
   }

}
```

OrientDB-NET - `AddEdge()`

This method is used to add an edge to the database. The new edge remains a part of the transaction until you commit the change and can be moved by rolling back to an earlier state.

Adding Edges

In order to add an edge to the database, you need to create an `OEdge` object, then call the `AddEdge()` method, passing to it the connecting vertices as arguments.

Syntax

```
OTransaction.AddEdge( OEdge edge,
             OVertex fromVertex,
             OVertex toVertex)
```

- **`edge`** Defines the edge object you want to add.
- **`fromVertex`** Defines the vertex the edge connects from.
- **`toVertex`** Defines the vertex th eedge connects to.

Example

In cases where you have a class that connects to multiple edges, you may find it more convenient to use a helper function to quickly define and add edges to records.

```
using Orient.Client;
using System;
...

// CONNECT EDGES
public void TrxConnectEdges(OTransaction trx, Dictionary<OEdge, Dictionary<string, OVertex>> edges)
{
   // LOG OPERATION
   Console.WriteLine("Adding Edges");

   // LOOP OVER EACH EDGE
   foreach(KeyValuePair<OEdge, Dictionary<string, OVertex>> edge in edges)
   {
      // DEFINE VERTICES
      OVertex from = edge.Value['from'];
      OVertex to = edge.Value['to'];

      // ADD VERTICES
      trx.AddEdge(edge.Key, from, to);
   }
}
```

OrientDB-NET - `AddOrUpdate<T>()`

This method adds a new record to the database. In the event that the record already exists, it updates the record with new data.

Adding or Updating Records

In deployments where there is the risk of the database changing while the transaction is open or where you are uncertain if a class exists on the database, you can call the `AddOrUpdate()` method to add a new record or update an existing one, depending on whether or not the record exists already on the database.

Syntax

```
OTranasction.AddOrUpdate<T>(T target)
```

- `target` Defines the object you want to add or update on the database. It is of the type defined by the generic.

Example

For instance, if you find yourself often adding or updating records with complex, but routine, ifnromation, you may find it useful to implement a helper function to simplify these operations.

```
using Orient.Client;
using System;
...

// ADD OR UPDATE TRANSACTION
public void TrxUpdate(OTransaction trx, Dictionary<ORID, Dictionary<string, string>> records)
{
   // LOG OPERATION
   Console.WriteLine("Updating Records in Transaction");

   // LOOP OVER RECORDS
   foreach(KeyValuePair<ORID, Dictionary<string, string>> record in records)
   {
      // LOAD RECORD
      ODocument document = LoadRecord().ORID(record.Key);

      // DEFINE FIELDS
      foreach(KeyValuePair<string, string> field in record.Value)
      {
          // SET FIELD
          document.SetField<string>(field.Key, field.Value);
      }

      // ADD OR UPDATE RECORD
      trx.AddOrUpdate<ODocument>(document);
   }
}
```

OrientDB-NET - `Delete<T>()`

This method is used to remove records from the database. Define the type through the generic.

Removing Records

In certain situations you may want to programmatically remove records from OrientDB. Using the `Delete<T>()` method you can remove the instance as part of a transaction.

Syntax

```
OTransaction.Delete<T>(T typedObject)
```

- **`typedObject`** Defines the object you want to remove. The object must be of the same type as defined in the `T` generic.

Example

For instance, say that you want to create a function for removing records from the database.

```
public void removeDocument(OTransaction trx, ODocument document)
{
   // REMOVE RECORD
   trx.Delete<ODocument>(document);
}
```

OrientDB-NET - `GetPendingObject<T>()`

This method returns objects from a transaction.

Retrieve Pending Objects

In cases where you need to operate on particular records already added to a transaction, you can use `GetPendingObject<T>()` to retrieve it.

Syntax

```
T trx.GetPendingObject<T>(ORID rid)
```

- **`rid`** Defines the Record ID that you want to retrieve.

Example

For instance, say you want to retrieve a vertex from a transaction:

```
public OVertex fetchVertex(OTransaction trx, ORID rid)
{
   // FETCH VERTEX
   OVertex target = trx.GetPendingObject<OVertex>(rid);
   return target;
}
```

OrientDB-NET - `Update<T>()`

This method allows you to update records as part of a transaction.

Updating Records

Using this method you can update records that already exist in the database as part of a transaction. This way you can evaluate the changes before committing them to the database.

Syntax

```
void trx.Update<T>(T typedObject)
```

- `typedObject` Defines the object you want to update.

Example

For instance, if you find yourself often updating records with complex information or changes made to multiple fields, you may find it useful to implement a helper function to simplify this process.

```
using Orient.Client;
using System;
...

// UPDATE RECORDS
public void updateRecord(OTransaction trx, Dictionary<ODocument, Dictionary<string, string>> records)
{
   // LOG OPERATION
   Console.WriteLine("Update Records");

   // LOOP OVER DOCUMENTS
   foreach(KeyValuePair<ODocument, Dictionary<string, string>> record in records)
   {
      // INITILAIZE VARIABLES
      ODocument document = record.Key;
      Dictionary<string, string> fields = record.Value;

      // SET CHANGES
      foreach(KeyValuePair<string, string> field in fields)
      {
          // SET FIELD
          document.SetField<string>(field.Key, field.Value);
      }

      // APPLY CHANGES TO TRANSACTION
      trx.Update<ODocument>(document);
   }
}
```

PhpOrient

OrientDB supports a number of application programming interfaces, natively through the JVM and externally through the Binary Protocol. In the event that you need or would prefer to develop your application for OrientDB using PHP, you can do so through the official driver: PhpOrient.

It requires OrientDB version 1.7.4 or later. It also requires that your application use PHP version 5.4 or later, with the Socket extension enabled.

Getting PhpOrient

In order to use PhpOrient with your application, you first need to install it on your system. There are two methods available to you in installing PhpOrient: registering PhpOrient as a dependency of your application or manually retrieving the source code and installing it separately on your system. Both methods require PHP Composer.

Dependency Registration

When working with an existing project or in cases where you already have PHP Composer installed, you can register PhpOrient as a dependency of your project. To do so, run the following command from your project directory:

```
$ php composer.phar require "ostico/phporient:dev-master" \
      --update-no-dev
```

Running this command registers PhpOrient as a requirement of your project. It also links the requirement to the GitHub repository for PhpOrient, so you can also install updates as they become available.

Manual Installation

In cases where you would like to manually install PhpOrient, you can retrieve the source code from GitHub, then install it locally with dependencies through PHP Composer.

1. Using `git`, retrieve the source code from GitHub:

   ```
   $ git clone https://github.com/Ostico/PhpOrient.git
   ```

2. Retrieve and install PHP Composer:

   ```
   $ cd PhpOrient
   $ php -r "readfile('https://getcomposer.org/installer');" | php
   ```

3. Install dependencies:

   ```
   php composer.phar --no-dev install
   ```

PhpOrient is now installed and ready for use on your system.

32-bit Architecture

While PhpOrient supports both 32-bit and 64-bit architectures, there are some issues that you should keep in mind when deploying applications on 32-bit systems. In order to support Java long integers and to prove better driver performance on these systems, you must install one of the following libraries:

- BCMath Arbitrary Precision Mathematics
- GNU Multiple Precision

When your 32-bit application receives a Java long integer from OrientDB, values greater than 2^32 are lost as they exceed the maximum possible number that the system can store in thirty-two bits. To get around this limitation, PhpOrient uses with either the BCMath or GMP libraries to always handle numbers as strings.

Using PhpOrient

During installation, PHP Composer generates an autoload file that you can use in retrieving PhpOrient classes and functions to use with your application. To implement these features, add the following lines to any file where you would like to have these features available:

```
require "vendor/autoload.php;
use PhpOrient\PhpOrient;
```

PhpOrient - Client Connections

Within your PHP application, performing operations on OrientDB servers and databases requires a client interface. By convention, this interface is called `$client` within these docs, but you can use any variable name you prefer. Once you have initialized this connection, you can begin to call additional methods to operate on servers and databases.

Initializing Client Connections

In order to initialize a client connection within your application, you need to create a new instance of `PhpOrient()`. Once you have instantiating the class, you need to define variables within the class to configure its connection.

```
require "vendor/autoload.php";
use PhpOrient\PhpOrient;

// INITIALIZE CLIENT CONNECTION
$client =  new PhpOrient('localhost', 2424);

// SET CREDENTIALS
$client->username = 'admin';
$client->password = 'admin_passwd';
```

Alternative Methods

In addition to the method shown above, there are two alternative methods available in configuring the client connection. The first of these is to manually set all of the configuration variables.

```
// INITIALIZE CLIENT
$client = new PhpOrient();

// CONFIGURE CLIENT
$client->hostname    = 'localhost';
$client->port        = 2424;
$client->username    = 'root';
$client->password    = 'root_passwd';
```

Alternatively, you can set the variables through the `configure()` method.

```
// INITIALIZE CLIENT
$client = new PhpOrient();

// CONFIGURE CLIENT
$client->configure(
    array(
        'username'    => 'root',
        'password'    => 'root_passwd',
        'hostname'    => 'localhost',
        'port'        => 2424));
```

Persistent Connections

Beginning in version 27, PhpOrient provides support for persistent client connections. This allows you to set a session token, then connect, disconnect and reconnect to the given session as suits the needs of your application.

When using persistent connections, use the `setSessionToken()` method with a boolean value to enable the feature. Then use the `getSessionToken()` method to retrieve a token for the given session.

```
// INITIALIZE CLIENT
$client = new PhpOrient('localhost', 2424);

// ENABLE PERSISTENT CONNECTIONS
$client->setSessionToken(true);

// RETRIEVE SESSION TOKEN
$sessionToken = $client->getSessionToken();
```

Once you have the session token, you can reattach to an existing session using it. For instance, elsewhere in your code, you might want a function to handle the reconnection:

```
// CREATE CLIENT
function getClient($sessionToken){

    // Instantiate Client
    $client = new PhpOrient("localhost", 2424);

    // Open Session
    $client->setSessionToken($sessionToken);

    return $client;
}
```

In cases where you call `getSessionToken()` before enabling persistent connections on the client, the method returns an empty string. You can use this behavior to perform basic checks to ensure everything is working properly. For instance,

```
function getToken(){

    // Log Operation
    echo "Retrieving Session Token";

    // Fetch Globals
    global $client;
    global $sessionToken;

    // Set Session Token
    $sessionToken = $client->getSessionToken();

    if ($sessionToken == ''){

        // Enable Session Token
        $client->setSessionToken(true);

        // Fetch New Token
        $sessionToken = $client->getSessionToken();
    }

}
```

PhpOrient - Server Operations

Once you have the client interface ready in your application, you can begin to operate on OrientDB servers and databases. In order to administer the server, you first need to connect to it.

Connecting to the Server

When you need to perform server operations, instantiate the client interface using credentials that can access your OrientDB Server, then issue the `connect()` method to establish a connection to the server.

```
require "vendor/autoload.php";
use PhpOrient\PhpOrient;

// INITIALIZE CLIENT
$client = new PhpOrient('localhost' 2424);

// CONFIGURE CONNECTION
$client->username = 'root';
$client->password = 'root_passwd';

// CONNECT
$client->connect();
```

When your application gets to this point, the `$client` variable is now connected to the OrientDB Server and able to perform various operations.

> **NOTE**: There are several methods available to you in initializing a client connection. For more information on the available methods, see Client Connections.

Operating on the Server

The client interface provides a number of methods for database and server operations. The table below provides a list of methods for server operations, specifically in creating, opening, listing and removing databases on the server.

Method	Description
`dbCreate()`	Creates a database
`dbDrop()`	Removes a database
`dbExists()`	Checks that database exists
`dbList()`	Lists databases on server
`dbOpen()`	Opens an existing database

PhpOrient - `dbCreate()`

Creates a database on the connected OrientDB Server.

Creating Databases

In the event that a database does not already exist on the server, you can create one from within your application, using the `dbCreate()` method. This method requires one argument, the database name, and can take two additional arguments defining the storage and database types. It returns the default cluster ID.

Syntax

```
$new_cluster_id = $client->dbCreate(
    <database>,
    <storage-type>,
    <database-type>)
```

- **`<database>`** Defines the database name.
- **`<storage-type>`** Defines the storage type to use. Valid storage types:
    - *`PhpOrient::STORAGE_TYPE_PLOCAL`* Sets PLocal storage. This is the default option.
    - *`PhpOrient::STORAGE_TYPE_MEMORY`* Sets in-memory storage.
- **`<database-type>`** Defines the database type to create. Valid database types:
    - *`PhpOrient::DATABASE_TYPE_GRAPH`* Sets the method to create a graph database. This is the default option.
    - *`PhpOrient:DATABASE_TYPE_DOCUMENT`* Sets the method to create a document database.

Example

Consider the use case of a web application. Rather than just assuming that OrientDB is ready to serve data to your application, you might want to start by checking whether a database exists and is ready for your use and in the event that it doesn't exist, have your application create it for you. For instance,

```
// OPEN OR CREATE DATABASE
function dbOpenCreate($dbname, $user, $passwd){

    // RETRIEVE GLOBAL CLIENT
    global $client;

    // CHECK IF EXISTS
    if !($client->dbExists($dbname)){
        // CREATE DATABASE
        $client->dbCreate($dbname,
            PhpOrient::STORAGE_TYPE_PLOCAL,
            PhpOrient::DATABASE_TYPE_GRAPH);
    }
    return $client->dbOpen($dbname, $user, $passwd);
}
```

This function takes the database name and database login credentials as arguments. Using `dbExists()` it determines whether the database exists on the database and creates it if it doesn't exist. Then it opens the given database, returning the cluster map.

PhpOrient - `dbDrop()`

Remove the given database from OrientDB.

Dropping Databases

In certain situations, you may want to remove a full database from the OrientDB Server. For instance, if you create a temporary database in memory for certain operations or if you want to provide the user with the ability to uninstall the database from within the application, without removing OrientDB itself.

Syntax

```
$client->dbDrop(
    <database-name>,
    <storage-type>)
```

- **`<database-name>`** Defines the database name.
- **`<storage-type>`** Defines the storage type. This is an optional variable, defaults to PLocal.
  - *`PhpOrinet::STORAGE_TYPE_PLOCAL`* Sets the PLocal storage type.
  - *`PhpOrient::STORAGE_TYPE_MEMORY`* Sets in-memory storage type.

Example

For instance, in the event that your application encounters a catastrophic failure in the database, you might want a function that allows you to reset it to a clean state.

```
// RESET DATABASE
function resetDatabase($client, $dbname){

    // REMOVE DATABASE
    $client->dbDrop($dbname, PhpOrient::STORAGE_TYPE_MEMORY);

    // CREATE DATABASE
    $client->dbCreate($dbname, PhpOrient::STORAGE_TYPE_MEMORY);
}
```

This function removes the given database, then uses the `dbCreate()` method to create a new one with the same name and storage type.

PhpOrient - `dbExists()`

This method determines whether a database exists on the server.

Checking Existence

There are two methods available in determining whether a database exists on the OrientDB Server. You can use the `dbList()` method to retrieve a list of the databases on the server and check them individually within your PHP code or you can individually check database names against this method.

Syntax

```
$client->dbExists(
    <database-name>,
    <database-type>)
```

- **`<database-name>`** Defines the database name.
- **`<database-type>`** Defines the storage type. The default type is a Graph database. Valid types:
  - *`PhpOrient::DATABASE_TYPE_GRAPH`* Sets to Graph database.
  - *`PhpOrient::DATABASE_TYPE_DOCUMENT`* Sets to Document database.

Example

Consider the use case of a web application. Rather than just assuming that OrientDB is ready to serve data to your application, you might want to start by checking whether a database exists and is ready for your use and in the event that it doesn't exist, have your application create it for you. For instance,

```
// OPEN OR CREATE DATABASE
function dbOpenCreate($client, $dbname, $user, $passwd){

    // CHECK IF EXISTS
    if !($client->dbExists($dbname)){
        // CREATE DATABASE
        $client->dbCreate($dbname,
            PhpOrient::STORAGE_TYPE_PLOCAL,
            PhpOrient::DATABASE_TYPE_GRAPH);
    }
    return $client->dbOpen($dbname, $user, $passwd);
}
```

This function takes the client interface, database name and login credentials as arguments. It uses this method to check whether the given database exists. If it does not exist, it creates it with `dbCreate()`. Then, it opens the given database, returning the cluster map.

PhpOrient - `dbList()`

Retrieves a list of databases on the server.

Listing Databases

In certain situations, you may want to list all databases available on the OrientDB Server. You might find this useful when operating on multiple databases or when logging.

Syntax

```
$client->dbList()
```

Example

For instance, when developing a web application that servers multiple sites each with its own database, you might want to list the database names in your logs when the application starts.

```
// LOOP OVER DATABASES AND ECHO NAMES
foreach ($client->dbList() as $db){
    echo "Database: $db";
}
```

PhpOrient - `dbOpen()`

This method opens a database on the client interface. The return value is a cluster map.

Opening Databases

When the database you want exists already on the OrientDB Server, you can open it on the client interface using this method with the appropriate name and login credentials. Once you have it open, a series of additional methods become available through the client interface. These allow you to insert, retrieve and modify records in the database.

Syntax

```
$client->dbOpen('<database-name>', '<user>', '<password>')
```

- `<database-name>` Defines the database you want to open.
- `<user>` Defines the user you want to access the database.
- `<password>` Defines the password to use.

Example

For instance, in the use case of web application, you might use something like this to connect to the database.

```
// CONNECT TO DATABASE
$ClusterMap = $client->dbOpen('GratefulDeadConcerts', 'admin', 'admin_passwd');
```

Working with Database

Once you have an open database on the client interface, a series of additional methods become available to you. These methods handle common operations on the database, in terms of inserting and fetching records as well as manipulating clusters on the database.

Method	Description
`command()`	Executes a command on the database.
`dataClusterAdd()`	Adds a cluster to the database.
`dataClusterCount()`	Counts records in a cluster or clusters.
`dataClusterDrop()`	Removes a cluster from the database.
`dataClusterDataRange()`	retrieves a range of Record ID's for the given cluster.
`dbCountRecords()`	Counts records on a database.
`dbReload()`	Reloads the database on the client interface.
`dbSize()`	Returns the size of the database.
`getTransactionStatement()`	Instantiates a transaction interface.
`query()`	Queries the database.
`queryAsync()`	Queries the database with support for callback functions and Fetching Strategies.
`recordCreate()`	Creates a record on database.
`recordLoad()`	Loads a record from the database.
`recordUpdate()`	Updates a record on the database.
`sqlBatch()`	Executes an SQL batch command.

PhpOrient - `command()`

This method issues an SQL command to the database.

Sending Commands

In certain situations, you may find it more convenient or preferable to issue commands to the database using SQL rather than PhpOrient methods. Use this method only to perform non-idempotent commands.

Syntax

```
$client->command(<sql>)
```

- `<sql>` Defines the SQL command to run.

Example

For instance, if you find yourself frequently inserting complex data into your database, you might want to develop a function that takes the client interface, class, and an array mapping property names to values.

```
function insertData($client, $class, $dataArray){

    // CONSTRUCT BASE SQL INSERT STATEMENT
    $sql = 'INSERT INTO $class';

    // LOOP IN DATA VALUES FROM ARRAY
    foreach($dataArray as $property => $value) {

        // ADD INSERT
        $sql = '$sql SET $property = \'$value\'';
    }

    // ISSUE COMMAND
    $client->command($sql);
}
```

Here, your function constructs an `INSERT` statement from the given array, then issues the command to OrientDB.

PhpOrient - `dataClusterAdd()`

This method creates a new cluster on the database.

Adding Clusters

In cases where you want to create clusters programmatically, such as in an initialization script that prepares OrientDB for your application, this method allows you to add new physical and in-memory clusters to the databases.

Syntax

```
$client->dataClusterAdd(
    "<name>",
    <cluster-type>)
```

- `<name>` Defines the cluster name. By convention, this is generally lowercase.
- `<cluster-type>` Defines the cluster type. Supported types are:
  - `PhpOrient::CLUSTER_TYPE_PHYSICAL` Sets the method to create a physical cluster, which is the default.
  - `PhpOrient::CLUSTER_TYPE_MEMORY` Sets the method to create an in-memory cluster.

Example

For instance, imagine you have an application that stores volatile data in-memory. You might want a function to create a series of in-memory clusters as need.

```
// CREATE AD-HOC MEMORY CLUSTERS
function createMemClusters($names){

    // LOG OPERATION
    echo "Creating Clusters";

    // FETCH GLOBAL CLIENT
    global $client;

    // CREATE CLUSTERS
    foreach($names as $name){

        // CREATE IN-MEMORY CLUSTER
        $client->dataClusterAdd($name,
            PhpOrient::CLUSTER_TYPE_MEMORY);
    }
}
```

PhpOrient - `dataClusterCount()`

This method returns a count of records in the given cluster.

Counting Records

In certain situations you may find it useful to check how many records a given cluster contains, for instance as part of logging or debugging, or to check cluster size before running a backup script.

Syntax

```
$client->dataClusterCount(<clusters>)
```

- `<clusters>` Defines the cluster or clusters you want to count on.

Example

Picture a web application where you want to count records in a given cluster. For instance, on a blog site you might want a basic count to show the number of entries you host on the site across several entries.

```
function countBlogs(){

    // Fetch Global Client
    global $client;

    // Return Record Count
    return $client->dataClusterCount(
        $client->getTransport()->getClusterMap()->getIdList());
}
```

PhpOrient - `dataClusterDrop()`

This method drops the given cluster.

Dropping Clusters

In certain situations, you may find it useful to programmatically remove clusters from the database. For instance, in the case of a maintenance script that frees up space when the given cluster is no longer needed. This method allows you to remove clusters by their Cluster ID, which is the first number in a Record ID.

Syntax

```
$client->dataClusterDrop(<cluster-id>)
```

- `<cluster-id>` Defines the Cluster ID to remove.

Example

Consider the example of a web application that uses multiple in-memory clusters for short term needs. For this purpose, you might want a function to quickly remove these clusters when you're finished using them.

```
// REMOVE CLUSTER
function removeCluster($clusterId){

    // Log Operation
    echo "Removing Cluster: $clusterId";

    // Fetch Global Client
    global $client;

    // Remove Cluster
    $client->dataClusterDrop($clusterId);
}
```

PhpOrient - `dataClusterDataRange()`

This method fetches a range of Record ID's by cluster.

Retrieving Ranges of Records

In some cases, you may want to retrieve all records in a given cluster. This method uses the Cluster ID to identify the records you want to access.

Syntax

```
$client->dataClusterDataRange(<cluster-id>)
```

- `<cluster-id>` Defines the Cluster ID for records you want to retrieve records from.

Example

Consider the example of a web application where you need to periodically retrieve and operate on large bodies of records by cluster. Rather than manually retrieving clusters by ID, you might set up global map to keep the references human readable.

```
// Clusters
$clusters = array(
    "BlogsUS" => array(3, 4, 5),
    "BlogsEU" => array(1, 2, 6, 9),
    "BlogsME" => array(7, 8)
);

// Fetch Record ID's
function fetchRIds($regionName){

    // Log Operation
    echo "Fetching Records: $regionName";

    // Initialize Global Variables
    global $clusters;
    global $client;

    // Initialize Local Variables
    $returnArray = array();
    $region = $clusters[$regionName];

    // Loop Over Cluster ID's in Region
    foreach($region as $clusterId){

        // Append Record ID's to Return Array
        $returnArray[] = $client->dataClusterDataRange($clusterId);
    }

    // Return Record ID's
    return $returnArray;
}
```

PhpOrient - `dbCountRecords()`

This method returns the number of records in the database.

Counting Records

There are three methods available to you in counting or sizing databases. This method returns the number of records in the database. You can also retrieve the size of the database with `dbSize()` and the number of records in a cluster with `dataClusterCount()` .

Syntax

```
$client->dbCountRecords()
```

Example

As an example, you might use this method in conjunction with unit testing, especially after performing a restore operation on a new server. That is, a quick way to determine whether the restore process properly executed is to check whether the database contains records.

```
// TEST DATABASE RESTORE
function testRestore($client, $user, $password, $databaseName){
    // LOG OPERATION
    echo "Checking Database: $databaseName";

    // TEST WHETHER DATABASE EXISTS
    assert($client->dbExists($databaseName));

    // TEST THAT DATABASE CONTAINS RECORDS
    $client->dbOpen($databaseName, $user, $password);
    assert($client->dbCountRecords() > 0);
}
```

PhpOrient - `dbReload()`

This method updates the client cluster map.

Reloading the Database

When you create or remove a new class or cluster on the database, it updates OrientDB but not the client interface you have created within your application. The client interface also does update when changes are made to OrientDB apart from your application. In these cases, you can use this method to retrieve an updated cluster map from OrientDB.

Syntax

```
$client->dbReload()
```

Example

For instance, rather than calling this method manually after creating a class on the database, you might develop your own function to save yourself the trouble of remembering to make the call elsewhere.

```
// ADD CLUSTER FUNCTION
function addCluster($client, $clusterName){

    // LOG OPERATION
    echo "Adding Cluster $clusterName";

    // ADD CLUSTER TO DATABASE
    $client->dataClusterAdd($clusterName, PhpOrient::CLUSTER_TYPE_PHYSICAL);

    // RELOAD DATABASE
    $reloaded_list = $client->dbReload;

    // RETURN NEW LIST
    return $reloaded_list;
}
```

PhpOrient - `dbSize()`

This method returns the size of the database.

Sizing Databases

There are three methods available to you in counting or sizing databases. This method returns the size of the database, `dbCountRecords()` returns the number of records in the database, and `dataClusterCount()` the number of records in a cluster.

Syntax

```
$client->dbSize()
```

Example

For instance, you might use this method in conjunction with unit testing, especially after performing a restore operation on a new server. That is, a quick way to determine whether the restore process properly executed is to check whether the database contains records.

```
// TEST DATABASE RESTORE
function testRestore($client, $user, $password, $databaseName){

    // LOG OPERATION
    echo "Checking Database: $databaseName";

    // TEST DATABASE EXISTENCE
    assert($client->dbExists($databaseName));

    // TEST THAT DATABASE CONTAINS RECORDS
    $client->dbOpen($databaseName, $user, $password);
    assert($client->dbSize() > 0);
}
```

PhpOrient - `query()`

This method issues an SQL query to the database.

Querying the Database

In the event htat you're more comfortable working in SQL, you can build and issue queries to OrientDB directly using this method.

Syntax

```
$client->query(<sql>)
```

- `<sql>` Defines the query you want to issue.

Example

In cases where you find yourself frequently issuing queries to OrientDB, you may find it convenient to construnction a function to manage the process.

```
// QUERY FUNCTION
function queryDatabase($className,
            $properties = array('*'),
            $whereCondtions = array(),
            $limit = 0){

    // LOG OPERATION
    echo "Querying $className";

    // FETCH GLOBAL CLIENT
    global $client;

    // CONSTRUCT SELECT STATEMENT
    if ($properties == array('*')) {
        $sql = "SELECT FROM $classname";
    } else {
        $props = join(', ', $properties)
        $sql = "SELECT $prop FROM $classname";
    }

    // CONSTRUCT WHERE CLAUSE
    if ($whereConditions != array()){
        $where = "WHERE";

        // LOOP OVER CONDITIONS
        foreach($whereConditions as $property => $value){
            $where = "$where $property = \'$value\'";
        }

        // ADD WHERE TO STATEMENT
        $sql = "$sql $where";
    }

    // ADD LIMIT
    if ($limit > 0){

        // ADD LIMIT TO STATEMENT
        $sql = "$sql LIMIT $limit";
    }

    // QUERY DATABASE
    return $client->query($sql);
}
```

PhpOrient - `queryAsync()`

This method issues a query to the database. For each record the query returns, it executes the given callback function.

Querying the Database

When issue a query to OrientDB using the `query()` method, PhpOrient executes the SQL statement against the open database and then gives you all of the records as a return value. In cases where this is not the desired result, you can use this method to execute a callback function on each record the query returns.

You may find this useful in cases where you need to initiate certain calculations in advance of the final result, or to log information as the query runs.

Example

During development, you may find it useful to implement a debugging option that allows you to dump additional information to standard output. Using this method with the `var_dump()` function, you can display information on records retrieved from the database.

```
// DEFAULT DEBUG OPTION
$debug = True;

// QUERY DEBUG FUNCTION
function queryDebug(Record $record){

    // LOG OPERATION
    echo "Record Returned:";

    // DUMP RECORD INFORMATION
    global $debug;
    if ($debug){

        // DUMP RECORD
        var_dump($record);
    }
}

// QUERY DATABASE
function queryDatabase($client, $sql, $fetchPlan){

    // LOG OPERATION
    echo "Querying Database";

    // ISSUE ASYNC QUERY
    $client->queryAsync($sql,
        ['fetch_plan' => $fetchPlan,
        '_callback' => queryDebug]);
}
```

PhpOrient - `recordCreate()`

This method creates a record on the database.

Creating Records

Eventually, you'll want your application to programmatically add records to OrientDB. Using this method you can create a new record and insert it into the database.

Syntax

```
$client->recordCreate(<record>)
```

- `<record>` Defines the record to create, it is an instance of `Record()`.

Example

For instance, say that you are developing a web application to manage blogs for multiple users. When the user is ready to publish a blog entry to the site, you might use a function such as this to handle creating the new record on the database.

```
// POST FUNCTION
function postBlog($title, $text, $user){

    // LOG OPERATION
    echo "Posting Blog: $title, for $user\n";

    // FETCH GLOBAL PHPORIENT CLIENT
    global $client;

    // BUILD RECORD CONTENT
    $recordContent = [
        'title': $title,
        'author': $user,
        'text': $text];

    // BUILD RECORD
    $record = (new Record() )->setOData($recordContent)->setOClass("Blog")->setRid(new ID(9));

    // CREATE RECORD
    $client->recordCreate($record);
}
```

> Note, when creating new `Record()` objects, you only set the Cluster ID component of the Record ID. This helps avoid unexpected results when the record is created on the database in the next line.

PhpOrient - `reacordLoad()`

This method returns a record from the database.

Loading Records

In addition to the `query()` method, you can also manually select and load records from the database using this method. Unlike the query, you need to define the record you want using its Record ID, through the `ID()` class.

Syntax

In OrientDB a RecordID is built from two numeric values: the Cluster ID and the Record Position. The `ID()` class in PhpOrient provides you with a few different ways to define the particular Record ID that you want to load.

```
// DEFINING RECORD ID AS STRING
$client->recordLoad(new ID('<record-id>'))

// DEFINING CLUSTER AND POSITION AS ARGUMENTS
$client->recordLoad(new ID(<cluster-id>, <record-position>))

// DEFINING CLUSTER AND POSITION WITH ARRAY
$client->recordLoad(new ID(['cluster' => <cluster-id>,
                            'position' => <record-position>]))
```

- **`<record-id>`** Defines the Record ID that you want to return.
- **`<cluster-id>`** Defines the Cluster ID to search.
- **`<record-position>`** Defines the record's position in the cluster.

When successful, this method returns an array with a single entry. In order to access the reocrd itself, you need to add a call to the 0 position at the end of the method, for instance:

```
$record = $client->recordLoad(new ID('#3:22'))[0]
```

Examples

For instance, if you find yourself frequently loading records by Record ID, you might want to build a function that retrieves an array of record ID's.

```
// FETCH RECORDS
function fetchRecords($client, $ridArray){

    // LOG OPERATION
    echo "Retrieving Records";

    // INITIALIZE ARRAY
    $records = array();

    // LOOP OVER RID'S
    foreach($rid in $ridArray){

        // FETCH RECORD
        $record = $client->recordLoad(new ID($rid))[0];

        // APPEND ARRAY
        array_push($records, $record);
    }

    // RETURN RECORDS
    return $records;

}
```

Loading Records with Callback Function

Similar to the `queryAsync()` method, you can define a fetching strategy and callback function. You can manage these features by passing a mapped array as the second argument to the method.

For instance, during development you might want to call the `var_dump()` function for each record the method returns.

```
// DEFAULT DEBUG OPTION
$debug = True;

// QUERY DEBUG FUNCTION
function queryDebug(Record $record){

    // LOG OPERATION
    echo "Record Returned:";

    // DUMP RECORD INFORMATION
    global $debug;
    if ($debug) {

        // DUMP RECORD
        var_dump($record);
    }
}

// QUERY DATABASE
function queryDatabase($client, $rid, $fetchPlan){

    // LOG OPERATION
    echo "Loading Record: $rid";

    // LOAD RECORD
    $record = $client->recordLoad(
        new ID($rid),
        ['fetch_plan' => $fetchPlan,
         '_callback' => $queryDebug]);

    // RETURN RECORD
    return $record;
}
```

PhpOrient - `recordUpdate()`

This method updates records on the database.

Updating Records

Your applications users may occasionally want to make changes to the database. Using this method you can update existing records on the database.

Syntax

```
$client->recordUpdate(<record>)
```

- **`<record>`** Defines the record that you want to update, with relevant changes. It is an instance of the `Record()` object.

Example

For instance, consider the example of a web application that hosts blogs for multiple users. While these users may generally post new blogs to the database, they may on occasion prefer to edit existing ones.

```
// UPDATE OR CREATE RECORD
function buildRecord($title, $text, $user){

    // LOG OPERATION
    echo "Posting Blog: $title, for $user";

    // BUILD RECORD CONTENT
    $recordContent = [
        "title":  $title,
        "author": $user,
        "text":   $text];

    // BUILD RECORD
    $record = (new Record() )->setOData($recordContent)->setOClass("Blog")->setRid(new ID(9));

    // FETCH GLOBAL CLIENT
    global $client;

    // UPDATE
    $client->recordUpdate($record);
}
```

PhpOrient - `sqlBatch()`

This method issues `BATCH` commands to the database.

Executing Batch Commands

OrientDB supports the execution of arbitrary scripts written JavaScript with a minimal SQL engine for batch commands. Using this method, you can execute batch commands through your PhpOrient application.

Syntax

```
$client->sqlBatch(<batch>)
```

- `<batch>` Defines a string containing the commands you want to execute.

Example

For instance, if you have a series of records that you want to create on the database, you might find it more convenient to manage them through batch commands in a function.

```
// BATCH CREATION
function batchCreate($records){

    // LOG OPERATION
    echo "Running Batch Command";

    // INITIALIZE BATCH COMMAND
    $batchCmd = "begin; "

    // LOOP THROUGH RECORDS
    foreach($records as $class => $data){

        // INITIALIZE RECORD CREATION
        $create = "insert into $class ";

        // LOOP OVER PROPERTIES
        foreach($data as $property => $value){

            // ADD SETTINGS
            $create = "$create set $property = '$value' ";
        }

        // ADD CREATE STATEMENT
        $batchCmd = "$batchCmd $create; ";

    }

    // ADD COMMIT LINE
    $batchCmd = "$batchCmd commit retry 100;";

    // FETCH GLOBAL CLIENT
    global $client;

    // EXECUTE BATCH COMMAND
    $client->sqlBatch($batchCmd);
}
```

PhpOrient - Cluster Maps

When you open a database on the client interface, the return value is cluster map. This PHP class provides you with an interface and a series of methods for manipulating clusters on the database.

Using Clusters

In order to retrieve the cluster map for a database, you need to open the database and set the return value on a variable. For instance,

```
$clusterMap = $client->dbOpen("GratefulDeadConcerts", "admin", "admin);
```

You can then use the `$clusterMap` object in calling additional methods.

Method	Description
`count()`	Returns a count of records in the cluster.
`dropClusterID()`	Removes a cluster from the database.
`getIdList()`	Retrieves a list of Cluster ID's.

`count()`

In cases where you want to know how many records the Cluster Map contains, you can obtains this using the `count()` method. For instance, you might want to test that a database contains records after opening it:

```
// Open Database
$clusterMap = $client->dbOpen("GratefulDeadConcerts", "admin", "admin);

// Report Count
$entityCount = $clusterMap->count();
echo "Database Contains: $entityCount records";
```

PhpOrient - `dropClusterID()`

This method removes the given cluster from the database using its Cluster ID.

Removing Clusters

In certain situations, you may find it useful to programmatically remove clusters from the database. For instance, in the case of a maintenance script that frees up space when the given cluster is no longer needed. This method allows you to remove clusters by their Cluster ID, which is the first numeral in the Record ID.

Syntax

```
$clusterMap->dropClusterID(<cluster-id>)
```

- `<cluster-id>` Defines the Cluster ID you want to remove.

Example

Consider the example of a web application that uses multiple in-memory clusters for short-term needs. For this purpose, you might want to create a function to quickly remove these clusters when you're finished using them.

```
// REMOVE CLUSTER
function removeCluster($clusterID){

    // Log Operation
    echo "Removing Cluster: $clusterID";

    // Fetch Global Cluster Map
    global $clusterMap;

    // Remove the Cluster
    $clusterMap->dropClusterID($clusterID);
}
```

PhpOrient - `getClusterID()`

This method returns the Cluster ID for the given name.

Retrieving Cluster ID's

In cases where you need or would like to operate on a cluster by its Cluster ID, you can retrieve it from the Cluster Map by passing the cluster name as an argument to this method.

Syntax

```
$clusterMap->getClusterID(<cluster-name>)
```

- **`<cluster-name>`** Defines the name of the cluster.

Example

For instance, say you want to retrieve records for clusters by cluster name, you might use this method in conjunction with `dataClusterDataRange()`.

```
// FETCH RECORDS
function fetchRecords($clusterName){

    // Log Operation
    echo "Retrieving Records on Cluster: $clusterName";

    // Fetch Globals
    global $client;
    global $clusterMap;

    // Find Cluster ID
    $clusterId = $clusterMap->getClusterID($clusterName);

    // Return Records
    return $client->dataClusterDataRange($clusterId);
}
```

PhpOrient - `[etIdList()`

This method returns a list of Cluster ID's on the database.

Retrieving Cluster ID's

Occasionally, you may want to work with a series of clusters in a given operation. Using this method, you can retrieve a list of Cluster ID's available on the database. You can then pass these ID's to other methods in performing further operations.

Syntax

```
$clusterMap->getIdList()
```

Example

In cases where you find yourself making frequent calls to all clusters on the database, you may find it convenient to write a simple function to fetch it and log the operation for you.

```
// FETCH CLUSTER ID'S
function fetchClusterIDs(){

    // Fetch Global Cluster Map
    global $clusterMap;

    // Fetch ID's
    $idList = $clusterMap->getIdList();
    $idCount = count($idList);

    // Log Operation
    echo "Retrieving $idCount Cluster ID's";

    // Return ID's
    return $idList;
}
```

PhpOrient - `Record()`

On occasion, you may want to operate a record or build one from scratch within your application, before adding it to or syncing it with OrientDB. In PhpOrient, these operations are controlled by the `Record()` class.

Working with Records

Record objects require no arguments to instantiate. Once created, you can call various methods on the object to define its class, Record ID, data and so on.

```
$record = new Record()
   ->setOClass('V')
   ->setRid(new ID(0))
   ->setOData(
      ['accommodation' => 'bungalow']);
```

Method	Description
`getOClass()`	Retrieve the class name
`getOData()`	Retrieves record data
`getRid()`	Returns the Record ID
`jsonSerialize()`	Returns record in serialized JSON format
`recordSerialize()`	Returns record in serialized format
`setOClass()`	Sets the class
`setOData()`	Sets record data
`setRid()`	Sets the Record ID

PhpOrient - `getOClass()`

This method retrieves the OrientDB class for the record.

Retrieving Classes

In cases where you want to access the class for the given `Record()` instance, this method returns the class name. You might find it useful in cases such as logging operations, where you would like to report the specific class rather than its Record ID.

Syntax

```
$record->getOClass()
```

Example

Consider the use-case of a logging operation tied to the callback function for an asynchronous query. For each record the query returns, it calls a function that logs the class that it's operating on.

```
// CALLBACK FUNCTION
function logReturn(Record $record){
    // Fetch Record ID
    $rid = $record->getRid()->_toString();

    // Fetch Class
    $class = $record-getOClass();

    // Log Operation
    echo "Retrieving $class Record: $rid";
}

// ASYNCHRONOUS QUERY
function runQuery($sql, $fetchPlan){

    // Fetch Global
    global $client;

    // Run Query
    $results = $client->queryAsync($sql,
        ['fetch_plan' => $fetchPlan,
        '_callback' => logReturn]);

    // Return Results
    return $results;
}
```

PhpOrient - `getOData()`

This method retrieves data from the record.

Retrieving Data

In cases where you want to operate on record data, this method allows you to retrieve the data from the `Record()` instance. It returns a mapped array of properties and values on the record.

Syntax

```
$record->getOData()
```

Example

Consider the use case of a web application that serves blogs. You might want to create a blog roll, which displays data on the most recent posts. You might want to create a function that standardizes the data retrieval process.

```
function blogRoll($limit){

    // Log Operation
    echo "Retrieving Blog Entries";

    // Retrieve Blogs
    $blogs = $client->query("SELECT FROM BlogEntry LIMIT $limit");

    $div = '<div id="blog-roll"><h3>Latest Posts</h3><ul>'

    foreach($blogs as $record){

        // Fetch Data
        $data = $record->getOData();
        $title = $data['title'];
        $link = $data['link'];

        $div = '$div <li><a src="$link">$title</a></li>';
    }
    $div = '$div</div>';

    // Return Blog Roll
    return $div;
}
```

PhpOrient - `getRid()`

This method returns the Record ID of the given Record object.

Retrieving Record ID's

In cases where you want to access the Record ID of a given `Record()` object, this method allows you to retrieve the `ID()` instance from the record. You can then use this to call additional methods on the Record ID in further operations.

Syntax

```
$record->getRid()
```

Example

Consider the use-case of a logging operation tied to an asynchronous query. As the query runs, each record triggers a callback function. Using this method, you can fetch the Record ID of each record to echo to the console.

```
// CALLBACK FUNCTION
function logReturn(Record $record) {

    // Fetch Record ID
    $id = $record->getRid();
    $rid = $id->_toString();

    // Log Operation
    echo 'Retrieved Record: $rid';
}

// ASYNCHRONOUS QUERY
function runQuery($sql, $fetchPlan){

    // Fetch Global
    global $client;

    // Run Query
    $results = $client->queryAsync($sql,
        ['fetch_plan' => $fetchPlan,
        '_callback' => logReturn]);

    // Return Results
    return $results;
}
```

PhpOrient - `jsonSerialize()`

This method returns the record serialized in JSON format.

Serializing Records

In some cases, you may find it more convenient to operate on a record in JSON format rather than the standard format provided by PhpOrient. Alternatively, you may want to save JSON instances of records for backup or logging purposes. This method takes no arguments and returns a JSON instance of the record. It is similar to the `recordSerialize()` method.

Syntax

```
$record->jsonSerialize()
```

Example

For instance, you might want to use this method as part of a logging operation, echoing a JSON instance of the record to the console on asynchronous queries.

```
// LOG QUERY
function logQuery(Record $record){

    // Fetch JSON
    $json = $record->jsonSerialize();

    // Log to Console
    echo "$json";
}

// ASYNC QUERY
function query($sql, $fetchPlan){

    // Query Database
    global $client;
    $results = $client->queryAsync($sql,
        ['fetch_plan' => $fetchPlan,
        '_callback' => logQuery ]);

    // Return Results
    return $results;
}
```

PhpOrient - `recordSerialize()`

This method returns a serialized instance of the given record.

Serializing Records

In cases where you want to pass the record instance to another application or would otherwise like to serialize the data, you can do so using this method. It takes no arguments and returns a serialized instance of the record. It is similar to the `jsonSerialize()` method, which serializes the record into JSON data.

Syntax

```
$record->recordSerialize()
```

Example

```
// FETCH SERIAL RECORD
function serialData(Record $record){

    // Fetch Serial Record
    $serial = $record->recordSerialize();
    echo "$serial";
}

// QUERY RECORDS
function serialRecordsQuery($sql, $fetchplan){

    // Query
    global $client;
    $results = $client->queryAsync($sql,
        ['fetch_plan' => $fetchPlan,
        '_callback' => serialData]

    // Return Results
    return $results;
}
```

PhpOrient - `setOClass()`

This method sets the class on the `Record()` object.

Setting Classes

When building records within your application this method allows you to define the class to which the record belongs in OrientDB.

Syntax

```
$record->setOClass(<class>)
```

- `<class>` Defines the class name.

Example

For instance, when creating a new `Record()` instance within your application, use this method to set the class.

```
function addEntry($data){

    // Log Operation
    echo "Creating Record";

    // Build Record
    $record = new Record()
        ->setOData($data)
        ->setRid(
            new ID(5))
        ->setOClass("entry");

    // Create Record
    global $client;
    $client->recordCreate($record);
}
```

PhpOrient - `setOData()`

This method sets data on a `Record()` object.

Setting Data

When building a record in your application to create or update on your database, this method is used to set the data object.

Syntax

```
$record->setOData(<data>)
```

- `<data>` Defines an array of the data you want to set on the record.

Example

In cases where you create records on a particular cluster or class frequently, you might find it convenient to hard core the feature into an creation function, requiring you to only set the specific data going into the record.

```
// CREATE RECORD
function addBlog($data){

    // Log Operation
    echo "Creating New Blog Entry";

    // Fetch Global Client
    global $client;

    // Build Record
    $record = new Record()
        ->setOData($data)
        ->setRid(
            new ID(5))
        ->setOClass("BlogEntry");

    // Create Record
    %client->recordCreate($record);
}
```

PhpOrient - `setRid()`

This method sets the Record ID for the `Record()` object.

Setting Record ID's

When instatitating a `Record()` object in your application, you may need to set the specific Record ID on the object before syncing it with OrientDB. You can also partially define the record, by setting the Cluster ID for the cluste you want to create it in. Once the `ID()` instance is ready, you can pass it to this method to set the Record ID on the ojbect.

Syntax

```
$record->setRid(<id>)
```

- **`<id>`** Defines the Record ID you want to set on the record. It is an instance of `ID()`.

Example

When creating new records in your application, you can use this method to define the Record ID or the Cluster ID for the new record before syncing it with the database. In cases where you do this often, you may want to use a dedicated function to ensure consistency with common values.

```
function addEntry($data){

    // Log Operation
    echo "Creating Record";

    // Build Record
    $record = new Record()
        ->setOData($data)
        ->setRid(
            new ID(5))
        ->setOClass("BlogEntry");

    // Create Record
    global $client;
    $client->recordCreate($record);
}
```

PhpOrient - `ID()`

This object provides an interface for operating on Record and Cluster ID's from within your PHP application.

Working with ID's

Certain methods and objects in PhpOrient take Record or Cluster ID's as an argument. In cases where you need to operate on these, use the `ID()` interface to generate the instance. It also provides methods for constructing and abstracting the ID from the interface.

Creating ID's

When creating a new instance of the `ID()` interface, there are a two approaches available to you: You can define the Cluster ID and Record ID using integers as arguments, or you can initialize the class without arguments as use the `parseString()` method to set the ID values from string. For instance,

```
// CREATE RID FROM INTEGERS
$clusterID = 5;
$recordID = 3
$rid = new ID($clusterID, $recordID);

// CREATE ID FROM STRING
$stringID = "#5:3";
$rid = new ID().parseString($stringID);
```

Using Cluster ID

In addition to these approaches, you can also partially instantiate an ID by only providing it with a Cluster ID. For instance,

```
$rid = new ID(5);
```

When you instantiate the `ID()` using this technique, it records the cluster you want to add the record to, but waits until you create the record on OrientDB before it sets the Record ID. In cases where you're working with multiple client connections, such as in a web application, this can help to avoid conflicts where two clients attempt to create a record with the same Record ID.

Retrieving Record ID

In cases where you want to extract the Record ID from a given `ID()` instance, for logging or other purposes, you can do so using the `__toString()` method.

```
// CREATE RECORD ID
$id = new ID(5, 3);

// FETCH RECORD ID
$rid = $id->__toString();
```

PhpOrient - Transactions

OrientDB supports transactions, allowing you to organize database operations into units of work that you can then commit or roll back as need. You may find this useful in cases where you want to provide reliable units of work to allow correct recovery from failures and keep the database consistent or you would like to provide isolation between clients accessing the database concurrently, such as in the case of a web application.

Using Transactions

Transactions are managed through a transaction interface. You can initialize this interface using the `getTransactionStatement()` method on the client interface. For instance,

```
// Fetch Transaction Interface
$tx = $client->getTransactionStatement();
```

Once you have this interface initialized, there are a series of methods you can call to begin transactions, attach database operations to the transaction, then commit the changes to the database or roll the changes back to the earlier database state.

Method	Description
`attach()`	Attach an operation to the transaction
`begin()`	Begin a transaction
`commit()`	Commit the transaction
`rollback()`	Roll the transaction back

Consider the example of a web application. You might want a dedicated function to handle common operations like updating records in the database. Using a global transaction interface, you can begin and commit transactions within the function.

```
// INITIALIZE TRANSACTION INTERFACE
$tx = $client->getTransactionStatement();

// UPDATE RECORD
function updateRecord($class, $data, $rid){

    // Log Operation
    echo "Updating Record";

    // Fetch Globals
    global $client;
    global $tx;

    // Begin Trasnaction
    $tx = $tx->begin();

    // Build Updated Record
    $record = new Record();
    $record->setOClass($class);
    $record->setOData($data);
    $record->setRid($rid);

    // Update Database
    $update = $client->recordUpdate($record);

    // Attach Operation to Transaction
    $tx->attach($update);

    // Commit Changes
    return $tx->commit();
}
```

> For more information, see Transactions.

PhpOrient - `attach()`

This method attaches the given database operation to a transaction.

Attaching Operations

In building transactions, you can attach specific operations to the given transaction. This allows you to later commit the transactions to the database or roll the changes back to an earlier state.

Syntax

```
$tx->attach(<operation>)
```

- `<operation>` Defines the operation to attach.

Example

For instance, imagine a web application that handles blog entries. With multiple users connecting to the application and attempting to update the database. Using transactions, you can isolate the changes they make to the database to help avoid conflicts.

```
// CREATE RECORDS
function createRecord($class, $records){

    // Log Operation
    echo "Creating Record";

    // Fetch Global Variables
    global $client;
    global $tx;

    // Begin Transaction
    $tx = $tx->begin()

    // Loop Over Record Data
    foreach($records as $record){

        // Create Record
        $createdRecord = $client->recordCreate(
            (new Record())
                ->setOClass($class)
                ->setOData($data)
                ->setRid(new ID())
        );

        // Attach Operation to Transaction
        $tx->attach($createdRecord);
    }

    // Commit Changes
    $tx->commit();
}
```

PhpOrient - `begin()`

This method begins a transaction.

Beginning Transactions

Once you have the transaction interface initialized through the `getTransactionStatement()` client interface method, using this method you can initialize a transaction statement, using the other methods to attach and commit or revert the changes as need.

Syntax

```
$tx = $tx->begin()
```

Example

Consider the use-case of a web application in which you frequently update records as part of a transaction. You might use a function similar to this to handle both the transaction and update operations together.

```
// INITIALIZE TRANSACTION INTERFACE
$tx = $client->getTransactionStatement();

// UPDATE RECORD
function updateRecord($class, $data, $rid){

    // Log Operation
    echo "Updating Record";

    // Fetch Globals
    global $client;
    global $tx;

    // Begin Trasnaction
    $tx = $tx->begin();

    // Build Updated Record
    $record = new Record();
    $record->setOClass($class);
    $record->setOData($data);
    $record->setRid($rid);

    // Update Database
    $update = $client->recordUpdate($record);

    // Attach Operation to Transaction
    $tx->attach($update);

    // Commit Changes
    return $tx->commit();
}
```

PhpOrient - `commit()`

This method commits operations attached to the transaction to the database.

Committing Transaction

When you're finished with a transaction and satisfied with the changes you've made to it, you need to commit the changes to the database to make them persistent. This method commits all attached operations.

Syntax

```
$tx->commit()
```

Example

In the event that you work with transactions often, you might want to put together a routine function to handle commit and rollback operations.

```
// TX HANDLER
function transactionOp($testResult){

    // Fetch Globals
    global $tx;

    if($testResult){

        // Log Operation
        echo "Commiting Transaction";

        // Commit
        $tx->commit();
    } else {

        // Log Operation
        echo "Rolling Back Transaction";
        $tx->rollback();
    }

}
```

Whenever you're finished with a transaction test it with a boolean operation, then pass the results to this function. If the test passed, the if statement within this function commits the transaction. If it failed, it rolls the database back the attached operations.

PhpOrient - `rollback()`

This method reverts changes made on the current transaction.

Rolling Back Transactions

In certain situations you may encounter an issue where you need to revert a series of changes already made to the database. When working with transactions, you can manage this by rolling the database back to its state when the transaction was begun.

Syntax

```
$tx->rollback()
```

Example

In the event that you work with transactions often, you might want to put together a routine function to handle commit and rollback operations.

```
// TX HANDLER
function transactionOp($testResult){

    // Fetch Globals
    global $tx;

    if($testResult){

        // Log Operation
        echo "Commiting Transaction";

        // Commit
        $tx->commit();
    } else {

        // Log Operation
        echo "Rolling Back Transaction";
        $tx->rollback();
    }

}
```

Whenever you're finished with a transaction test it with a boolean operation, then pass the results to this function. If the test passed, the if statement within this function commits the transaction. If it failed, it rolls the database back the attached operations.

MarcoPolo: Elixir with OrientDB

OrientDB supports a number of application programming interfaces, natively through the JVM and externally through Java wrappers and the Binary Protocol. In the event that you need or would prefer to develop your OrientDB application Erlang VM language Elixir, you can do so through the driver MarcoPolo.

> **NOTE**: OrientDB Documentation for MarcoPolo is currently in development. If you have a question that is not currently addressed here, please consult the project documentation.

Getting MarcoPolo

In order to use MarcoPolo in your Elixir application, you first need to register it as a dependency for the Mix build tool. To do so, edit the `mix.exs` file in the project root directory and add MarcoPolo as a dependency to the `deps` function.

```
# Project Dependencies
def deps do
    [{:marco_polo, "~> 0.1"}]
end
```

With this line added, Mix now knows that your application requires MarcoPolo. To retrieve and compile the package, use Mix:

```
$ mix deps.get
$ mix deps.compile
```

When you run the `deps.get` command, Mix calls the Hex package manager to fetch the MarcoPolo package and its dependencies. The second command then compiles these packages for use with your application. You can see that the dependencies were installed without error by calling Mix again with the `deps` argument:

```
$ mix deps
...
* marco_polo 0.2.2 (Hex package) (mix)
  locked at 0.2.2 (marco_polo) 40x47150
  ok
```

Using MarcoPolo

Once you have registered MarcoPolo as a dependency with Mix you can begin to use it in developing your Elixir application.

```
@doc """ Connects to OrientDB Server """
def connect_orientdb_server(user, passwd) do
    {:ok, conn} = MarcoPolo.start_link(
        user: user, password: passwd, connection: :server)
end
```

MarcoPolo - Server Operations

OrientDB differentiates between connections made to the database and connections made to the server. While you can get both from the `start_link()` function in MarcoPolo, which you choose determines the kinds of operations you can perform. Server operations including creating and dropping databases, and checking that they exist.

Connecting to the Server

In order to operate on the server, you first need to connect to it. To manage this, call the `start_link()` function, then pass `:server` to the `connection` parameter. For instance.

```
@doc """ Connect to OrientDB Server """
def orientdb_server(user, passwd) ::
    {:ok, conn } = MarcoPolo.start_link(
        user: user, password: passwd, connection: :server)
end
```

Using Server Connections

When you call the `start_live()` function, it returns a connection interface to your application, which by convention is called `conn` here. With the connection interface initialized to your OrientDB Server, there are a number of operations that become available by passing the `conn` value to MarcoPolo functions.

```
@doc """ Check if Database Exists, Create it if it does not """
def create_db(conn, dbname, type) when type in [:plocal, :memory] do

    # Check Database Existence
    unless MarcoPolo.db_exists?(conn, dbname, type) do

        # Log Operation
        IO.puts("Creating Database: #{dbname}")

        # Create Database
        MarcoPolo.db_create(conn, dbname, :document, type)

    end
end
```

Function	Description
`create_db()`	Creates a database.
`db_exists?()`	Determines if database exists.
`distrib_config()`	Fetches distributed server configuration.
`drop_db()`	Removes a database.
`stop()`	Closes the server connection.

Closing Connections

When you are finished with a connection, whether the connection interface was opened on a database or a server, you can close the connection and free up resources by calling the `stop()` function.

For instance, consider the use case of a web application that operates on several databases on an OrientDB server. You might want a function to loop over a series of database names, determining whether or not they exist and creating them where they don't, then close the connection once this operation is complete.

```
@doc """ Takes connection and list of databases and creates those that don't exist. """
def create_databases(user, passwd, databases) do

    # Open Connection
    IO.puts("Connecting to OrientDB Server")
    {:ok, conn} = MarcoPolo.start_link(
        user: user,
        password: passwd,
        connect: :server)

    # Loop Over Databases
    for db <- databases do

        # Check Existence
        IO.puts("Database Test: #{db}")
        unless MarcoPolo.exists(conn, db, :graph, :plocal) do

            # Create Database
            IO.puts("Database #{db} not found, creating...")
            MarcoPolo.create_db(conn, db, :graph, :plocal)

        end
    end

    # Close Connection
    IO.puts("Closing Server Connection")
    MarcoPolo.stop(conn)

end
```

MarcoPolo - `create_db()`

This function creates a database on the connected OrientDB Server.

Creating Databases

While it is possible to connect your Elixir application to an existing database on the OrientDB Server, it is more likely that you'll want to create databases dynamically from within your application. The `create_db()` takes the connection interface, database name and database options, then creates a database on the Server. In the event that there is a problem it returns an error message.

Syntax

```
create_db(<conn>, <name>, <database-type>, <storage type>, <opts>)
```

- **`<conn>`** Provides the server connection.
- **`<name>`** Defines the database name.
- **`<database-type>`** Defines the database-type. Supported types are,
  - *`:document`* Creates a Document database.
  - *`:graph`* Creates a Graph database.
- **`<storage type>`** Defines the storage-type. Supported types are,
  - *`:plocal`* Sets to the PLocal storage-type.
  - *`:memory`* Sets to the in-memory storage-type.
- **`<opts>`** Used for any additional keyword options passed to the function.

Options

This function supports one additional option:

- **`:timeout`** Defines the timeout value in milliseconds. In the event that the operation takes longer than the allotted time, MarcoPolo sends an exit signal to the calling process.

Return Values

When the function is successful, it returns `:ok`. In the event that there is a problem, it returns `{:error, term}`, where `term` is the error message.

Example

For instance, you might want a function that integrates the `create_db()` function with the `db_exists?()` function. This would allow you to create a new database only in cases where one of the same name and type does not already exist.

```
@doc """ Check if Database Exists, Create it if it does not """
def create_db(conn, dbname, type) when type in [:plocal, :memory] do

    # Check Database Existence
    unless MarcoPolo.db_exists?(conn, dbname, type) do

        # Log Operation
        IO.puts("Creating Database: #{dbname}")

        # Create Database
        MarcoPolo.db_create(conn, dbname, :document, type)

    end
end
```

MarcoPolo - `db_exists?()`

This function retunrs a boolean value indicating whether the given database exists on the server.

Checking Database Existence

Occasionally, you may encounter issues where you aren't certain whether a particular database exists on a server. This might come up in distributed deployments or similar situations where you have many OrientDB Servers running in different data centers used for various purposes. This function checks with OrientDB to confirm whether the given database exists on the server.

Syntax

```
db_exists?(<conn>, <database>, <storage-type>, <opts>)
```

- `<conn>` Defines the server connection.
- `<database>` Defines the database name.
- `<storage-type>` Defines the storage type. There are two storage types supported:
  - `:plocal` Sets the storage-type to PLocal storage.
  - `:memory` Sets the storage-type to in-memory storage.
- `<opts>` Defines additional options to pass to the function. For more information on available options, see the Options section below.

Options

This function supports one additional option:

- `:timeout` Defines the timeout value in milliseconds. In the event that the operation takes longer than the allotted time, MarcoPolo sends an exit signal to the calling process.

Return Value

When the operation is successful, the function returns the tuple `{:ok, exists}`, where the variable is a boolean value indicating whether or not the database exists on the server. In the event that the operation fails, the it returns the tuple `{:error, message}`, where the variable provides the exception message.

Example

For instance, you might want a functin that integrates the `create_db()` function with `db_exists?()` function. This would allow you to create a new database only in cases where one of the same name and type does not already exist.

```
@doc """ Check if Database Exists, Create it if it does not """
def create_db(conn, dbname, type) when type in [:plocal, :memory] do

    # Check Database Existence
    unless MarcoPolo.db_exists?(conn, dbname, type) do

        # Log Operation
        IO.puts("Creating Database: #{dbname}")

        # Create Database
        MarcoPolo.db_create(conn, dbname, :document, type)

    end
end
```

MarcoPolo - `distrib_config()`

This function returns the distributed configuration of the OrientDB Server.

Working with Distributed Deployments

In deployments where high availability is a concern, OrientDB can run distributed across several servers, allowing you to achieve the maximum in performance, scalability and robustness possible.

When using this deployment architecture, the OrientDB Server pushes data to clients whenever changes occur in the distributed configuration. In the event that you would like to review or operate on this configuration from within your Elixir application, this function allows you to retrieve it as a `Document` instance.

> For more information on managing these deployments, see Distributed Architecture.

Syntax

```
distrib_config(<conn>, <opts>)
```

- `<conn>` Defines the database connection.
- `<opts>` Defines any additional options you would like to pass to the function. For more information on the available options, see the Options section below.

Options

This function supports one additional option:

- `:timeout` Defines the timeout value in milliseconds. In the event that the operation takes longer than the allotted time, MarcoPolo sends an exit signal to the calling process.

Return Values

This operatation returns a Document instance containing the current state of the distributed configuration.

MarcoPolo - `drop_db()`

This function removes the given database from the server.

Removing Databases

Occasionally, you may find yourself in situations where you need to remove an entire database from the server. For instance, in cases where your application requires volatile in-memory databases for short-term use. Using this function, you can remove a database from the server.

Syntax

```
drop_db(<conn>, <database>, <storage-type>, <opts>)
```

- **`<conn>`** Defines the server connection.
- **`<database>`** Defines the database name.
- **`<storage-type>`** Defines the database storage-type. Supported storage-types include,
  - *`:plocal`* Sets it to the PLocal storage-type.
  - *`:memory`* Sets it to the Memory storage-type.
- **`<opts>`** Defines additional options. For more information on the available options, see the Options section below.

Options

This function supports one additional option:

- **`:timeout`** Defines the timeout value in milliseconds. In the event that the operation takes longer than the allotted time, MarcoPolo sends an exit signal to the calling process.

Return Values

When the operatin is successful, this function returns `:ok`. In the event that it encounters an error, instead it returns the tuple `{:error, message}`, where the variable is the exception message.

Example

Consider the use case of an application that operates with multiple databases in-memory. These databases are ad hoc, brought online to address short term needs and then removed when the job is complete.

```
@doc """ Create ad hoc Databases """
def adhoc_db(conn, name, operation)

    # Create Database
    try do

        # Create Temporary Database
        MarcoPolo.create_db(conn, name, :graph, :memory)

        # Create Database Connection
        dbconn = MarcoPolo.start_link("admin", "admin_passwd", connection: {:db, name})

        # Call Operation
        operation(dbconn)

    after
        # Remove Temporary Database
        MarcoPolo.drop_db(conn, name, :memory)

    end
end
```

MarcoPolo - Database Operations

OrientDB differentiates between connections made to the database and connections made to the server. While you can get both from the `start_link()` function in MarcoPolo, which you choose determines the kinds of operations you can perform. Database operations

Connecting to the Database

In order to operate on a database, you first need to connnect to it. To manage this, call the `start_lik()` function with your database credentials, then pass the database name to the `connection:` parameter. For instance,

```
@doc """ Connect to the given database using default credentials. """
def orientdb_database(user, passwd, dbname) ::
    {:ok, conn} = MarcoPolo.start_link(
        user: user, password: passwd,
        connect: {:db, dbname})
end
```

This function returns a connection interface that you can then pass to other MarcoPolo functions in performing further operations.

Using Database Connections

When you call the `start_link()` function, it returns a connection interface to your application, which by convention is called `conn` here. With the connection interface initialized to a particular database, there are a number of operations you can begin to call on the database, by passing the `conn` value to MarcoPolo functions.

```
@doc """ Report the size of the connected database to stdout """
def size_database(conn) do

    # Fetch Size
    size = MarcoPolo.db_size(conn)

    # Log Size
    IO.puts("Database Size: #{size}")
end
```

Function	Description
`command()`	Executes a query or command on the database.
`create_record()`	Creates a record.
`db_countrecords()`	Returns the number of records in the database.
`db_reload()`	Reloads the database.
`db_size()`	Returns the size of the database.
`delete_record()`	Removes the given record from the database.
`live_query()`	Subscribes to a live query.
`live_query_unsubscribe()`	Unsubscribes from a live query.
`load_record()`	Loads a record into your application.
`script()`	Executes a script on the database in the given language.
`stop()`	Closes the database connection.
`update_record()`	Updates the given record.

Closing Connections

When you're finished using a connection, you can close it to free up resources by calliing the `stop()` function and passing it the connection argument.

For instance, imagine that you have a series of scripts that you need to run on an OrientDB database. These scripts are used to perform a series of routine maintenance and logging tasks each night, to prepare reports for administrators the following morning. You might want to set up a dedicated function to manage such operations:

```
@doc """ Runs nightly maintenance scripts """
def maint_nightly(dbname, user, passwd, scripts) do

    # Log Operation
    IO.puts("Running Nightly Maintenance Scripts")

    # Open Connection
    IO.puts("Opening Database: #{dbname}")
    {:ok, conn } = MarcoPolo.start_link(
        user: user,
        password: passwd,
        connect: {:db, dbname})

    # Loop Over Each Script
    for script <- scripts do

        # Execute Script
        case MarcoPolo.script(conn, "JavaScript", script) do
            {:ok, record} -> IO.puts("Script Successful")
            {:error, reason} -> IO.puts("Error: #{reason}")
        end

    end

    # Close Database when Done
    IO.puts("Closing Database: #{dbname}")
    MarcoPolo.stop(conn)

end
```

Bear in mind, the `stop()` function returns `:ok` regardless of whether or not it was successful in closing connection.

MarcoPolo - `command()`

This function executes a query or command on the database.

Sending Commands

OrientDB SQL differentiates between idempotent queries (such as `SELECT` and non-idempotent commands, such as `INSERT`. The MarcoPolo Elixir API does not make this distinction, providing this function for use with both queries and commands..

Syntax

```
command(<conn>, <query>, <opts>)
```

- **`<conn>`** Defines the database connection.
- **`<query>`** Defines the query or command you want to execute.
- **`<opts>`** Defines additional options to set on the command. For more information, see options below.

Options

When issuing queries or commands using this function, there are a series of additional options available to you to further define how MarcoPolo performs the operation.

- **`:params`** Defines a map of parameters for OrientDB to use in building prepared statements. The map must use atoms or strings as keys, but can take any encodable term as values.

  It defaults to `#{}`

- **`:timeout`** Defines the timeout value in milliseconds. In the event that the query takes longer than the allotted time, MarcoPolo sends an exit signal to the calling process.

- **`:fetch_plan`** Defines a fetch plan, which is only available when using this function with idempotent queries. It is a mandatory argument with fetch queries.

Return Values

When the query or command is successful it returns the tuple `{:ok, values}`. The `values` variable is a map with the following keys:

- **`:response`** Provides the return value given by OrientDB. This varies depending the query. For instance, `SELECT` returns a list of records, `CREATE CLUSTER` returns the new cluster's Cluster ID.

- **`:linked_records`** Provides a set of additional records fetched by OrientDB. The `:fetch_plan` option controls the number of records retrieved to this value.

In the event that the query or command fails, it returns the tuple `{:error, term}`. The `term` variable provides the error message.

Examples

Non-idempotent Commands

For instance, in cases where you find yourself making frequent insertions of a particular class, you might want to set up a function to streamline this process and to make it easier to insert a series of records through a single function call.

```
@doc """Inserts records from array into the given class"""
def insert_records(conn, class, properties, records) do

    # Log Operation
    IO.puts("Adding Records to #{class}")

    # Build Initial INSERT Statement
    joined_prop = Enum.join(properties, ",")
    insert = "INSERT INTO #{class} (#{joined_prop}) "

    # Loop over Given Records
    for record <- records do

        # Complete INSERT Statement
        data = Enum.join(record, ", ")
        sql = "#{insert} VALUES(#{data})"

        # Issue INSERT Statement
        MarcoPolo.command(conn, sql)

    end
end
```

Idempotent Queries

You might find yourself in similar situations when working with idempotent queries. For instance, if you frequently query the same class on the database with close or identical options passed to the query, you can save yourself some time by standardizing the process in a function.

```
@doc """ Retrieve User Profile """
def fetch_profile(conn, user) do

    # Log Operation
    IO.puts("Retrieving Profile for User ID: #{uid}"

    # Build SELECT Statement
    select = "SELECT FROM :class WHERE userId=':uid'"

    # Query Database
    options = [params: %{uid: user, class: "Profile"}, fetch_plan: "*:-1"]
    MarcoPolo.command(conn, select, options)
end
```

MarcoPolo - `create_record()`

This method create a record on the database, using the given data.

Creating Records

When you want to create new records, you can build `Document` instances within your Elixir application code and pass it to this method to add it to the database.

Syntax

```
create_record(<conn>, <cluster-id>, <data>, <opts>)
```

- **`<conn>`** Defines the database connection.
- **`<cluster-id>`** Defines the Cluster you want to use.
- **`<data>`** Defines the data to use in the record.
- **`<opts>`** Defines optional options to pass to the function. For more information on the available options, see Options below.

Options

This function takes the following options:

- **`:no_response`** Defines whether the function should wait on a response from the OrientDB Server. If set `false`, it returns `:ok` regardless of whether or not the operation was successful.

- **`:timeout`** Defines the timeout value in milliseconds. In the event that the query takes longer than the alloted time, MarcoPolo sends an exit signal to the calling process.

Return Values

When the function is successful, it returns the tuple `{:ok, record_info}`. The `record_info` value is a tuple containing the Record ID and record version. In the event that the record creation fails, it returns `{:error, term}`. The `term` variable provides the error message.

Example

For instance, consider the use case of a web application. You might want a function to streamline adding new blog entries to the database.

```
@doc """Takes given data and add new blog entry to database"""
def add_blog(conn, author, title, text) do

    # Log Operation
    IO.puts("Adding Blog for User: #{author}")

    # Build Document
    record = %MarcoPolo.Document{class: "BlogEntry",
        fields: %{
            "title" => title,
            "author" => author,
            "text": => text}}

    # Create Record
    case MarcoPolo.create_record(conn, 15, record) do
        {:ok, {record_id, version}} -> IO.puts("Record Created")
        {:error, reason} -> IO.puts("Error: #{reason}")

    end
end
```

MarcoPolo - `db_countrecords()`

This function returns the number of records in a database.

Counting Records

In certain situations you may find it useful to size the database or to count the number of records it contains. The `db_size()` function returns the size. This function returns the database record count.

Syntax

```
db_countrecords(<conn>, <opts>)
```

- `<conn>` Defines the database connection.
- `<opts>` Defines additional options. For more information on the available options, see Options.

Options

This function only provides one additional option:

- `:timeout` Defines the timeout value in milliseconds. In the event that the operation takes longer than the allotted time, MarcoPolo sends an exit signal to the calling process.

Return Values

When this function is successful it returns the tuple `{:ok, count}`, where `count` is a non-negative integer indicating the number of records in the connected database. In the event that the function fails, it returns the tuple `{:error, reason}`, where the reason is the exception message the function receives.

Example

For instance, consider the use case of a logging operation. Whenever you connect to the database it logs the record count to the console.

```
@doc """ Open Database """
def connect() do

    # Log Operation
    IO.puts("Connecting to Database")

    # Connect to Database
    {:ok, conn} = MarcoPolo.start_link(
            user: "admin",
            password: "admin_passwd",
            connection: {:db, "blog"})

    # Log Number of Records
    count = MarcoPolo.db_countrecords(conn, {:timeout 5000})
    IO.puts("Record Count: #{count}")

    # Return Connection
    {:ok, conn}

end
```

MarcoPolo - `db_reload()`

This function reloads the database connection.

Reloading Databases

Occasionally, you may need to reload the database connection.

Syntax

```
db_reload(<conn>, <opts>)
```

- `<conn>` Defines the database connection.
- `<opts>` Defines additional options for the function.

Options

This function can take one additional option.

- `:timeout` Defines the timeout value in milliseconds. In the event that the reload operate takes longer than the alloted time, MarcoPolo sends an exit signal to the calling process.

Example

Consider the use case of a multithreaded application or some similar deployment in which multiple clients are operating on the given OrientDB database. You might want to set up a basic timer that reloads the database connection periodically from a separate thread.

```
@doc """ Reload the database connection at set intervals"""
def reload(conn, interval) do

    # Log Operataion
    IO.puts("Starting Reload Function")

    # Start Timer
    timer.apply_interval(interval, MarcoPolo, db_reload, conn)

end
```

MarcoPolo - `db_size()`

This function returns the size of the database.

Sizing the Database

In certain situations you may find it useful to size the database or to count the number of records it contains. The `db_countrecords()` function returns the record count. This function returns the size.

Syntax

```
db_size(<conn>, <opts>)
```

- `<conn>` Defines the database connection.
- `<opts>` Defines additional options for the function.

Options

This function provides only one additional option:

- `:timeout` Defines the timeout value in milliseconds. In the event that the operation takes longer than the allotted time, MarcoPolo sends an exit signal to the calling process.

Return Value

When this function is successful, it returns the tuple `{:ok, size}`, where `size` is a non-negative integer indicating the database size. In the event that the operation fails, the function returns the tipe `{:error, reason}`, where `reason` contains the exception message.

Example

For instance, consider the use case of a logging operation. Whenever you close a database connection, you would like to log the size of the database at the time it was closed, to check against later.

```
@doc """ Close the Database """
def close_database(conn) do

    # Log Operation
    IO.puts("Closing Database")

    # Fetch Size
    size = MarcoPolo.db_size(conn)
    IO.puts("Database Size: #{size}")

    # Close Database
    MarcoPolo.stop(conn)
    IO.puts("Database Closed")

end
```

MarcoPolo - `delete_record()`

This method removes a record from the database, identified by its Record ID.

Removing Records

Occasionally, you may find you need to remove records from the database and have to decide which from within your Elixir application. This function allows you to do so, removing records by their Record ID's.

Syntax

```
delete_record(<conn>, <record-id>, <version>, <opts>)
```

- **`<conn>`** Defines the database connection.
- **`<record-id>`** Defines the Record ID, as an `RID` instance.
- **`<version>`** Defines the record version.
- **`<opts>`** Defines additional options for the function. For more information, see the Options section below.

Options

This functin provides two additional options:

- **`:no_response`** Defeines whether you want your application to wait for a response from OrientDB. When set to `true`, it returns `:ok` on every operation, regardless of whether it's successful.

- **`:timeout`** Defines the timeout value in milliseconds. In the event that the query takes longer than the alloted time, MarcoPolo sends an exit signal to the calling process.

Return Values

When this operation is successful, the function returns the tuple `{:ok, passed}`, where the `passed` variable is a boolean value indicating whether the record was successfully deleted. In the event that the operation fails, it returns the tuple `{:error, message}`, where the variable contains the exception message.

Example

Consider the use case where you need to remove a series of records from the database. Rather than calling `delete_record()` individually on each instance, you mght want to create a function to handle the deletions.

```
@doc """Function to remove records from the database. It takes as arguments
the database connection interface and a list of tuples indicating
the records to remove.  Each tuple follows the pattern {cluster-id, [list of record id's]}."""
def remove_records(conn, record_list) do

    # Log Operation
    IO.puts("Remvoing Records")

    # Loop over Clusters
    for cluster <- record_list do

        # Fetch Cluster ID
        cluster_id = cluster.id

        records = cluster.records

        # Loop Over Records
        for record <- records do

            # Construct RID
            rid = MarcoPolo.RID(cluster_id, record)
            # Remove Records
            MarcoPolo.delete_record(conn, rid)

        end
    end
end
```

MarcoPolo - `live_query()`

This function subscribes to a live query

Subscribing to Live Queries

When you issue queries using the stnadrd `command()` function, what it returns is effectively a snapshot of the records in the state they held when the query was issued. In the event that another client modifies these records, there's no way you'll know unless you reissue the query.

To get around this limitation, OrientDB provides Live Queries. Instead of returning records, these queries return a subscription token. When the records assigned to this token recieve an update, OrientDB pushes the changes to the given receiver function.

This function subscribes to live queries. When you're finished with the live query, you can call the `live_query_unsubscribe()` function to deregister the token.

Syntax

```
live_query(<conn>, <query>, <receiver>, <opts>)
```

- **`<conn>`** Defines the database connection.
- **`<query>`** Defines the query.
- **`<receiver>`** Defines the function to receive the records
- **`<opts>`** Defines additional options for the function. For more information, see Options.

Options

This function can take one additional option:

- **`:timeout`** Defines the timeout value in milliseconds. In the event that the operation takes longer than the allotted time, MarcoPolo sends an exit signal to the calling process.

Return Value

When this operation is successful, it returns the tuple `{:ok, token}`, where the variable is a subscription token, which OrientDB uses to register any changes made to the records the query returns. You can use this token with the `live_query_unsubscribe()` function when you're ready to unsubscribe.

In the event that the operation fails, it returns `{:error, message}`, where the variable provides the exception message.

Example

Imagine you have an application that handles monitoring for various environmental sensors. Every fifteen minutes your application calls a series of functions to update the OrientDB database. You might use a Live Query to test whether the sensor reading has met an alert condition, causing the application to send emails or text messages to on-call operators. After a given timeout interval, the query resets itself, unsubscribing from the given Live Query.

```
@doc """ Check Readings for Alert Conditions """
def check_readings(record) do

    # Retrieve Data
    data = record.fields["reading"]

    if data >= threshold do

        # Log Alert
        IO.puts("Alert Condition: #{data}")

        # Call Notification Function
        notify_operator(record)

    end
end

@doc """ Handler for monitor function check_readings() """
def read_handler(conn, sensor, interval) do

    # Log Operation
    IO.puts("Initializing #{sensor} Monitor")

    # Call Live Query
    {:ok, token } -> MarcoPolo.live_query(conn,
        "LIVE SELECT FROM Sensors WHERE sensor_name = '#{sensor}'",
        check_readings)

    # Wait Interval
    Process.sleep(interval)

    # Unsubscribe and Restart Monitor
    MarcoPolo.live_query_unsubscribe(token)

    # Recursive Restart
    read_handler(conn, sensor, interval)

end
```

MarcoPolo - `live_query_unsubscribe()`

This function unsubscribes a receiver from a live query.

Unsubscribing from Live Queries

When you issue queries using the standard `command()` function, what it returns is effectively a snapshot of the records in the state they held when the query was issued. In the event that another client modifies these records, there's no way you'll know unless you reissue the query.

To get around this limitation, OrientDB provides Live Queries. Instead of returning records, these queries return a subscription token. When the records assigned to this token receive an update, OrientDB pushes the changes to the receiver function.

To subscribe to a live query in your Elixir application, use the `live_query()` function. This function unsubscribes to the live query.

Syntax

```
live_query_unsubscribe(<conn>, <token>)
```

- **`<conn>`** Defines the database connection.
- **`<token>`** Defines the live query token.

Return Values

This function only returns the value `:ok`.

Example

Imagine you have an application that handles monitoring for various environmental sensors. Every fifteen minutes your application calls a series of functions to update the OrientDB database. You might use a Live Query to test whether the sensor reading has met an alert condition, causing the application to send emails or text messages to on-call operators. After a given timeout interval, the query resets itself, unsubscribing from the given Live Query.

```
@doc """ Check Readings for Alert Conditions """
def check_readings(record) do

    # Retrieve Data
    data = record.fields["reading"]

    if data >= threshold do

        # Log Alert
        IO.puts("Alert Condition: #{data}")

        # Call Notification Function
        notify_operator(record)

    end
end

@doc """ Handler for monitor function check_readings() """
def read_handler(conn, sensor, interval) do

    # Log Operation
    IO.puts("Initializing #{sensor} Monitor")

    # Call Live Query
    {:ok, token } -> MarcoPolo.live_query(conn,
        "LIVE SELECT FROM Sensors WHERE sensor_name = '#{sensor}'",
        check_readings)

    # Wait Interval
    Process.sleep(interval)

    # Unsubscribe and Restart Monitor
    MarcoPolo.live_query_unsubscribe(token)

    # Recursive Restart
    read_handler(conn, sensor, interval)

end
```

MarcoPolo - `load_record()`

This function loads a record from the database.

Loading Records

Eventually, you'll want to operate on records from within your Elixir application. Using this function, you can retrieve records from the database by their Record ID's.

Syntax

```
load_record(<conn>, <record-id>, <opts>)
```

- `<conn>` Defines the database connection.
- `<record-id>` Defines the Record ID.
- `<opts>` Defines addtional options for the function. For more information on the available options, see the Options section below.

Options

This function supports a series of additional options defined through the final argument.

- `:fetch_plan` Defines a fetch plan.
- `:ignore_cache` Defines whether you want to ignore the cache. It defaults to `true`.
- `:load_tombstones` Defines whether you want to load information on deleted records. It defaults to `false`.
- `:if_version_not_latest` Defines whether you want to load the record in cases where the provided version is not the latest. This functionality was introduced in version 2.1 of OrientDB.
- `:version` Defines the record version that you want to load.
- `:timeout` Defines the timeout value in milliseconds. In the event that the query takes lnger than the allotted time, MarcoPolo sends an exit signal to the calling process.

Return Values

When the operation is successful, the function returns the tuple `{:ok, response}`. The variable itself is a tuple that contains two elements: the loaded record and the set of records linked to it. You can control the number and depth of linked records using the `:fetch_plan` option.

In the event that the operation fails, it returns the tuple `{:error, message}`, where the variable provides the exception message.

Example

Picture an environmental monitoring application where various sensors on write to particular clusters in your database. For each sensor you have a record defining what it monitors and where it is located, and edges that connect to separate vertices noting the readings taken at particular times. You might use a function like the below to retrieve histories on particular sensors.

```
@doc """ Retrieves sensor data """
def fetch_data(conn, cluster, id) do

    # Log Operation
    IO.puts("Loading Records")


    # Create Record ID
    rid = MarcoPolo.RID( cluster_id: cluster, position: id)

    # Define Options
    options = [fetch_plan: "*.-1"]

    # Load Record
    MarcoPolo.load_record(conn, rid, options)

end
```

MarcoPolo - `script()`

This method executes a script on the database.

> **NOTE**: In order for this to work, you must first enable scripting on OrientDB. For more information, see JavaScript.

Scripting the Database

OrientDB provides support for scripting operations in JavaScript and other languages. This allows you to develop a repository of common operations or to share functions between applications developed in different languages.

Syntax

```
script(<conn>, <language>, <script>, <opts>)
```

- `<conn>` Defines the database connection.
- `<language>` Defines the language of the script.
- `<script>` Defines the script to execute.
- `<opts>` Defines additional options for the function.

Options

This function only provides on additional option:

- `:timeout` Defines the timeout value in milliseconds. In the event that the script takes longer than the allotted time to finish, MarcoPolo sends an exit signal to the calling process.

Return Values

When the script is successful, it returns the tuple `{:ok, record}`, where the variable is the last record the script operates on. In the event that the operation fails, it returns the tuple `{:error, message}` where the variable is the exception message.

Example

For instance, imagine you have a series of stored scripts to handle common operations on OrientDB. In porting your current application to Elixir, you may want to save these scripts and continue to use them.

```
@doc """ Run the given script on OrientDB"""
def run_script(conn, path) do

    # Log Operation
    IO.puts("Running Script #{path}")
    {:ok, script} = IO.read(path)

    # Run Script
    {:ok, return} = MarcoPolo.script(conn, "JavaScript", script)

end
```

MarcoPolo - `update_record()`

This function updates a record on the database.

Updatating Records

When you're finished making changes to records loaded into your Elixir application, this function allows you to use the new data to update existing records on the database.

Syntax

```
update_record(<conn>, <record-id>, <version>,
              <data>, <update-content>, <opts>)
```

- **`<conn>`** Defines the database connection.
- **`<record-id>`** Defines the Record ID.
- **`<version>`** Defines the record version.
- **`<data>`** Defines the data you want to add.
- **`<update-content>`** Defines whether you want to update the content.
- **`<opts>`** Defines additional options for the function. For more information on the available options, see the Options section below.

Options

This function supports two additional options:

- **`:no_response`** Defines whether you want your application to wait for a response from OrientDB. When set to `true`, it sends the update and returns `:ok` regardless of whether the operation was successful.
- **`:timeout`** Defines the timeout value in milliseconds. In the event that the update takes longer than the allotted time, MarcoPolo sends an exit signal to the calling process.

Return Values

When the operation is successful, the function returns the tuple `{:ok, version}`, where the variable is a non-negative integer indicating the updated version number on the record. In the event that the operation fails, it returns the tuple `{:error, message}`, where the variable is the exception message.

Example

For instance, consider the use case of a web application. You might want a function to streamline updating blog entires on the database.

```
@doc """Takes given data and updates blog entry to database"""
def update_blog(conn, cluster, position, version, author, title, text) do

    # Log Operation
    IO.puts("Updating Blog \##{cluster}:#{position} for User: #{author}")

    # Build Document
    record = %MarcoPolo.Document{class: "BlogEntry",
        fields: %{
            "title" => title,
            "author" => author,
            "text": => text}}

    # Set Record ID
    rid = MarcoPolo.RID(
        cluster_id: cluster,
        position: position)


    # Update Record
    MarcoPolo.update_record(conn, rid, version, record, true)

end
```

MarcoPolo - Types

When you store data on the database, OrientDB sets it using the typing conventions of Java. In turn, when MarcoPolo retrieves data from OrientDB, the return values use the typing conventions of Elixir. This ensures that the Erlang VM can understand these return values.

Typing in Java is somewhat more granular than it is in Elixir. For instance, where Elixir has the integer, Java has the 2-byte short, the 4-byte integer, and the 8-byte long. Records created in MarcoPolo encode Elixir integers as the 4-byte Java integers. In cases where you want to take advantage of Java typing in data storage or simply use the more granular types available in that language, you can set the type you want MarcoPolo to use when sending data to OrientDB.

Typing in MarcoPolo

In any situation where you send data from your application to OrientDB, MarcoPolo converts the internal type in Elixir to a default type in Java. So, for instance, if you set a boolean value on a record it gets set in OrientDB as the `java.lang.Boolean` type.

Java types that you retrieve from OrientDB are converted to their approximate Elixir type. So, if you query OrientDB and retrieve an instance of `java.lang.Long` it is set in your application as an Elixir integer.

You can also force the type to store on OrientDB by using a tuple, where the first value is the type you want and the second the value you want to set. For instance,

```
data = {:long, 944}
```

Here, an integer in your application is sent to OrientDB as a long integer. When your application retrieves a long integer, it is set on your application as an integer.

Numeric Types

- **Integers**: `83` or `{:int, 83`
    - Typed in OrientDB as `java.lang.Integer` or `int`
    - Returned to MarcoPolo as `83`
- **Short Integers**: `{:short, 83}`
    - Typed in OrientDB as `java.lang.Short` or `short`
    - Returned to MarcoPolo as `83`
- **Long Integer**: `{:long, 83}`
    - Typed in OrientDB as `java.lang.Long` or `long`
    - Returned to MarcoPolo as `83`
- **Doubles**: `3.14`
    - Typed in OrientDB as `java.lang.Double` or `double`
    - Returned to MarcoPolo as `3.14`
- **Floats**: `{:float, 3.14}`
    - Typed in OrientDB as `java.lang.Float` or `float`
    - Returned to MarcoPolo as `3.14`
- **Decminals**: Using Decimal, `Decimal.new(3.14)`
    - Typed in OrientDB as `java.math.BigDecimal`
    - Returned to MarcoPolo as `Decimal.new(3.14)`

Date and Time Types

- **Date**: Using Date, `%MarcoPolo.Date{year: 2017, month: 5, day: 5}`
    - Typed in OrientDB as `java.util.Date`
    - Returned to MarcoPolo as `%MarcoPolo.Date{year: 2017 month: 5 day: 5}`
- **DateTime**: Using DateTime, `%MarcoPolo.DateTime{year: 2017,i month: 5, day: 5, hour: 15, minute: 30, sec: 0, msec: 0}`
    - Typed in OrientDB as `java.util.Date`

- Returned to MarcoPolo as `%MarcoPolo.DateTime{year: 2017,i month: 5, day: 5, hour: 15, minute: 30, sec: 0, msec: 0}`

Embedded Types

- **Document**: Using Document, `%MarcoPolo.Document{}`
  - Typed in OrientDB as `ORecord`
  - Returned to MarcoPolo as the same value.
- **List**: `[1, "foo", {:float, 3.14}]`
  - Typed in OrientDB as `List<Object>`
  - Returned ot MarcoPolo as `[1, "foo", 3.14`
- **Hash Set**: `#HashSet<[2, 1]>`
  - Typed in OrientDB as `Set<Object>`
  - Returned to MarcoPolo as `#HashSet<[2, 1]>`
- **Map**: `%{"foo" => true}`
  - Typed in OrientDB as `Map<String, ORecord>`
  - Returned to MarcoPolo as `%{"foo" => true}`

Link Types

- **Link**: Using RID, `%MarcoPolo.RID{cluster_id: 21, position: 3}`
  - Typed in OrientDB as `ORID`
  - Returned to MarcoPolo as `%MarcoPolo.RID{cluster_id: 21, position: 3}`
- **Link List**: `{:link_list, [%MarcoPolo.RID{}, ...]}`
  - Typed in OrientDB as `List<ORID>`
  - Returned to MarcoPolo as `{:link_list, [%MarcoPolo.RID{}, ...]}`
- **Link Set**: `{:link_set, #HashSet<%MarcoPolo.RID{}, ...>}`
  - Typed in OrientDB as `SET<ORID>`
  - Returned to MarcoPolo as `{:link_set, #HashSet<%MarcoPolo.RID{}, ...>}`
- **Link Map**: `{:link_map, %{"foo" => %MarcoPolo.RID{}, ...}}`
  - Typed in OrientDB as `Map<String, ORID>`
  - Returned to MarcoPolo as `{:link_map, %{"foo" => %MarcoPolo.RID{}, ...}}`

Other Types

- **Binary**: `{:binary, <<7, 2>>`
  - Typed in OrientDB as `byte[]`
  - Returned to MarcoPolo as `<<7, 2>>`
- **Boolean**: `true` or `false`
  - Typed in OrientDB as `java.lang.Boolean`
  - Returned to MarcoPolo as same value.
- **String**: `"foo"` or `<<1, 2, 3>>`
  - Typed in OrientDB as `java.lang.String`
  - Returned to MarcoPolo as the same value.

MarcoPolo - Structs

In addition to the Elixir type conversion as described in Types, MarcoPolo also provides a series of dedicated structs to use when working with OrientDB.

Struct	Description
`MarcoPolo.BinaryRecord`	Defines binary data
`MarcoPolo.Date`	Defines a date
`MarcoPolo.DateTime`	Defines date and time
`MarcoPolo.Document`	Defines an OrientDB Document
`MarcoPolo.FetchPlan`	Provides functions for traversing records
`MarcoPolo.RID`	Defines a Record ID

MarcoPolo - `BinaryRecord`

This struct represents binary data in your Elixir application. It rends as an `ORecordBytes` class in OrientDB.

Working with Binary Records

```
%MarcoPolo.BinaryRecord{
    :rid <record-id>,
    :contents <record-data>,
    :version <record-version>}
```

- **`<record-id>`** Defines the Record ID, an instance of MarcoPolo.RID.
- **`<record-data>`** Defines record data.
- **`<record-version>`** Defines the record version, a non-negative integer.

Example

In cases where you create binary records frequently with the same data, you might create a function to generate the struct from limited data. For instance, say you have a web application where new blog entries are all created on the same cluster:

```
@doc """ Create binary record of blog entry """
def gen_blog(blog_data) do

    # Create and Return Binary Record
    %MarcoPolo.BinaryRecord{
        :rid MarcoPolo.RID(:cluster 14),
        :contents blog_data,
        :version 1}

end
```

MarcoPolo - `Date`

This struct is defines date objects in your Elixir application. It renders as a `java.util.Date` class in OrientDB.

Working with Dates

```
%MarcoPolo.Date{
    :year <year>,
    :month <month>,
    :day <day>}
```

- `<year>` Defines the year in the date, a non-negative integer. Defaults to 0.
- `<month>` Defines the month in the date, a non-negative integer between 1 and 12. Defaults to 1.
- `<day>` Defines the day in the date, a non-negative integer between 1 and 31. Defaults to 1.

Example

For instance, imagine you have an application that logs quarterly reports in OrientDB. You might create a function to automatically generate fixed date objects for each quarter of the fiscal year, which in the United States runs from October 1 to September 30.

```
@doc """ Generate Date for Fiscal Quarter """
def gen_date_quarter(quart, year) do

    # Determine Month of Fiscal Quarter
    case quart do
        1 -> month = 10
        2 -> month = 1
        3 -> month = 4
        4 -> month = 7
    end

    # Generate and Return Date
    %MarcoPolo.Date{
        :year year,
        :month month,
        :day 1}
end
```

MarcoPolo - `DateTime`

This struct defines datetime objects in your Elixir application. It renders as a `java.util.Date` class in OrientDB.

Working with DateTimes

```
%MarcoPolo.DateTime{
   :year <year>,
   :month <month>,
   :day <day>,
   :hour <hour>,
   :min <minute>,
   :sec <sec>,
   :msec <msec>}
```

- `<year>` Defines the year in the date, a non-negative integer. Defaults to 0.w
- `<month>` Defines the month in the date, a non-negative integer between 1 and 12. Defaults to 1.
- `<day>` Defines the day in the date, a non-negative integer between 1 and 31. Defaults to 1.
- `<hour>` Defines the hour in the time, a non-negative integer between 0 and 23. Defaults to 0.
- `<minute>` Defines the minute of the time, a non-negative integer between 0 and 59. Defaults to 0.
- `<sec>` Defines the second of the time, a non-negative integer between 0 and 59. Defaults to 0.
- `<msec>` Defines the microseconds of the time, a non-negative integer between 0 and 999. Defaults to 0.

Example

Imagine a case where you run a particular operation at set times each day. Rather than defining DateTime objects every time you need to timestamp this operation, you might want a function to set the relevant data for you:

```
@doc """ Generate Date for Fiscal Quarter """
def gen_datetime(year, month, day, count) do

    # Determine Month of Fiscal Quarter
    case count do
        1 -> hour = 6
        2 -> hour = 12
        3 -> hour = 18
        4 -> hour = 0
    end

    # Generate and Return Date
    %MarcoPolo.Date{
        :year year,
        :month month,
        :day day,
        :hour hour}
end
```

MarcoPolo - `Document`

This struct defines a document for your Elixir application. It renders as an `ODocument` class in OrientDB.

Working with Documents

```
%MarcoPolo.Document{
    :rid <rid>,
    :class <class>,
    :version <version>,
    :fields <data>}
```

- **`<rid>`** Defines the Record ID. It is an instance of MarcoPolo.RID.
- **`<class>`** Defines the record class.
- **`<version>`** Defines the record version.
- **`<data>`** Defines the record data.

Example

In cases where your application generates a series of very similar documents, you might create a function that populates default values:

```
@doc """ Generate Blog Document """
def gen_blogdoc(title, text) do

    %MarcoPolo.Document{
        :rid MarcoPolo.RID(:cluster 4),
        :class "BlogEntry",
        :version 1,
        :fields %{
            "title" => title
            "text" => text}}
end
```

MarcoPolo - `FetchPlan`

This struct defines a fetching strategy for you to use queries. It provides functions that you may find useful when traversing records.

Working with Fetch Plans

Using the `FetchPlan` struct, you gain access to two additional functions: `resolve_links()` and `resolve_links!()`. Each performs the same operation, however `resolve_links!()` raises an exception when it attempts to retrieve records that don't exist on the database.

Syntax

```
# Resolve Links, ignore Exception if Record Not Found
resolve_links(<rids>, <linked>)

# Resolve Links, raise Exception if Record Not Found
resolve_links!(<rids>, <linked>)
```

- **`<rid>`** Defines a set of Record ID's, as `MarcoPolo.RID` instances.
- **`<linked>`** Defines the document the given records link to, as a `MarcoPolo.Document` instance.

MarcoPolo - `RID`

This struct defines a Record ID for your Elixir application. It renders as an `ORID` instance in OrientDB.

Working with Record ID's

```
%MarcoPolo.RIDr{
    :cluster_id <cluster>,
    :position <position>}
```

- **`<cluster>`** Defines the cluster the record occurs in.
- **`<position>`** Defines the position of the record in the cluster.

Scala API

There's no specific API written in Scala for OrientDB, but since there's a Java API it'easy to use that one to access OrientDB from Scala, and if needed to write wrappers around it for making a more Scala-like API.

Here we just explain how to start using Scala for doing some basic operations on OrientDB, based on this GitHub repository: OrientDbScalaExample that uses the Graph API. To fully leverage the features of the API, refer to the Java documentation.

Java method invocation problems

Usually the main problems are related to the difference in calling conventions between Scala and Java. Attention must be paid when passing parameters to methods with varargs like `OrientGraph.addVertex(...)` , because if not converted to java's repeated args correctly it will cause a compilation problem.

More detailed info here: http://stackoverflow.com/questions/3022865/calling-java-vararg-method-from-scala-with-primitives

http://stackoverflow.com/questions/1008783/using-varargs-from-scala

http://stackoverflow.com/questions/3856536/how-to-pass-a-string-scala-vararg-to-a-java-method-using-scala-2-8

build.sbt

Let's start defining the build.sbt. All we need is a library dependency:

```
libraryDependencies ++= Seq(
  "com.orientechnologies" % "orientdb-graphdb" % "2.2.0",
)
```

Creating/Opening a Database

We can start creating/opening a database:

```
    val uri: String = "plocal:target/database/scala_sample"
    val graph: OrientGraph = new OrientGraph(uri)
```

If the database at the specified uri is existing, it will be just opened; if it's not existing, it will be created and opened.

Creating new classes

If we need to define new classes, we can use the *createVertexType()* method of the OrientGraph class:

```
    val person: OrientVertexType = graph.createVertexType("Person")
    person.createProperty("firstName", OType.STRING)
    person.createProperty("lastName", OType.STRING)
```

we can define as many properties we need, each one belonging to a Type (the second parameter of the *createProperty()* method).

In the same way, we can extend edges, like in this example:

```
    val work: OrientEdgeType = graph.createEdgeType("Work")
    work.createProperty("startDate", OType.DATE)
    work.createProperty("endDate", OType.DATE)
```

Adding vertices

If we need to add Vertices, we can use the *OrientGraph.addVertex()* method, like in this example:

```
    val johnDoe: Vertex = graph.addVertex("class:Person", Nil: _*)
    johnDoe.setProperty("firstName", "John")
    johnDoe.setProperty("lastName", "Doe")
```

As seen before in the paragraph "Java method invocation problems", the second parameter of the *addVertex()* method (the *Nil:*_*) is needed to tell the Scala compiler which of the overloaded version of the method to use. If not supplied, the call will not compile.

Luckily, the properties of the vertex can be specified in the constructor itself:

```
    val johnSmith: Vertex = graph.addVertex("class:Person", "firstName", "John", "lastName", "Smith")
```

Adding edges

If we need to add edges, we can use the *OrientGraph.addEdge()* method:

```
    val johnDoeAcme: Edge = graph.addEdge(null, johnDoe, acme, WorkEdgeLabel)
    johnDoeAcme.setProperty("startDate", "2010-01-01")
    johnDoeAcme.setProperty("endDate", "2013-04-21")
```

The first parameter of the call *addEdge()* is the RID, which is automatically generated by OrientDB and hence not needed. Unfortunately the edge has no overloaded constructor to pass all the properties in one call. Another way to add an edge is starting form a vertex, like in this case:

```
    val johnSmithAcme: Edge = johnSmith.addEdge(WorkEdgeLabel, acme)
    johnSmithAcme.setProperty("startDate", "2009-01-01")
```

In this case we're connecting the *johnSmith* vertex to the *acme* company directly using the *addEdge()* method defined in the Vertex interface.

Querying the database

We can access our data using the *OrientGraph.command()* method:

```
    val results: OrientDynaElementIterable = graph
          .command(new OCommandSQL(s"SELECT expand(in('Work')) FROM Company WHERE name='ACME'"))
          .execute()
```

that will return an *Iterable* containing the results. In this case, the query finds out all the vertices of the *Person* class that have a *Work* relationship with the *acme* company (or - in other words - all the vertices that have edges labeled *Work* going out to vertices of class *Company* that have the property *name* set to *acme*); the *iterable* contains as many elements as the vertices matching the query. In this code sample, we can see how to access the edges of a vertex while iterating on the results:

```
    results.foreach(v => {

        // gets the person
        val person = v.asInstanceOf[OrientVertex]

        // checks if the out edge Work contains the "endDate" property
        val workEdgeIterator = person.getEdges(Direction.OUT, WorkEdgeLabel).iterator()
        val status = if (!workEdgeIterator.isEmpty && workEdgeIterator.next().getProperty("endDate") != null) "retired" else "
 active"

        println(s"Name: ${person.getProperty("lastName")}, ${person.getProperty("firstName")} [${status}]")
    })
```

since the GraphAPI is Tinkerpop compliant, we have to cast every element of the *Iterable* to the implementation of Tinkerpop supplied by OrientDB (*OrientVertex*), and from this object access the edges.

Using SQL

Using SQL commands is straightforward:

```
graph.command(new OCommandSQL("DELETE VERTEX V")).execute()
```

where graph is an instance of OrientGraph class.

OrientDbSample.scala

This is the complete file:

```
import com.orientechnologies.orient.core.metadata.schema.OType
import com.orientechnologies.orient.core.sql.OCommandSQL
import com.tinkerpop.blueprints.{Direction, Edge, Vertex}
import com.tinkerpop.blueprints.impls.orient._

import scala.collection.JavaConversions._

object OrientDbSample extends App {

    val WorkEdgeLabel = "Work"

    // opens the DB (if not existing, it will create it)
    val uri: String = "plocal:target/database/scala_sample"
    val graph: OrientGraph = new OrientGraph(uri)

    try {

        // if the database does not contain the classes we need (i.e. it was just created),
        // then adds them
        if (graph.getVertexType("Person") == null) {

            // we now extend the Vertex class for Person and Company
            val person: OrientVertexType = graph.createVertexType("Person")
            person.createProperty("firstName", OType.STRING)
            person.createProperty("lastName", OType.STRING)

            val company: OrientVertexType = graph.createVertexType("Company")
            company.createProperty("name", OType.STRING)
            company.createProperty("revenue", OType.LONG)

            // we now extend the Edge class for a "Work" relationship
            // between Person and Company
            val work: OrientEdgeType = graph.createEdgeType(WorkEdgeLabel)
            work.createProperty("startDate", OType.DATE)
            work.createProperty("endDate", OType.DATE)
        }
        else {

            // cleans up the DB since it was already created in a preceding run
            graph.command(new OCommandSQL("DELETE VERTEX V")).execute()
            graph.command(new OCommandSQL("DELETE EDGE E")).execute()
        }

        // adds some people
        // (we have to force a vararg call in addVertex() method to avoid ambiguous
        // reference compile error, which is pretty ugly)
        val johnDoe: Vertex = graph.addVertex("class:Person", Nil: _*)
        johnDoe.setProperty("firstName", "John")
        johnDoe.setProperty("lastName", "Doe")

        // we can also set properties directly in the constructor call
        val johnSmith: Vertex = graph.addVertex("class:Person", "firstName", "John", "lastName", "Smith")
        val janeDoe: Vertex = graph.addVertex("class:Person", "firstName", "Jane", "lastName", "Doe")

        // creates a Company
        val acme: Vertex = graph.addVertex("class:Company", "name", "ACME", "revenue", "10000000")

        // creates edge JohnDoe worked for ACME
        val johnDoeAcme: Edge = graph.addEdge(null, johnDoe, acme, WorkEdgeLabel)
        johnDoeAcme.setProperty("startDate", "2010-01-01")
        johnDoeAcme.setProperty("endDate", "2013-04-21")

        // another way to create an edge, starting from the source vertex
        val johnSmithAcme: Edge = johnSmith.addEdge(WorkEdgeLabel, acme)
        johnSmithAcme.setProperty("startDate", "2009-01-01")

        // prints all the people who works/worked for ACME
        val res: OrientDynaElementIterable = graph
          .command(new OCommandSQL(s"SELECT expand(in('${WorkEdgeLabel}')) FROM Company WHERE name='ACME'"))
          .execute()

        println("ACME people:")
        res.foreach(v => {

            // gets the person
            val person = v.asInstanceOf[OrientVertex]

            // checks if the out edge Work contains the "endDate" property
            val workEdgeIterator = person.getEdges(Direction.OUT, WorkEdgeLabel).iterator()
            val status = if (!workEdgeIterator.isEmpty && workEdgeIterator.next().getProperty("endDate") != null) "retired" el
se "active"

            println(s"Name: ${person.getProperty("lastName")}, ${person.getProperty("firstName")} [${status}]")
        })
    }
    finally {
        graph.shutdown()
    }
}
```

HTTP Protocol

OrientDB RESTful HTTP protocol allows to talk with a OrientDB Server instance using the HTTP protocol and JSON. OrientDB supports also a highly optimized Binary protocol for superior performances.

Available Commands

allocation **DB's defragmentation**	**batch** **Batch of commands**	**class** **Operations on schema classes**	**cluster** **Operations on clusters**
command Executes commands	**connect** Create the session	**database** Information about database	**disconnect** Disconnect session
document Operations on documents by RID GET - HEAD - POST - PUT - DELETE - PATCH	**documentbyclass** Operations on documents by Class	**export** Exports a database	**function** Executes a server-side function
index Operations on indexes	**listDatabases** Available databases	**property** Operations on schema properties	**query** Query
server Information about the server			

HTTP Methods

This protocol uses the four methods of the HTTP protocol:

- **GET**, to retrieve values from the database. It's idempotent that means no changes to the database happen. Remember that in IE6 the URL can be maximum of 2,083 characters. Other browsers supports longer URLs, but if you want to stay compatible with all limit to 2,083 characters
- **POST**, to insert values into the database
- **PUT**, to change values into the database
- **DELETE**, to delete values from the database

When using POST and PUT the following are important when preparing the contents of the post message:

- Always have the content type set to “application/json” or "application/xml"
- Where data or data structure is involved the content is in JSON format
- For OrientDB SQL or Gremlin the content itself is just text

Headers

All the requests must have these 2 headers:

```
'Accept-Encoding': 'gzip,deflate'
'Content-Length': <content-length>
`
```

Where the `<content-length>` is the length of the request's body.

Syntax

The REST API is very flexible, with the following features:

- Data returned is in JSON format
- JSONP callback is supported
- Support for http and https connections
- The API itself is case insensitive
- API can just be used as a wrapper to retrieve (and control) data through requests written in OrientDB SQL or Gremlin
- You can avoid using `#` for RecordIDs in URLs, if you prefer. Just drop the `#` from the URL and it will still work

---

The REST syntax used is the same for all the four HTTP methods:

Syntax: `http://<server>:<port>/<command>/[<database>/<arguments>]`

Results are always in JSON format. Support for 'document' object types is through the use of the attribute `@type : 'd'`. This also applies when using inner document objects. Example:

```
{
  "@type"  : "d",
  "Name"   : "Test",
  "Data"   : { "@type": "d",
               "value": 0 },
  "@class" : "SimpleEntity"
}
```

JSONP is also supported by adding a *callback* parameter to the request (containing the callback function name).

Syntax: `http://<server>:<port>/<command>/[<database>/<arguments>]?callback=<callbackFunctionName>`

Commands are divided in two main categories:

- Server commands, such as to know server statistics and to create a new database
- Database commands, all the commands against a database

Authentication and security

All the commands (but the Disconnect need a valid authentication before to get executed. The OrientDB Server checks if the Authorization HTTP header is present, otherwise answers with a request of authentication (HTTP error code: 401).

The HTTP client (or the Internet Browser) must send user and password using the HTTP Base authentication. Password is encoded using Base64 algorithm. Please note that if you want to encrypt the password using a safe mode take in consideration to use SSL connections.

Server commands use the realm "OrientDB Server", while the database commands use a realm per database in this form: `"OrientDB db-<database>"`, where `<database>` is the database name. In this way the Browser/HTTP client can reuse user and password inserted multiple times until the session expires or the "Disconnect" is called.

On first call (or when the session is expired and a new authentication is required), OrientDB returns the OSESSIONID parameter in response's HTTP header. On further calls the client should pass this OSESSIONID header in the requests and OrientDB will skip the authentication because a session is alive. By default sessions expire after 300 seconds (5 minutes), but you can change this configuration by setting the global setting: `network.http.sessionExpireTimeout`

JSON data type handling and Schema-less mode

Since OrientDB supports also schema-less/hybrid modes how to manage types? JSON doesn't support all the types OrientDB has, so how can I pass the right type when it's not defined in the schema?

The answer is using the special field **"@fieldTypes"** as string containing all the field types separated by comma. Example:

```
{ "@class":"Account", "date": 1350426789, "amount": 100.34,
  "@fieldTypes": "date=t,amount=c" }
```

The supported special types are:

- 'f' for float
- 'c' for decimal
- 'l' for long
- 'd' for double
- 'b' for byte and binary
- 'a' for date
- 't' for datetime
- 's' for short
- 'e' for Set, because arrays and List are serialized as arrays like [3,4,5]
- 'x' for links
- 'n' for linksets
- 'z' for linklist
- 'm' for linkmap
- 'g' for linkbag
- 'u' for custom

Keep-Alive

Attention: OrientDB HTTP API utilizes Keep-Alive feature for better performance: the TCP/IP socket is kept open avoiding the creation of a new one for each command. If you need to re-authenticate, open a new connection avoiding to reuse the already open one. To force closing put "Connection: close" in the request header.

HTTP commands

Connect

GET - Connect

Connect to a remote server using basic authentication.

Syntax: `http://<server>:[<port>]/connect/<database>`

Example

HTTP GET request: `http://localhost:2480/connect/demo` HTTP response: 204 if ok, otherwise 401.

Database

GET - Database

HTTP GET request: `http://localhost:2480/database/demo` HTTP response:

```
{
  "server": {
    "version": "1.1.0-SNAPSHOT",
    "osName": "Windows 7",
    "osVersion": "6.1",
    "osArch": "amd64",
    "javaVendor": "Oracle Corporation",
    "javaVersion": "23.0-b21"
  }, "classes": [],
  ...
}
```

POST - Database

Create a new database.

Syntax: `http://<server>:[<port>]/database/<database>/<type>`

HTTP POST request: `http://localhost:2480/database/demo/plocal`

HTTP response: `{ "classes" : [], "clusters": [], "users": [], "roles": [], "config":[], "properties":{} }`

Class

GET - Class

Gets informations about requested class.

Syntax: `http://<server>:[<port>]/class/<database>/<class-name>`

HTTP response:

```
{ "class": {
    "name": "<class-name>"
    "properties": [
      { "name": <property-name>,
        "type": <property-type>,
        "mandatory": <mandatory>,
        "notNull": <not-null>,
        "min": <min>,
        "max": <max>
      }
    ]
  }
}
```

For more information about properties look at the supported types, or see the SQL Create property page for text values to be used when getting or posting class commands

Example

HTTP GET request: `http://localhost:2480/class/demo/OFunction` HTTP response:

```
{
  "name": "OFunction",
  "superClass": "",
  "alias": null,
  "abstract": false,
  "strictmode": false,
  "clusters": [
    7
  ],
  "defaultCluster": 7,
  "records": 0,
  "properties": [
    {
      "name": "language",
      "type": "STRING",
      "mandatory": false,
      "readonly": false,
      "notNull": false,
      "min": null,
      "max": null,
      "collate": "default"
    },
    {
      "name": "name",
      "type": "STRING",
      "mandatory": false,
      "readonly": false,
      "notNull": false,
      "min": null,
      "max": null,
      "collate": "default"
    },
    {
      "name": "idempotent",
      "type": "BOOLEAN",
      "mandatory": false,
      "readonly": false,
      "notNull": false,
      "min": null,
      "max": null,
      "collate": "default"
    },
    {
      "name": "code",
      "type": "STRING",
      "mandatory": false,
      "readonly": false,
      "notNull": false,
      "min": null,
      "max": null,
      "collate": "default"
    },
    {
      "name": "parameters",
      "linkedType": "STRING",
      "type": "EMBEDDEDLIST",
      "mandatory": false,
      "readonly": false,
      "notNull": false,
      "min": null,
      "max": null,
      "collate": "default"
    }
  ]
}
```

POST - Class

Create a new class where the schema of the vertexes or edges is known. OrientDB allows (encourages) classes to be derived from other class definitions – this is achieved by using the COMMAND call with an OrientDB SQL command. Returns the id of the new class created.

Syntax: `http://<server>:[<port>]/class/<database>/<class-name>`

HTTP POST request: `http://localhost:2480/class/demo/Address2` HTTP response: `9`

Property

POST - Property

Create one or more properties into a given class. Returns the number of properties of the class.

Single property creation

Syntax: `http://<server>:[<port>]/property/<database>/<class-name>/<property-name>/[<property-type>]`

Creates a property named `<property-name>` in `<class-name>`. If `<property-type>` is not specified the property will be created as STRING.

Multiple property creation

Syntax: `http://<server>:[<port>]/property/<database>/<class-name>/`

Requires a JSON document post request content:

```
{
  "fieldName": {
      "propertyType": "<property-type>"
  },
  "fieldName": {
      "propertyType": "LINK",
      "linkedClass": "<linked-class>"
  },
  "fieldName": {
      "propertyType": "<LINKMAP|LINKLIST|LINKSET>",
      "linkedClass": "<linked-class>"
  },
  "fieldName": {
      "propertyType": "<LINKMAP|LINKLIST|LINKSET>",
      "linkedType": "<linked-type>"
  }
}
```

Example

Single property:

String Property Example: HTTP POST request: `http://localhost:2480/class/demo/simpleField` HTTP response: `1`

Type Property Example: HTTP POST request: `http://localhost:2480/class/demo/dateField/DATE` HTTP response: `1`

Link Property Example: HTTP POST request: `http://localhost:2480/class/demo/linkField/LINK/Person` HTTP response: `1`

Multiple properties: HTTP POST request: `http://localhost:2480/class/demo/` HTTP POST content:

```
{
  "name": {
      "propertyType": "STRING"
  },
  "father": {
      "propertyType": "LINK",
      "linkedClass": "Person"
  },
  "addresses": {
      "propertyType": "LINKMAP",
      "linkedClass": "Address"
  },
  "examsRatings": {
      "propertyType": "LINKMAP",
      "linkedType": "INTEGER"
  }
  "events": {
      "propertyType": "LINKLIST",
      "linkedType": "DATE"
  }
  "family": {
      "propertyType": "LINKLIST",
      "linkedClass": "Person"
  }
...
```

HTTP response: `6`

Cluster

GET - Cluster

Where the primary usage is a document db, or where the developer wants to optimise retrieval using the clustering of the database, use the CLUSTER command to browse the records of the requested cluster.

Syntax: `http://<server>:[<port>]/cluster/<database>/<cluster-name>/`

Where `<limit>` is optional and tells the maximum of records to load. Default is 20.

Example

HTTP GET request: `http://localhost:2480/cluster/demo/Address`

HTTP response:

```
{ "schema": {
    "id": 5,
    "name": "Address"
  },
  "result": [{
      "_id": "11:0",
      "_ver": 0,
      "@class": "Address",
      "type": "Residence",
      "street": "Piazza Navona, 1",
      "city": "12:0"
    }
...
```

Command

POST - Command

Execute a command against the database. Returns the records affected or the list of records for queries. Command executed via POST can be non-idempotent (look at Query).

Syntax: `http://<server>:[<port>]/command/<database>/<language>[/<command-text>[/limit[/<fetchPlan>]]]`

The content can be `<command-text>` or starting from v2.2 a json containing the command and parameters:

- by parameter name: `{"command":<command-text>, "parameters":{"<param-name>":<param-value>} }`
- by parameter position: `{"command":<command-text>, "parameters":[<param-value>] }`

Where:

- `<language>` is the name of the language between those supported. OrientDB distribution comes with "sql" and GraphDB distribution has both "sql" and "gremlin"
- `command-text` is the text containing the command to execute
- `limit` is the maximum number of record to return. Optional, default is 20
- `fetchPlan` is the fetching strategy to use. For more information look at Fetching Strategies. Optional, default is *:1 (1 depth level only)
- `returnExecutionPlan` (since v 3.0.15) if set to "false", the execution plan for the statement is not returned in the JSON result. You can also use `return-execution-plan` header property instead.

The *command-text* can appear in either the URL or the content of the POST transmission. Where the command-text is included in the URL, it must be encoded as per normal URL encoding. By default the result is returned in JSON. To have the result in CSV, pass "Accept: text/csv" in HTTP Request.

Starting from v2.2, the HTTP payload can be a JSON with both command to execute and parameters. Example:

Execute a query passing parameters by name:

```
{
  "command": "select from V where name = :name and city = :city",
  "parameters": {
    "name": "Luca",
    "city": "Rome"
  }
}
```

Execute a query passing parameters by position:

```
{
  "command": "select from V where name = ? and city = ?",
  "parameters": [ "Luca", "Rome" ]
}
```

Read the SQL section or the Gremlin introduction for the type of commands.

Example

HTTP POST request: `http://localhost:2480/command/demo/sql` content: `update Profile set online = false`

HTTP response: `10`

Or the same:

HTTP POST request: `http://localhost:2480/command/demo/sql/update Profile set online = false`

HTTP response: `10`

Extract the user list in CSV format using curl

```
curl --user admin:admin --header "Accept: text/csv" -d "select from ouser" "http://localhost:2480/command/GratefulDeadConcerts
/sql"
```

Batch

POST - Batch

Executes a batch of operations in a single call. This is useful to reduce network latency issuing multiple commands as multiple requests. Batch command supports transactions as well.

Syntax: `http://<server>:[<port>]/batch/<database>`

Content: { "transaction" : , "operations" : [{ "type" : "" }*] }

Returns: the result of last operation.

Where: *type* can be:

- 'c' for create, 'record' field is expected.
- 'u' for update, 'record' field is expected.
- 'd' for delete. The '@rid' field only is needed.
- 'cmd' for commands (Since v1.6). The expected fields are:
  - 'language', between those supported (sql, gremlin, script, etc.)
  - 'command' as the text of the command to execute
- 'script' for scripts (Since v1.6). The expected fields are:
  - 'language', between the language installed in the JVM. Javascript is the default one, but you can also use SQL (see below)
  - 'script' as the text of the script to execute

Example

```
{ "transaction" : true,
  "operations" : [
    { "type" : "u",
      "record" : {
        "@rid" : "#14:122",
        "name" : "Luca",
        "vehicle" : "Car"
      }
    }, {
      "type" : "d",
      "record" : {
        "@rid" : "#14:100"
      }
    }, {
      "type" : "c",
      "record" : {
        "@class" : "City",
        "name" : "Venice"
      }
    }, {
      "type" : "cmd",
      "language" : "sql",
      "command" : "create edge Friend from #10:33 to #11:33"
    }, {
      "type" : "script",
      "language" : "javascript",
      "script" : "orient.getGraph().createVertex('class:Account')"
    }
  ]
}
```

SQL batch

```
{ "transaction" : true,
  "operations" : [
    {
      "type" : "script",
      "language" : "sql",
      "script" : [ "LET account = CREATE VERTEX Account SET name = 'Luke'",
                   "LET city = SELECT FROM City WHERE name = 'London'",
                   "CREATE EDGE Lives FROM $account TO $city RETRY 100" ]
    }
  ]
}
```

Function

POST and GET - Function

Executes a server-side function against the database. Returns the result of the function that can be a string or a JSON containing the document(s) returned.

The difference between GET and POST method calls are if the function has been declared as idempotent. In this case can be called also by GET, otherwise only POST is accepted.

Syntax: `http://<server>:[<port>]/function/<database>/<name>[/<argument>*]<server>`

Where

- `<name>` is the name of the function
- `<argument>`, optional, are the arguments to pass to the function. They are passed by position.

Creation of functions, when not using the Java API, can be done through the Studio in either Orient DB SQL or Java – see the OrientDB Functions page.

Example

HTTP POST request: `http://localhost:2480/function/demo/sum/3/5`

HTTP response: `8.0`

Database

GET - Database

Retrieve all the information about a database.

Syntax: `http://<server>:[<port>]/database/<database>`

Example

HTTP GET request: `http://localhost:2480/database/demo`

HTTP response:

```
{"classes": [
  {
    "id": 0,
    "name": "ORole",
    "clusters": [3],
    "defaultCluster": 3, "records": 0},
  {
    "id": 1,
    "name": "OUser",
    "clusters": [4],
    "defaultCluster": 4, "records": 0},
  {
...
```

POST - Database

Create a new database. Requires additional authentication to the server.

Syntax for the url `http://:

- *storage* can be
- 'plocal' for disk-based database
- 'memory' for in memory only database.

- *type*, is optional, and can be document or graph. By default is a document.

Example

HTTP POST request: `http://localhost:2480/database/demo2/local` HTTP response:

```
{ "classes": [
  {
    "id": 0,
    "name": "ORole",
    "clusters": [3],
    "defaultCluster": 3, "records": 0},
  {
    "id": 1,
    "name": "OUser",
    "clusters": [4],
    "defaultCluster": 4, "records": 0},
  {
...
```

DELETE - Database

Drop a database. Requires additional authentication to the server.

Syntax: `http://<server>:[<port>]/database/<databaseName>`

Where:

- **databaseName** is the name of database

Example

HTTP DELETE request: `http://localhost:2480/database/demo2` HTTP response code 204

Export

GET - Export

Exports a gzip file that contains the database JSON export.

Syntax: http://:[]/export/

HTTP GET request: `http://localhost:2480/export/demo2` HTTP response: demo2.gzip file

Import

POST - Import

Imports a database from an uploaded JSON text file.

Syntax: `http://<server>:[<port>]/import/<database>`

The body of the HTTP call has to be the JSON of an exported DB (plain text). Multipart is not supported.

Important: Connect required: the connection with the selected database must be already established

Example

HTTP POST request: `http://localhost:2480/import/` HTTP response: returns a JSON object containing the result text *Success*:

```
{
  "responseText": "Database imported correctly"
}
```

Fail::

```
{
  "responseText": "Error message"
}
```

List Databases

GET - List Databases

Retrieves the available databases.

Syntax: `http://<server>:<port>/listDatabases`

To let to the Studio to display the database list by default the permission to list the database is assigned to guest. Remove this permission if you don't want anonymous user can display it.

For more details see Server Resources

Example of configuration of "guest" server user: a15b5e6bb7d06bd5d6c35db97e51400b

Example

HTTP GET request: `http://localhost:2480/listDatabases` HTTP response:

```
{
  "@type": "d", "@version": 0,
    "databases": ["demo", "temp"]
      }
```

Disconnect

GET - Disconnect

Syntax: `http://<server>:[<port>]/disconnect`

Example

HTTP GET request: `http://localhost:2480/disconnect` HTTP response: empty.

Document

GET - Document

This is a key way to retrieve data from the database, especially when combined with a `<fetchplan>`. Where a single result is required then the RID can be used to retrieve that single document.

Syntax: `http://<server>:[<port>]/document/<database>/<record-id>[/<fetchPlan>]`

Where:

- `<record-id>` See Concepts: Record ID
- `<fetchPlan>` Optional, is the fetch plan used. 0 means the root record, -1 infinite depth, positive numbers is the depth level. Look at Fetching Strategies for more information.

Example

HTTP GET request: `http://localhost:2480/document/demo/9:0`

HTTP response can be:

- HTTP Code 200, with the document in JSON format in the payload, such as:

```
{
"_id": "9:0",
"_ver": 2,
"@class": "Profile",
"nick": "GGaribaldi",
"followings": [],
"followers": [],
"name": "Giuseppe",
"surname": "Garibaldi",
"location": "11:0",
"invitedBy": null,
"sex": "male",
"online": true
}
```

- HTTP Code 404, if the document was not found

The example above can be extended to return all the edges and vertices beneath #9:0

HTTP GET request: `http://localhost:2480/document/demo/9:0/*:-1`

HEAD - Document

Check if a document exists

Syntax: `http://<server>:[<port>]/document/<database>/<record-id>`

Where:

- `<record-id>` See Concepts: Record ID

Example

HTTP HEAD request: `http://localhost:2480/document/demo/9:0`

HTTP response can be:

- HTTP Code 204, if the document exists
- HTTP Code 404, if the document was not found

POST - Document

Create a new document. Returns the document with the new @rid assigned. Before 1.4.x the return was the @rid content only.

Syntax: `http://<server>:[<port>]/document/<database>`

Example

HTTP POST request: `http://localhost:2480/document/demo`

```
  content:
  {
    "@class": "Profile",
    "nick": "GGaribaldi",
    "followings": [],
    "followers": [],
    "name": "Giuseppe",
    "surname": "Garibaldi",
    "location": "11:0",
    "invitedBy": null,
    "sex": "male",
    "online": true
  }
```

HTTP response, as the document created with the assigned RecordID as @rid:

```
{
  "@rid": "#11:4456",
  "@class": "Profile",
  "nick": "GGaribaldi",
  "followings": [],
  "followers": [],
  "name": "Giuseppe",
  "surname": "Garibaldi",
  "location": "11:0",
  "invitedBy": null,
  "sex": "male",
  "online": true
}
```

PUT - Document

Update a document. Remember to always pass the version to update. This prevent to update documents changed by other users (MVCC).

Syntax: `http://<server>:[<port>]/document/<database>[/<record-id>][?updateMode=full|partial]` Where:

- **updateMode** can be **full** (default) or **partial**. With partial mode only the delta of changes is sent, otherwise the entire document is replaced (full mode)

Example

HTTP PUT request: `http://localhost:2480/document/demo/9:0`

```
content:
{
  "@class": "Profile",
  "@version": 3,
  "nick": "GGaribaldi",
  "followings": [],
  "followers": [],
  "name": "Giuseppe",
  "online": true
}
```

HTTP response, as the updated document with the updated @version field (Since v1.6):

```
content:
{
  "@class": "Profile",
  "@version": 4,
  "nick": "GGaribaldi",
  "followings": [],
  "followers": [],
  "name": "Giuseppe",
  "online": true
}
```

PATCH - Document

Update a document with only the difference to apply. Remember to always pass the version to update. This prevent to update documents changed by other users (MVCC).

Syntax: `http://<server>:[<port>]/document/<database>[/<record-id>]` Where:

Example

This is the document 9:0 before to apply the patch:

```
{
  "@class": "Profile",
  "@version": 4,
  "name": "Jay",
  "amount": 10000
}
```

HTTP PATCH request: `http://localhost:2480/document/demo/9:0`

```
content:
{
  "@class": "Profile",
  "@version": 4,
  "amount": 20000
}
```

HTTP response, as the updated document with the updated @version field (Since v1.6):

```
content:
{
  "@class": "Profile",
  "@version": 5,
  "name": "Jay",
  "amount": 20000
}
```

DELETE - Document

Delete a document.

Syntax: `http://<server>:[<port>]/document/<database>/<record-id>`

Example

HTTP DELETE request: `http://localhost:2480/document/demo/9:0`

HTTP response: empty

Document By Class

GET Document by Class

Retrieve a document by cluster name and record position.

Syntax: `http://<server>:[<port>]/documentbyclass/<database>/<class-name>/<record-position>[/fetchPlan]`

Where:

- `<class-name>` is the name of the document's class
- `<record-position>` is the absolute position of the record inside the class' default cluster
- `<fetchPlan>` Optional, is the fetch plan used. 0 means the root record, -1 infinite depth, positive numbers is the depth level. Look at Fetching Strategies for more information.

Example

HTTP GET request: `http://localhost:2480/documentbyclass/demo/Profile/0`

HTTP response:

```
{
  "_id": "9:0",
  "_ver": 2,
  "@class": "Profile",
  "nick": "GGaribaldi",
  "followings": [],
  "followers": [],
  "name": "Giuseppe",
  "surname": "Garibaldi",
  "location": "11:0",
  "invitedBy": null,
  "sex": "male",
  "online": true
}
```

HEAD - Document by Class

Check if a document exists

Syntax: `http://<server>:[<port>]/documentbyclass/<database>/<class-name>/<record-position>`

Where:

- `<class-name>` is the name of the document's class
- `<record-position>` is the absolute position of the record inside the class' default cluster

Example

HTTP HEAD request: `http://localhost:2480/documentbyclass/demo/Profile/0`

HTTP response can be:

- HTTP Code 204, if the document exists
- HTTP Code 404, if the document was not found

Allocation

GET - Allocation

Retrieve information about the storage space of a disk-based database.

Syntax: `http://<server>:[<port>]/allocation/<database>`

Example

HTTP GET request: `http://localhost:2480/allocation/demo`

HTTP response: `{ "size": 61910, "segments": [ {"type": "d", "offset": 0, "size": 33154}, {"type": "h", "offset": 33154, "size": 4859}, {"type": "h", "offset": 3420, "size": 9392}, {"type": "d", "offset": 12812, "size": 49098} ], "dataSize": 47659, "dataSizePercent": 76, "holesSize": 14251, "holesSizePercent": 24 }`

Index

NOTE: Every single new database has the default manual index called "dictionary".

GET - Index

Retrieve a record looking into the index.

Syntax: `http://<server>:[<port>]/index/<database>/<index-name>/<key>`

Example

HTTP GET request: `http://localhost:2480/index/demo/dictionary/test` HTTP response:

```
{
  "name" : "Jay",
  "surname" : "Miner"
}
```

PUT - Index

Create or modify an index entry.

Syntax: `http://<server>:[<port>]/index/<database>/<index-name>/<key>`

Example

HTTP PUT request: `http://localhost:2480/index/demo/dictionary/test` content: `{ "name" : "Jay", "surname" : "Miner" }`

HTTP response: No response.

DELETE - Index

Remove an index entry.

Syntax: `http://<server>:[<port>]/index/<database>/<index-name>/<key>`

Example

HTTP DELETE request: `http://localhost:2480/index/demo/dictionary/test` HTTP response: No response.

Query

GET - Query

Execute a query against the database. Query means only idempotent commands like SQL SELECT and TRAVERSE. Idempotent means the command is read-only and can't change the database. Remember that in IE6 the URL can be maximum of 2,083 characters. Other browsers supports longer URLs, but if you want to stay compatible with all limit to 2,083 characters.

Syntax: `http://<server>:[<port>]/query/<database>/<language>/<query-text>[/<limit>][/<fetchPlan>]`

Where:

- `<language>` is the name of the language between those supported. OrientDB distribution comes with "sql" only. Gremlin language cannot be executed with **query** because it cannot guarantee to be idempotent. To execute Gremlin use command instead.
- `query-text` is the text containing the query to execute
- `limit` is the maximum number of record to return. Optional, default is 20
- `fetchPlan` is the fetching strategy to use. For more information look at Fetching Strategies. Optional, default is *:1 (1 depth level only)

Other key points:

- To use commands that change the database (non-idempotent), see the POST – Command section
- The command-text included in the URL must be encoded as per a normal URL
- See the SQL section for the type of queries that can be sent

Example

HTTP GET request: `http://localhost:2480/query/demo/sql/select from Profile`

HTTP response:

```
{ "result": [
{
  "_id": "-3:1",
  "_ver": 0,
  "@class": "Address",
  "type": "Residence",
  "street": "Piazza di Spagna",
  "city": "-4:0"
},
{
  "_id": "-3:2",
  "_ver": 0,
  "@class": "Address",
  "type": "Residence",
  "street": "test",
  "city": "-4:1"
}] }
```

The same query with the limit to maximum 20 results using the fetch plan *:-1 that means load all recursively:

HTTP GET request: `http://localhost:2480/query/demo/sql/select from Profile/20/*:-1`

Server

GET - Server

Retrieve information about the connected OrientDB Server. Requires additional authentication to the server.

Syntax: `http://<server>:[<port>]/server`

Example

HTTP GET request: `http://localhost:2480/server` HTTP response:

```
{
  "connections": [{
    "id": "4",
    "id": "4",
    "remoteAddress": "0:0:0:0:0:0:0:1:52504",
    "db": "-",
    "user": "-",
    "protocol": "HTTP-DB",
    "totalRequests": "1",
    "commandInfo": "Server status",
    "commandDetail": "-",
    "lastCommandOn": "2010-05-26 05:08:58",
    "lastCommandInfo": "-",
    "lastCommandDetail": "-",
    "lastExecutionTime": "0",
    "totalWorkingTime": "0",
...
```

POST - Server

Changes server configuration. Supported configuration are:

- any setting contained in OGlobalConfiguation class, by using the prefix `configuration` in setting-name
- logging level, by using the prefix `log` in setting-name

Syntax: `http://<server>:[<port>]/server/<setting-name>/<setting-value>`

Example

Example on changing the server log level to FINEST

```
localhost:2480/server/log.console/FINEST
```

Example on changing the default timeout for query to 10 seconds

```
localhost:2480/server/configuration.command.timeout/10000
```

DISTRIBUTED

GET - Status

Shows the status of the OrientDB Server in distributed mode only with Enterprise Edition. Server authentication required.

Syntax: `http://<ip>:<port>/distributed/stats`

HTTP Response:

```
{
    "localName": "cluster",
    "localId": "f3e70ed0-3f9a-4a64-afa4-ccd82368c774",
    "members": [
        {
            "id": 0,
            "uuid": "f3e70ed0-3f9a-4a64-afa4-ccd82368c774",
            "name": "node1",
            "publicAddress": null,
            "startedOn": "2017-09-28 11:03:49:167",
            "status": "ONLINE",
            "connections": 0,
            "listeners": [
                {
                    "protocol": "ONetworkProtocolBinary",
                    "listen": "192.168.14.1:2424"
                },
                {
                    "protocol": "ONetworkProtocolHttpDb",
                    "listen": "192.168.14.1:2480"
                }
            ],
            "user_replicator": "-9131097675098168918",
            "databases": [
                "GratefulDeadConcerts"
            ],
            "usedMemory": 104192088,
            "freeMemory": 105523112,
            "maxMemory": 3817865216,
            "latencies": {},
            "messages": {},
            "cpu": 0.25265485966884516
        }
    ],
    "clusterStats": {
        "node1": {
            "realtime": {
                "from": 1506589429001,
                "to": 9223372036854775807,
                "chronos": {},
                "statistics": {},
                "counters": {},
                "sizes": {},
                "texts": {},
                "tips": {}
            }
        }
    },
    "databasesStatus": {
        "GratefulDeadConcerts": {
            "node1": "ONLINE"
        }
    }
}
```

Connection

POST - Connection

Syntax: `http://<server>:[<port>]/connection/<command>/<id>`

Where:

- **command** can be:
  - **kill** to kill a connection
  - **interrupt** to interrupt the operation (if possible)
- **id**, as the connection id. To know all the connections use GET /connection/[<db>]

You've to execute this command authenticated in the OrientDB Server realm (no database realm), so get the root password from config/orientdb-server-config.xml file (last section).

Binary Protocol

Current protocol version for 2.1.x: **32**. Look at compatibility for retro-compatibility.

Introduction

The OrientDB binary protocol is the fastest way to interface a client application to an OrientDB Server instance. The aim of this page is to provide a starting point from which to build a language binding, maintaining high-performance.

If you'd like to develop a new binding, please take a look to the available ones before starting a new project from scratch: Existent Drivers.

Also, check the available REST implementations.

Before starting, please note that:

- **Record** is an abstraction of **Document**. However, keep in mind that in OrientDB you can handle structures at a lower level than Documents. These include positional records, raw strings, raw bytes, etc.

For more in-depth information please look at the Java classes:

- Client side: OStorageRemote.java
- Server side: ONetworkProtocolBinary.java
- Protocol constants: OChannelBinaryProtocol.java

Connection

(Since 0.9.24-SNAPSHOT Nov 25th 2010) Once connected, the server sends a short number (2 byte) containing the binary protocol number. The client should check that it supports that version of the protocol. Every time the protocol changes the version is incremented.

Getting started

After the connection has been established, a client can **Connect** to the server or request the opening of a database **Database Open**. Currently, only TCP/IP raw sockets are supported. For this operation use socket APIs appropriate to the language you're using. After the **Connect** and **Database Open** all the client's requests are sent to the server until the client closes the socket. When the socket is closed, OrientDB Server instance frees resources the used for the connection.

The first operation following the socket-level connection must be one of:

- Connect to the server to work with the OrientDB Server instance
- Open a database to open an existing database

In both cases a Session-Id is sent back to the client. The server assigns a unique Session-Id to the client. This value must be used for all further operations against the server. You may open a database after connecting to the server, using the same Session-Id

Session

The session management supports two modes: stateful and stateless:

- the stateful is based on a Session-id
- the stateless is based on a Token

The session mode is selected at open/connect operation.

Session-Id

All the operations that follow the open/connect must contain, as the first parameter, the client **Session-Id** (as Integer, 4 bytes) and it will be sent back on completion of the request just after the result field.

NOTE: In order to create a new server-side connection, the client must send a negative number into the open/connect calls.

This **Session-Id** can be used into the client to keep track of the requests if it handles multiple session bound to the same connection. In this way the client can implement a sharing policy to save resources. This requires that the client implementation handle the response returned and dispatch it to the correct caller thread.

❗	Opening multiple TCP/IP sockets against OrientDB Server allows to parallelize requests. However, pay attention to use one Session-id per connection. If multiple sockets use the same Session-Id, requests will not be executed concurrently on the server side.

Token

All the operation in a stateless session are based on the token, the token is a byte[] that contains all the information for the interaction with the server, the token is acquired at the moment of open or connect, and need to be resend for each request. the session id used in the stateful requests is still there and is used to associate the request to the response. in the response can be resend a token in case of expire renew.

Enable debug messages on protocol

To make the development of a new client easier it's strongly suggested to activate debug mode on the binary channel. To activate this, edit the file `orientdb-server-config.xml` and configure the new parameter `network.binary.debug` on the "binary" or "distributed" listener. E.g.:

```
...
<listener protocol="distributed" port-range="2424-2430"
ip-address="127.0.0.1">
  <parameters>
    <parameter name="network.binary.debug" value="true" />
  </parameters>
</listener>
...
```

In the log file (or the console if you have configured the `orientdb-server-log.properties` file) all the packets received will be printed.

Exchange

This is the typical exchange of messages between client and server sides:

```
+------+ +------+
|Client| |Server|
+------+ +------+
| TCP/IP Socket connection |
+------------------------->|
| DB_OPEN |
+------------------------->|
| RESPONSE (+ SESSION-ID) |
+<-------------------------+
... ...
| REQUEST (+ SESSION-ID) |
+------------------------->|
| RESPONSE (+ SESSION-ID) |
+<-------------------------+
... ...
| DB_CLOSE (+ SESSION-ID) |
+------------------------->|
| TCP/IP Socket close |
+------------------------->|
```

Network message format

In explaining the network messages these conventions will be used:

- fields are bracketed by parenthesis and contain the name and the type separated by ':'. E.g. `(length:int)`

Supported types

The network protocol supports different types of information:

Type	Minimum length in bytes	Maximum length in bytes	Notes	Example
boolean	1	1	Single byte: 1 = true, 0 = false	1
byte	1	1	Single byte, used to store small numbers and booleans	1
short	2	2	Signed short type	01
int	4	4	Signed integer type	0001
long	8	8	Signed long type	00000001
bytes	4	N	Used for binary data. The format is `(length:int)[(bytes)]` . Send -1 as NULL	`000511111`
string	4	N	Used for text messages.The format is: `(length:int)[(bytes)](content:<length>)` . Send -1 as NULL	`0005Hello`
record	2	N	An entire record serialized. The format depends if a RID is passed or an entire record with its content. In case of null record then -2 as short is passed. In case of RID -3 is passes as short and then the RID: `(-3:short)(cluster-id:short)(cluster-position:long)` . In case of record: `(0:short)(record-type:byte)(cluster-id:short)(cluster-position:long)(record-version:int)(record-content:bytes)`	
strings	4	N	Used for multiple text messages. The format is: `(length:int)[(Nth-string:string)]`	`00020005Hello0007World!`

Note when the type of a field in a response depends on the values of the previous fields, that field will be written without the type (e.g., `(a-field)`). The type of the field will be then specified based on the values of the previous fields in the description of the response.

Record format

The record format is choose during the CONNECT or DB_OPEN request, the formats available are:

CSV (serialization-impl value "ORecordDocument2csv") Binary (serialization-impl value "ORecordSerializerBinary")

The CSV format is the default for all the versions 0. *and 1.* or for any client with Network Protocol Version < 22

Request

Each request has own format depending of the operation requested. The operation requested is indicated in the first byte:

- *1 byte* for the operation. See Operation types for the list
- **4 bytes** for the Session-Id number as Integer
- **N bytes** optional token bytes only present if the REQUEST_CONNECT/REQUEST_DB_OPEN return a token.
- **N bytes** = message content based on the operation type

Operation types

Command	Value as byte	Description	A
Server *(CONNECT Operations)*			
REQUEST_SHUTDOWN	1	Shut down server.	n
REQUEST_CONNECT	2	Required initial operation to access to server commands.	n
REQUEST_DB_OPEN	3	**Required initial operation** to access to the database.	n
REQUEST_DB_CREATE	4	Add a new database.	n
REQUEST_DB_EXIST	6	Check if database exists.	n
REQUEST_DB_DROP	7	Delete database.	n
REQUEST_CONFIG_GET	70	Get a configuration property.	n
REQUEST_CONFIG_SET	71	Set a configuration property.	n
REQUEST_CONFIG_LIST	72	Get a list of configuration properties.	n
REQUEST_DB_LIST	74	Get a list of databases.	n
Database *(DB_OPEN Operations)*			
REQUEST_DB_CLOSE	5	Close a database.	n
REQUEST_DB_SIZE	8	Get the size of a database (in bytes).	n
REQUEST_DB_COUNTRECORDS	9	Get total number of records in a database.	n
REQUEST_DATACLUSTER_ADD (deprecated)	10	Add a data cluster.	n
REQUEST_DATACLUSTER_DROP (deprecated)	11	Delete a data cluster.	n
REQUEST_DATACLUSTER_COUNT (deprecated)	12	Get the total number of data clusters.	n
REQUEST_DATACLUSTER_DATARANGE (deprecated)	13	Get the data range of data clusters.	n
REQUEST_DATACLUSTER_COPY	14	Copy a data cluster.	n
REQUEST_DATACLUSTER_LH_CLUSTER_IS_USED	16		n
REQUEST_RECORD_METADATA	29	Get metadata from a record.	n
REQUEST_RECORD_LOAD	30	Load a record.	n
REQUEST_RECORD_LOAD_IF_VERSION_NOT_LATEST	44	Load a record.	n
REQUEST_RECORD_CREATE	31	Add a record.	y
REQUEST_RECORD_UPDATE	32		y
REQUEST_RECORD_DELETE	33	Delete a record.	y
REQUEST_RECORD_COPY	34	Copy a record.	y
REQUEST_RECORD_CLEAN_OUT	38	Clean out record.	y
REQUEST_POSITIONS_FLOOR	39	Get the last record.	y
REQUEST_COUNT *(DEPRECATED)*	40	See REQUEST_DATACLUSTER_COUNT	n

REQUEST_COMMAND	41	Execute a command.	n
REQUEST_POSITIONS_CEILING	42	Get the first record.	n
REQUEST_TX_COMMIT	60	Commit transaction.	n
REQUEST_DB_RELOAD	73	Reload database.	n
REQUEST_PUSH_RECORD	79		n
REQUEST_PUSH_DISTRIB_CONFIG	80		n
REQUEST_PUSH_LIVE_QUERY	81		n
REQUEST_DB_COPY	90		n
REQUEST_REPLICATION	91		n
REQUEST_CLUSTER	92		n
REQUEST_DB_TRANSFER	93		n
REQUEST_DB_FREEZE	94		n
REQUEST_DB_RELEASE	95		n
REQUEST_DATACLUSTER_FREEZE (deprecated)	96		n
REQUEST_DATACLUSTER_RELEASE (deprecated)	97		n
REQUEST_CREATE_SBTREE_BONSAI	110	Creates an sb-tree bonsai on the remote server	n
REQUEST_SBTREE_BONSAI_GET	111	Get value by key from sb-tree bonsai	n
REQUEST_SBTREE_BONSAI_FIRST_KEY	112	Get first key from sb-tree bonsai	n
REQUEST_SBTREE_BONSAI_GET_ENTRIES_MAJOR	113	Gets the portion of entries greater than the specified one. If returns 0 entries than the specified entrie is the largest	n
REQUEST_RIDBAG_GET_SIZE	114	Rid-bag specific operation. Send but does not save changes of rid bag. Retrieves computed size of rid bag.	n
REQUEST_INDEX_GET	120	Lookup in an index by key	n
REQUEST_INDEX_PUT	121	Create or update an entry in an index	n
REQUEST_INDEX_REMOVE	122	Remove an entry in an index by key	n
REQUEST_INCREMENTAL_RESTORE	Incremental restore	no	2 r

Response

Every request has a response unless the command supports the asynchronous mode (look at the table above).

- **1 byte**: Success status of the request if succeeded or failed (0=OK, 1=ERROR)
- **4 bytes**: Session-Id (Integer)
- **N bytes** optional token, is only present for token based session (REQUEST_CONNECT/REQUEST_DB_OPEN return a token) and is usually empty(N=0) is only filled up by the server when renew of an expiring token is required.
- **N bytes**: Message content depending on the operation requested

Push Request

A push request is a message sent by the server without any request from the client, it has a similar structure of a response and is distinguished using the respose status byte:

- **1 byte**: Success status has value 3 in case of push request
- **4 bytes**: Session-Id has everytime MIN_INTEGER value (-2^31)
- **1 byte**: Push command id
- **N bytes**: Message content depending on the push massage, this is written as a `(content:bytes)` having inside the details of the specific message.

Statuses

Every time the client sends a request, and the command is not in asynchronous mode (look at the table above), client must read the one-byte response status that indicates OK or ERROR. The rest of response bytes depends on this first byte.

```
* OK = 0;
* ERROR = 1;
* PUSH_REQUEST = 3
```

OK response bytes are depends for every request type. ERROR response bytes sequence described below.

Errors

The format is: `[(1)(exception-class:string)(exception-message:string)]*(0)(serialized-exception:bytes)`

The pairs exception-class and exception-message continue while the following byte is 1. A 0 in this position indicates that no more data follows.

E.g. (parentheses are used here just to separate fields to make this easier to read: they are not present in the server response):

```
(1)(com.orientechnologies.orient.core.exception.OStorageException)(Can't open the storage 'demo')(0)
```

Example of 2 depth-levels exception:

```
(1)(com.orientechnologies.orient.core.exception.OStorageException)(Can't open the storage 'demo')(1)(com.orientechnologies.ori
ent.core.exception.OStorageException)(File not found)(0)
```

Since 1.6.1 we also send serialized version of exception thrown on server side. This allows to preserve full stack trace of server exception on client side but this feature can be used by Java clients only.

Operations

This section explains the *request* and *response* messages of all suported operations.

REQUEST_SHUTDOWN

Shut down the server. Requires "shutdown" permission to be set in *orientdb-server-config.xml* file.

```
Request: (user-name:string)(user-password:string)
Response: empty
```

Typically the credentials are those of the OrientDB server administrator. This is not the same as the *admin* user for individual databases.

REQUEST_CONNECT

This is the first operation requested by the client when it needs to work with the server instance. This operation returns the Session-Id of the new client to reuse for all the next calls.

```
Request: (driver-name:string)(driver-version:string)(protocol-version:short)(client-id:string)(serialization-impl:string)(toke
n-session:boolean)(support-push)(collect-stats)(user-name:string)(user-password:string)
Response: (session-id:int)(token:bytes)
```

Request

- client's **driver-name** - the name of the client driver. Example: "OrientDB Java client"
- client's **driver-version** - the version of the client driver. Example: "1.0rc8-SNAPSHOT"
- client's **protocol-version** - the version of the protocol the client wants to use. Example: 30
- client's **client-id** - can be null for clients. In clustered configurations it's the distributed node ID as TCP `host:port` . Example: "10.10.10.10:2480"
- client's **serialization-impl** - the serialization format required by the client
- **token-session** - true if the client wants to use a token-based session, false otherwise
- **support-push** - supports push messages from the server (starting from v34)
- **collect-stats** - collects statistics for the connection (starting from v34)
- **user-name** - the username of the user on the server. Example: "root"
- **user-password** - the password of the user on the server. Example: "37aed6392"

Typically the credentials are those of the OrientDB server administrator. This is not the same as the *admin* user for individual databases.

Response

- **session-id** - the new session id or a match id in case of token authentication.
- **token** - the token for token-based authentication. If the clients sends **token-session** as false in the request or the server doesn't support token-based authentication, this will be an empty `byte[]` .

REQUEST_DB_OPEN

This is the first operation the client should call. It opens a database on the remote OrientDB Server. This operation returns the Session-Id of the new client to reuse for all the next calls and the list of configured clusters in the opened databse.

```
Request: (driver-name:string)(driver-version:string)(protocol-version:short)(client-id:string)(serialization-impl:string)(toke
n-session:boolean)(support-push:boolean)(collect-stats:boolean)(database-name:string)(user-name:string)(user-password:string)
Response: (session-id:int)(token:bytes)(num-of-clusters:short)[(cluster-name:string)(cluster-id:short)](cluster-config:bytes)(
orientdb-release:string)
```

Request

- client's **driver-name** - the name of the client driver. Example: "OrientDB Java client".
- client's **driver-version** - the version of the client driver. Example: "1.0rc8-SNAPSHOT"
- client's **protocol-version** - the version of the protocol the client wants to use. Example: 30.
- client's **client-id** - can be null for clients. In clustered configurations it's the distributed node ID as TCP `host:port` . Example: "10.10.10.10:2480".
- client's **serialization-impl** - the serialization format required by the client.
- **token-session** - true if the client wants to use a token-based session, false otherwise.
- **support-push** - true if the client support push request
- **collect-stats** - true if this connection is to be counted on the server stats, normal client should use true
- **database-name** - the name of the database to connect to. Example: "demo".
- **user-name** - the username of the user on the server. Example: "root".
- **user-password** - the password of the user on the server. Example: "37aed6392".

Response

- **session-id** - the new session id or a match id in case of token authentication.
- **token** - the token for token-based authentication. If the clients sends **token-session** as false in the request or the server doesn't support token-based authentication, this will be an empty `byte[]` .
- **num-of-clusters** - the size of the array of clusters in the form `(cluster-name:string)(cluster-id:short)` that follows this number.

- **cluster-name**, **cluster-id** - the name and id of a cluster.
- **cluster-config** - it's usually null unless running in a server clustered configuration.
- **orientdb-release** - contains the version of the OrientDB release deployed on the server and optionally the build number. Example: "1.4.0-SNAPSHOT (build 13)".

REQUEST_DB_REOPEN

Used on new sockets for associate the specific socket with the server side session for the specific client, can be used exclusively with the token authentication

```
Request:empty
Response:(session-id:int)
```

REQUEST_DB_CREATE

Creates a database in the remote OrientDB server instance.

```
Request: (database-name:string)(database-type:string)(storage-type:string)(backup-path)
Response: empty
```

Request

- **database-name** - the name of the database to create. Example: "MyDatabase".
- **database-type** - the type of the database to create. Can be either `document` or `graph` (since version 8). Example: "document".
- **storage-type** - specifies the storage type of the database to create. It can be one of the supported types:
    - `plocal` - persistent database
    - `memory` - volatile database
- **backup-path** - path of the backup file to restore located on the server's file system (since version 36). This is used when a database is created starting from a previous backup

Note: it doesn't make sense to use `remote` in this context.

REQUEST_DB_CLOSE

Closes the database and the network connection to the OrientDB server instance. No response is expected.

```
Request: empty
Response: no response, the socket is just closed at server side
```

REQUEST_DB_EXIST

Asks if a database exists in the OrientDB server instance.

```
Request: (database-name:string)(server-storage-type:string)
Response: (result:boolean)
```

Request

- **database-name** - the name of the target database. *Note* that this was empty before `1.0rc1`.
- **storage-type** - specifies the storage type of the database to be checked for existence. Since `1.5-snapshot`. It can be one of the supported types:
    - `plocal` - persistent database
    - `memory` - volatile database

Response

- **result** - true if the given database exists, false otherwise.

REQUEST_DB_RELOAD

Reloads information about the given database. Available since `1.0rc4`.

```
Request: empty
Response: (num-of-clusters:short)[(cluster-name:string)(cluster-id:short)]
```

Response

- **num-of-clusters** - the size of the array of clusters in the form `(cluster-name:string)(cluster-id:short)` that follows this number.
- **cluster-name**, **cluster-id** - the name and id of a cluster.

REQUEST_DB_DROP

Removes a database from the OrientDB server instance. This operation returns a successful response if the database is deleted successfully. Otherwise, if the database doesn't exist on the server, it returns an error (an `OStorageException`).

```
Request: (database-name:string)(storage-type:string)
Response: empty
```

Request

- **database-name** - the name of the database to remove.
- **storage-type** - specifies the storage type of the database to create. Since `1.5-snapshot`. It can be one of the supported types:
  - `plocal` - persistent database
  - `memory` - volatile database

REQUEST_DB_SIZE

Returns the size of the currently open database.

```
Request: empty
Response: (size:long)
```

Response

- **size** - the size of the current database.

REQUEST_DB_COUNTRECORDS

Returns the number of records in the currently open database.

```
Request: empty
Response: (count:long)
```

Response

- **count** - the number of records in the current database.

REQUEST_DATACLUSTER_ADD

Add a new data cluster. Deprecated.

```
Request: (name:string)(cluster-id:short - since 1.6 snapshot)
Response: (new-cluster:short)
```

Where: type is one of "PHYSICAL" or "MEMORY". If cluster-id is -1 (recommended value) new cluster id will be generated.

REQUEST_DATACLUSTER_DROP

Remove a cluster. Deprecated.

```
Request: (cluster-number:short)
Response: (delete-on-clientside:byte)
```

Where:

- **delete-on-clientside** can be 1 if the cluster has been successfully removed and the client has to remove too, otherwise 0

REQUEST_DATACLUSTER_COUNT

Returns the number of records in one or more clusters. Deprecated.

```
Request: (cluster-count:short)(cluster-number:short)*(count-tombstones:byte)
Response: (records-in-clusters:long)
```

Where:

- **cluster-count** the number of requested clusters
- **cluster-number** the cluster id of each single cluster
- **count-tombstones** the flag which indicates whether deleted records should be taken in account. It is applicable for autosharded storage only, otherwise it is ignored.
- **records-in-clusters** is the total number of records found in the requested clusters

Example

Request the record count for clusters 5, 6 and 7. Note the "03" at the beginning to tell you're passing 3 cluster ids (as short each). 1,000 as long (8 bytes) is the answer.

```
Request: 03050607
Response: 00001000
```

REQUEST_DATACLUSTER_DATARANGE

Returns the range of record ids for a cluster. Deprecated.

```
Request: (cluster-number:short)
Response: (begin:long)(end:long)
```

Example

Request the range for cluster 7. The range 0-1,000 is returned in the response as 2 longs (8 bytes each).

```
Request: 07
Response: 0000000000001000
```

REQUEST_RECORD_LOAD

Loads a record by its Record ID, according to a fetch plan.

```
Request: (cluster-id:short)(cluster-position:long)(fetch-plan:string)(ignore-cache:boolean)(load-tombstones:boolean)
Response: [(payload-status:byte)[(record-type:byte)(record-version:int)(record-content:bytes)]*]+
```

Request

- **cluster-id**, **cluster-position** - the Record ID of the record.
- **fetch-plan** - the fetch plan to use or an empty string.
- **ignore-cache** - if true tells the server to ignore the cache, if false tells the server to not ignore the cache. Available since protocol v.9 (introduced in release 1.0rc9).
- **load-tombstones** - a flag which indicates whether information about deleted record should be loaded. The flag is applied only to autosharded storage and ignored otherwise.

Response

- **payload-status** - can be:
  - `0` : no records remain to be fetched.
  - `1` : a record is returned as resultset.
  - `2` : a record is returned as pre-fetched to be loaded in client's cache only. It's not part of the result set but the client knows that it's available for later access. This value is not currently used.
- **record-type** - can be:
  - `d` : document
  - `b` : raw bytes
  - `f` : flat data

REQUEST_RECORD_LOAD_IF_VERSION_NOT_LATEST

Loads a record by Record ID, according to a fetch plan. The record is only loaded if the persistent version is more recent of the version specified in the request.

```
Request: (cluster-id:short)(cluster-position:long)(version:int)(fetch-plan:string)(ignore-cache:boolean)
Response: [(payload-status:byte)[(record-type:byte)(record-version:int)(record-content:bytes)]*]*
```

Request

- **cluster-id**, **cluster-position** - the Record ID of the record.
- **version** - the version of the record to fetch.
- **fetch-plan** - the fetch plan to use or an empty string.
- **ignore-cache** - if true tells the server to ignore the cache, if false tells the server to not ignore the cache. Available since protocol v.9 (introduced in release 1.0rc9).

Response

- `payload-status` - can be:
  - `0` : no records remain to be fetched.
  - `1` : a record is returned as resultset.
  - `2` : a record is returned as pre-fetched to be loaded in client's cache only. It's not part of the result set but the client knows that it's available for later access. This value is not currently used.

- `record-type` - can be:
  - `d` : document
  - `b` : raw bytes
  - `f` : flat data

REQUEST_RECORD_CREATE

Creates a new record. Returns the Record ID of the newly created record.. New records can have version > 0 (since `1.0`) in case the Record ID has been recycled.

```
Request: (cluster-id:short)(record-content:bytes)(record-type:byte)(mode:byte)
Response: (cluster-id:short)(cluster-position:long)(record-version:int)(count-of-collection-changes)[(uuid-most-sig-bits:long)
(uuid-least-sig-bits:long)(updated-file-id:long)(updated-page-index:long)(updated-page-offset:int)]*
```

Request

- **cluster-id** - the id of the cluster in which to create the new record.
- **record-content** - the record to create serialized using the appropriate serialization format chosen at connection time.
- **record-type** - the type of the record to create. It can be:
  - `d` : document
  - `b` : raw bytes
  - `f` : flat data
- **mode** - can be:
  - `0` - **synchronous**. It's the default mode which waits for the answer before the response is sent.
  - `1` - **asynchronous**. The response is identical to the synchronous response, but the driver is encouraged to manage the answer in a callback.
  - `2` - **no-response**. Don't wait for the answer (*fire and forget*). This mode is useful on massive operations since it reduces network latency.

In versions before `2.0` , the response started with an additional **datasegment-id**, the segment id to store the data (available since version 10 - `1.0-SNAPSHOT`), with -1 meaning default one.

Response

- **cluster-id**, **cluster-position** - the Record ID of the newly created record.
- **record-version** - the version of the newly created record.

The last part of response (from `count-of-collection-changes` on) refers to RidBag management. Take a look at the main page for more details.

REQUEST_RECORD_UPDATE

Updates the record identified by the given Record ID. Returns the new version of the record.

```
Request: (cluster-id:short)(cluster-position:long)(update-content:boolean)(record-content:bytes)(record-version:int)(record-ty
pe:byte)(mode:byte)
Response: (record-version:int)(count-of-collection-changes)[(uuid-most-sig-bits:long)(uuid-least-sig-bits:long)(updated-file-i
d:long)(updated-page-index:long)(updated-page-offset:int)]*
```

Request

- **cluster-id**, **cluster-position** - the Record ID of the record to update.
- **update-content** - can be:
  - true - the content of the record has been changed and should be updated in the storage.
  - false - the record was modified but its own content has not changed: related collections (e.g. RidBags) have to be updated, but the record version and its contents should not be updated.
- **record-content** - the new contents of the record serialized using the appropriate serialization format chosen at connection time.

- **record-version** - the version of the record to update.
- **record-type** - the type of the record to create. It can be:
  - `d` : document
  - `b` : raw bytes
  - `f` : flat data
- **mode** - can be:
  - `0` - **synchronous**. It's the default mode which waits for the answer before the response is sent.
  - `1` - **asynchronous**. The response is identical to the synchronous response, but the driver is encouraged to manage the answer in a callback.
  - `2` - **no-response**. Don't wait for the answer (*fire and forget*). This mode is useful on massive operations since it reduces network latency.

Response

- **record-version** - the new version of the updated record

The last part of response (from `count-of-collection-changes` on) refers to RidBag management. Take a look at the main page for more details.

REQUEST_RECORD_DELETE

Delete a record identified by the given Record ID. During the optimistic transaction the record will be deleted only if the given version and the version of the record on the server match. This operation returns true if the record is deleted successfully, false otherwise.

```
Request: (cluster-id:short)(cluster-position:long)(record-version:int)(mode:byte)
Response: (has-been-deleted:boolean)
```

Request

- **cluster-id**, **cluster-position** - the Record ID of the record to delete.
- **record-version** - the version of the record to delete.
- **mode** - can be:
  - `0` - **synchronous**. It's the default mode which waits for the answer before the response is sent.
  - `1` - **asynchronous**. The response is identical to the synchronous response, but the driver is encouraged to manage the answer in a callback.
  - `2` - **no-response**. Don't wait for the answer (*fire and forget*). This mode is useful on massive operations since it reduces network latency.

Response

- **has-been-deleted** - true if the record is deleted successfully, false if it's not or if the record with the given Record ID doesn't exist.

REQUEST_COMMAND

Executes remote commands.

```
Request: (mode:byte)(command-payload-length:int)(class-name:string)(command-payload)
Response:
- synchronous commands: [(synch-result-type:byte)[(synch-result-content:?)]]+
- asynchronous commands: [(asynch-result-type:byte)[(asynch-result-content:?)]*](pre-fetched-record-size.md)[(pre-fetched-reco
rd)]*+
```

Request

- **mode** - it can assume one of the following values:
  - `a` - asynchronous mode

  - `s` - synchronous mode
  - `l` - live mode
- **command-payload-length** - the length of the **class-name** field plus the length of the **command-payload** field.
- **class-name** - the class name of the command implementation. There are some short forms for the most common commands, which are:
  - `q` - stands for "query" as idempotent command (e.g., `SELECT`). It's like passing `com.orientechnologies.orient.core.sql.query.OSQLSynchquery`.
  - `c` - stands for "command" as non-idempotent command (e.g., `INSERT` or `UPDATE`). It's like passing `com.orientechnologies.orient.core.sql.OCommandSQL`.
  - `s` - stands for "script" (for server-side scripting using languages like JavaScript). It's like passing `com.orientechnologies.orient.core.command.script.OCommandScript`.
  - any other string - the string is the class name of the command. The command will be created via reflection using the default constructor and invoking the `fromStream()` method against it.
- **command-payload** - is the payload of the command as specified in the "Commands" section.

Response

Response is different for synchronous and asynchronous request:

- **synchronous**:
- **synch-result-type** can be:
  - 'n', means null result
  - 'r', means single record returned
  - 'l', list of records. The format is:
    - an integer to indicate the collection size. Starting form v32, size can be -1 to stream a resultset. Last item will be null
    - all the records and each entry is typed with a short that can be:
      - '0' a record in the next bytes
      - '-2' no record and is considered as a null record
      - '-3' only a recordId in the next bytes
  - 's', set of records. The format is:
    - an integer to indicate the collection size. Starting form v32, size can be -1 to stream a resultset. Last item will be null
    - all the records and each entry is typed with a short that can be:
      - '0' a record in the next bytes
      - '-2' no record and is considered as a null record
      - '-3' only a recordId in the next bytes
  - 'w', is a simple result wrapped inside a single record, deserialize the record as the `r` option and unwrap the real result reading the field `result` of the record.
  - 'i', iterable of records
    - the result records will be streamed, no size as start is given, each entry has a flag at the start(same as **asynch-result-type**)
      - 0: no record remain to be fetched
      - 1: a record in the next bytes
      - 2: a recordin the next bytes to be loaded in client's cache only. It's not part of the result set but
- **synch-result-content**, can only be a record
- **pre-fetched-record-size**, as the number of pre-fetched records not directly part of the result set but joined to it by fetching
- **pre-fetched-record** as the pre-fetched record content
- **asynchronous**:
- **asynch-result-type** can be:
  - 0: no records remain to be fetched
  - 1: a record is returned as a resultset
  - 2: a record is returned as pre-fetched to be loaded in client's cache only. It's not part of the result set but the client knows that it's available for later access
- **asynch-result-content**, can only be a record

REQUEST_TX_COMMIT

Commits a transaction. This operation flushes all the pending changes to the server side.

```
Request: (transaction-id:int)(using-tx-log:boolean)(tx-entry)*(0-byte indicating end-of-records)
Response: (created-record-count:int)[(client-specified-cluster-id:short)(client-specified-cluster-position:long)(created-clust
er-id:short)(created-cluster-position:long)]*(updated-record-count:int)[(updated-cluster-id:short)(updated-cluster-position:lo
ng)(new-record-version:int)]*(count-of-collection-changes:int)[(uuid-most-sig-bits:long)(uuid-least-sig-bits:long)(updated-fil
e-id:long)(updated-page-index:long)(updated-page-offset:int)]*
```

Request

- **transaction-id** - the id of the transaction. Read the "Transaction ID" section below for more information.
- **using-tx-log** - tells the server whether to use the transaction log to recover the transaction or not. Use `true` by default to ensure consistency. *Note*: disabling the log could speed up the execution of the transaction, but it makes impossible to rollback the transaction in case of errors. This could lead to inconsistencies in indexes as well, since in case of duplicated keys the rollback is not called to restore the index status.
- **tx-entry** - a list of elements (terminated by a 0 byte) with the form described below.

Transaction entry

Each transaction entry can specify one out of three actions to perform: create, update or delete.

The general form of a transaction entry (**tx-entry** above) is:

```
(1:byte)(operation-type:byte)(cluster-id:short)(cluster-position:long)(record-type:byte)(entry-content)
```

The first byte means that there's another entry next. The values of the rest of these attributes depend directly on the operation type.

Update

- **operation-type** - has the value 1.
- **cluster-id**, **cluster-position** - the Record ID of the record to update.
- **record-type** - the type of the record to update (`d` for document, `b` for raw bytes and `f` for flat data).
- **entry-content** - has the form `(version:int)(update-content:boolean)(record-content:bytes)` where:
  - **update-content** - can be:
    - true - the content of the record has been changed and should be updated in the storage.
    - false - the record was modified but its own content has not changed: related collections (e.g. RidBags) have to be updated, but the record version and its contents should not be updated.
  - **version** - the version of the record to update.
  - **record-content** - the new contents of the record serialized using the appropriate serialization format chosen at connection time.

Delete

- **operation-type** - has the value 2.
- **cluster-id**, **cluster-position** - the Record ID of the record to update.
- **record-type** - the type of the record to update (`d` for document, `b` for raw bytes and `f` for flat data).
- **entry-content** - has the form `(version:int)` where:
  - **version** - the version of the record to delete.

Create

- **operation-type** - has the value 3.
- **cluster-id**, **cluster-position** - when creating a new record, set the cluster id to `-1` . The cluster position must be an integer `< -1` , unique in the scope of the transaction (meaning that if two new records are being created in the same transaction, they should have two different ids both `< -1`).
- **record-type** - the type of the record to update (`d` for document, `b` for raw bytes and `f` for flat data).
- **entry-content** - has the form `(record-content:bytes)` where:
  - **record-content** - the new contents of the record serialized using the appropriate serialization format chosen at connection time.

Transaction ID

Each transaction must have an ID; the client is responsible for assigning an ID to each transaction. The ID must be unique in the scope of each session.

Response

The response contains two parts:

- a map of "temporary" client-generated record ids to "real" server-provided record ids for each **created** record (not guaranteed to have the same order as the records in the request).
- a map of **updated** record ids to update record versions.

If the version of a created record is not `0`, then the Record ID of the created record will also appear in the list of "updated" records, along with its new version. This is a known bug.

Look at Optimistic Transaction to know how temporary Record IDs are managed.

The last part of response (from `count-of-collection-changes` on) refers to RidBag management. Take a look at the main page for more details.

REQUEST_INDEX_GET

Lookups in an index by key.

```
Request: (index-name:string)(key:document)(fetch-plan:string)
Response: (result-type:byte)
```

Request

- **index-name** - the name of the index.
- **key** - a document whose `"key"` field contains the key.
- **fetch-plan** - the fetch plan to use or an empty string.

Response

- **key** - is stored in the field named "key" inside the document
- **result-type** can be:
  - 'n', means null result
  - 'r', means single record returned
  - 'l', list of records. The format is:
  - an integer to indicate the collection size
  - all the records one by one
  - 's', set of records. The format is:
  - an integer to indicate the collection size
  - all the records one by one
  - 'a', serialized result, a byte[] is sent
- **synch-result-content**, can only be a record
- **pre-fetched-record-size**, as the number of pre-fetched records not directly part of the result set but joined to it by fetching
- **pre-fetched-record** as the pre-fetched record content

REQUEST_INDEX_PUT

Create or update an entry in index by key.

```
Request: (index-name:string)(key:document)(value:rid)
Response: no response
```

Where:

- **key** is stored in the field named "key" inside the document

REQUEST_INDEX_REMOVE

Remove an entry by key from an index. It returns true if the entry was present, otherwise false.

```
Request: (index-name:string)(key:document)
Response: (found:boolean)
```

Where:

- **key** is stored in the field named "key" inside the document

REQUEST_CREATE_SBTREE_BONSAI

```
Request: (clusterId:int)
Response: (collectionPointer)
```

See: serialization of collection pointer

Creates an sb-tree bonsai on the remote server.

REQUEST_SBTREE_BONSAI_GET

```
Request: (collectionPointer)(key:binary)
Response: (valueSerializerId:byte)(value:binary)
```

See: serialization of collection pointer

Get value by key from sb-tree bonsai.

Key and value are serialized according to format of tree serializer. If the operation is used by RidBag key is always a RID and value can be null or integer.

REQUEST_SBTREE_BONSAI_FIRST_KEY

```
Request: (collectionPointer)
Response: (keySerializerId:byte)(key:binary)
```

See: serialization of collection pointer

Get first key from sb-tree bonsai. Null if tree is empty.

Key are serialized according to format of tree serializer. If the operation is used by RidBag key is null or RID.

REQUEST_SBTREE_BONSAI_GET_ENTRIES_MAJOR

```
Request: (collectionPointer)(key:binary)(inclusive:boolean)(pageSize:int)
Response: (count:int)[(key:binary)(value:binary)]*
```

See: serialization of collection pointer

Gets the portion of entries major than specified one. If returns 0 entries than the specified entry is the largest.

Keys and values are serialized according to format of tree serializer. If the operation is used by RidBag key is always a RID and value is integer.

Default pageSize is 128.

REQUEST_RIDBAG_GET_SIZE

```
Request: (collectionPointer)(collectionChanges)
Response: (size:int)
```

See: serialization of collection pointer, serialization of collection changes

Rid-bag specific operation. Send but does not save changes of rid bag. Retrieves computed size of rid bag.

Special use of LINKSET types

> NOTE. Since 1.7rc1 this feature is deprecated. Usage of RidBag is preferable.

Starting from 1.0rc8-SNAPSHOT OrientDB can transform collections of links from the classic mode:

```
[#10:3,#10:4,#10:5]
```

to:

```
(ORIDs@pageSize:16,root:#2:6)
```

For more information look at the announcement of this new feature: https://groups.google.com/d/topic/orient-database/QF52JEwCuTM/discussion

In practice to optimize cases with many relationships/edges the collection is transformed in a mvrb-tree. This is because the embedded object. In that case the important thing is the link to the root node of the balanced tree.

You can disable this behaviour by setting

mvrbtree.ridBinaryThreshold = -1

Where *mvrbtree.ridBinaryThreshold* is the threshold where OrientDB will use the tree instead of plain collection (as before). -1 means "hey, never use the new mode but leave all as before".

Tree node binary structure

To improve performance this structure is managed in binary form. Below how is made:

```
+-----------+-----------+--------+------------+----------+-----------+---------------------+
| TREE SIZE | NODE SIZE | COLOR .| PARENT RID | LEFT RID | RIGHT RID | RID LIST .......... |
+-----------+-----------+--------+------------+----------+-----------+---------------------+
| 4 bytes . | 4 bytes . | 1 byte | 10 bytes ..| 10 bytes | 10 bytes .| 10 * MAX_SIZE bytes |
+-----------+-----------+--------+------------+----------+-----------+---------------------+
= 39 bytes + 10 * PAGE-SIZE bytes
```

Where:

- *TREE SIZE* as signed integer (4 bytes) containing the size of the tree. Only the root node has this value updated, so to know the size of the collection you need to load the root node and get this field. other nodes can contain not updated values because upon rotation of pieces of the tree (made during tree rebalancing) the root can change and the old root will have the "old" size as dirty.
- *NODE SIZE* as signed integer (4 bytes) containing number of entries in this node. It's always <= to the page-size defined at the tree level and equals for all the nodes. By default page-size is 16 items
- *COLOR* as 1 byte containing 1=Black, 0=Red. To know more about the meaning of this look at Red-Black Trees
- **PARENT RID** as RID (10 bytes) of the parent node record
- **LEFT RID** as RID (10 bytes) of the left node record
- **RIGHT RID** as RID (10 bytes) of the right node record
- **RID LIST** as the list of RIDs containing the references to the records. This is pre-allocated to the configured page-size. Since each RID takes 10 bytes, a page-size of 16 means 16 x 10bytes = 160bytes

The size of the tree-node on disk (and memory) is fixed to avoid fragmentation. To compute it: 39 bytes + 10 * PAGE-SIZE bytes. For a page-size = 16 you'll have 39 + 160 = 199 bytes.

REQUEST_PUSH_DISTRIB_CONFIG

```
(configuration:document)
```

where: **configuration** is and oriendb document serialized with the network Record Format, that contain the distributed configuration.

REQUEST_PUSH_LIVE_QUERY

```
(message-type:byte)(message-body)
```

where:
message-type is the type of message and can have as a value

- *RECORD* = 'r'
- *UNSUBSCRIBE* = 'u'

message-body is one for each type of message

Record Message Body:

```
(operation:byte)(query_token:int)(record-type:byte)(record-version:int)(cluster-id:short)(cluster-position:long)(record-conten
t:bytes)
```

where:
operation the tipe of operation happened, possible values

- *LOADED* = 0
- *UPDATED* = 1
- *DELETED* = 2
- *CREATED* = 3

query_token the token that identify the relative query of the push message, it match the result token of the live query command request.
record-type type of the record ('d' or 'b')
record-version record version
cluster-id record cluster id
cluster-position record cluster position
record-content record content

Usubscribe Message Body:

```
(query_token:int)
```

query_token the token for identify the query that has been usubscribed.

History

version 36

add support for REQUEST_INCREMENTAL_RESTORE

version 35

command result review: add support for "wrapped types" on command result set, removed support for "simple types".

is now possible a new option `w` over the one already existent `r` , `s` , `l` , `i` it consist in a document serialized in the same way of `r` that wrap the result in a field called `result` .

the old options `a` for simple results is now removed.

version 34

Add flags `support-push` and `collect-stats` on REQUEST_DB_OPEN.

version 33

Removed the token data from error heandling header in case of non token session. Removed the db-type from REQUEST_DB_OPEN added REQUEST_DB_REOPEN

Version 32

Added support of streamable resultset in case of sync command, added a new result of type 'i' that stream the result in the same way of async result.

Version 31

Added new commands to manipulate idexes: REQUEST_INDEX_GET, REQUEST_INDEX_PUT and REQUEST_INDEX_REMOVE.

Version 30

Added new command REQUEST_RECORD_LOAD_IF_VERSION_NOT_LATEST

Version 29

Added support support of live query in REQUEST_COMMAND, added new push command REQUEST_PUSH_LIVE_QUERY

Version 28

Since version 28 the REQUEST_RECORD_LOAD response order is changed from: `[(payload-status:byte)[(record-content:bytes)(record-version:int)(record-type:byte)]*]+` to: `[(payload-status:byte)[(record-type:byte)(record-version:int)(record-content:bytes)]*]+`

Version 27

Since version 27 is introduced an extension to allow use a token based session, if this modality is enabled a few things change in the modality the protocol works.

- in the first negotiation the client should ask for a token based authentication using the token-auth flag
- the server will reply with a token or an empty byte array that means that it not support token based session and is using a old style session.
- if the server don't send back the token the client can fail or drop back the the old modality.
- for each request the client should send the token and the sessionId
- the sessionId is needed only for match a response to a request
- if used the token the connections can be shared between users and db of the same server, not needed to have connection associated

to db and user.

protocol methods changed:

REQUEST_DB_OPEN

- request add token session flag
- response add of the token

REQUEST_CONNECT

- request add token session flag
- response add of the token

Version 26

Added cluster-id in the REQUEST_CREATE_RECORD response.

Version 25

Reviewd serialization of index changes in the REQUEST_TX_COMMIT for detais #2676 Removed double serialization of commands parameters, now the parameters are directly serialized in a document see Network Binary Protocol Commands and #2301

Version 24

- cluster-type and cluster-dataSegmentId parameters were removed from response for REQUEST_DB_OPEN, REQUEST_DB_RELOAD requests.
- datasegment-id parameter was removed from REQUEST_RECORD_CREATE request.
- type, location and datasegment-name parameters were removed from REQUEST_DATACLUSTER_ADD request.
- REQUEST_DATASEGMENT_ADD request was removed.
- REQUEST_DATASEGMENT_DROP request was removed.

Version 23

- Add support of `updateContent` flag to UPDATE_RECORD and COMMIT

Version 22

- REQUEST_CONNECT and REQUEST_OPEN now send the document serialization format that the client require

Version 21

- REQUEST_SBTREE_BONSAI_GET_ENTRIES_MAJOR (which is used to iterate through SBTree) now gets "pageSize" as int as last argument. Version 20 had a fixed pageSize=5. The new version provides configurable pageSize by client. Default pageSize value for protocol=20 has been changed to 128.

Version 20

- Rid bag commands were introduced.
- Save/commit was adapted to support client notifications about changes of collection pointers.

Version 19

- Serialized version of server exception is sent to the client.

Version 18

- Ability to set cluster id during cluster creation was added.

Version 17

- Synchronous commands can send fetched records like asynchronous one.

Version 16

- Storage type is required for REQUEST_DB_FREEZE, REQUEST_DB_RELEASE, REQUEST_DB_DROP, REQUEST_DB_EXIST commands.
- This is required to support plocal storage.

Version 15

- SET types are stored in different way then LIST. Before rel. 15 both were stored between squared braces [] while now SET are stored between <>

Version 14

- DB_OPEN returns information about version of OrientDB deployed on server.

Version 13

- To support upcoming auto-sharding support feature following changes were done
  - RECORD_LOAD flag to support ability to load tombstones was added.
  - DATACLUSTER_COUNT flag to support ability to count tombstones in cluster was added.

Version 12

- DB_OPEN returns the dataSegmentId foreach cluster

Version 11

- RECORD_CREATE always returns the record version. This was necessary because new records could have version > 0 to avoid MVCC problems on RID recycle

Compatibility

Current release of OrientDB server supports older client versions.

- version 35: 100% compatible 2.2-SNAPSHOT
- version 34: 100% compatible 2.2-SNAPSHOT
- version 34: 100% compatible 2.2-SNAPSHOT
- version 33: 100% compatible 2.2-SNAPSHOT
- version 32: 100% compatible 2.1-SNAPSHOT
- version 31: 100% compatible 2.1-SNAPSHOT

- version 30: 100% compatible 2.1-SNAPSHOT
- version 29: 100% compatible 2.1-SNAPSHOT
- version 28: 100% compatible 2.1-SNAPSHOT
- version 27: 100% compatible 2.0-SNAPSHOT
- version 26: 100% compatible 2.0-SNAPSHOT
- version 25: 100% compatible 2.0-SNAPSHOT
- version 24: 100% compatible 2.0-SNAPSHOT
- version 23: 100% compatible 2.0-SNAPSHOT
- version 22: 100% compatible 2.0-SNAPSHOT
- version 22: 100% compatible 2.0-SNAPSHOT
- version 21: 100% compatible 1.7-SNAPSHOT
- version 20: 100% compatible 1.7rc1-SNAPSHOT
- version 19: 100% compatible 1.6.1-SNAPSHOT
- version 18: 100% compatible 1.6-SNAPSHOT
- version 17: 100% compatible. 1.5
- version 16: 100% compatible. 1.5-SNAPSHOT
- version 15: 100% compatible. 1.4-SNAPSHOT
- version 14: 100% compatible. 1.4-SNAPSHOT
- version 13: 100% compatible. 1.3-SNAPSHOT
- version 12: 100% compatible. 1.3-SNAPSHOT
- version 11: 100% compatible. 1.0-SNAPSHOT
- version 10: 100% compatible. 1.0rc9-SNAPSHOT
- version 9: 100% compatible. 1.0rc9-SNAPSHOT
- version 8: 100% compatible. 1.0rc9-SNAPSHOT
- version 7: 100% compatible. 1.0rc7-SNAPSHOT - 1.0rc8
- version 6: 100% compatible. Before 1.0rc7-SNAPSHOT
- < version 6: not compatible

CSV Serialization

The CSV serialzation is the format how record are serialized in the orientdb 0. *and 1.* version.

Documents are serialized in a proprietary format (as a string) derived from JSON, but more compact. The string retrieved from the storage could be filled with spaces. This is due to the oversize feature if it is set. Just ignore the tailing spaces.

To know more about types look at Supported types.

These are the rules:

- Any string content must escape some characters:
- `" -> \"`
- `\ -> \`
- The **class**, if present, is at the begin and must end with <code>@</code>. E.g. `Customer@`
- Each **Field** must be present with its name and value separated by `:` . E.g. `name:"Barack"`
- **Fields** must be separated by `,` . E.g. `name:"Barack",surname:"Obama"`
- All **Strings** must be enclosed by `"` character. E.g. `city:"Rome"`
- All **Binary** content (like byte[must be encoded in Base64 and enclosed by underscore `_` character. E.g. `buffer:AAECAwQFBgcICQoLDA0ODxAREhMUFRYXGBkaGx` . Since v1.0rc7
- *Numbers* (integer, long, short, byte, floats, double) are formatted as strings as ouput by the Java toString() method. No thousands separator must be used. The decimal separator is always `.` Starting from version 0.9.25, if the type is not integer, a suffix is used to distinguish the right type when unmarshalled: b=byte, s=short, l=long, f=float, d=double, c=BigDecimal (since 1.0rc8). E.g. `salary:120.3f` or `code:124b` .
- Output of Floats
- Output of Doubles
- Output of BigDecimal
- **Booleans** are expressed as `true` and `false` always in lower-case. They are recognized as boolean since the text has no double quote as is the case with strings
- **Dates** must be in the POSIX format (also called UNIX format: http://en.wikipedia.org/wiki/Unix_time). Are always stored as longs but end with:
- the 't' character when it's DATETIME type (default in schema-less mode when a Date object is used). Datetime handles the maximum precision up to milliseconds. E.g. `lastUpdate:1296279468000t` is read as 2011-01-29 05:37:48
- the 'a' character when it's DATE type. Date handles up to day as precision. E.g. `lastUpdate:1306281600000a` is read as 2011-05-25 00:00:00 (Available since 1.0rc2)
- **Record ID** (link) must be prefixed by `#` . A Record Id always has the format `<cluster-id>:<cluster-position>` . E.g. `location:#3:2`
- **Embedded** documents are enclosed by parenthesis `(` and `)` characters. E.g. `(name:"rules")` . *Note: before SVN revision 2007 (0.9.24-snapshot) only* ` </code> characters were used to begin and end the embedded document.*
- **Lists** (array and list) must be enclosed by `[` and `]` characters. E.g. `[1,2,3]` , `[#10:3,#10:4]` and `[(name:"Luca")]` . Before rel.15 SET type was stored as a list, but now it uses own format (see below)
- **Sets** (collections without duplicates) must be enclosed by `<` and `>` characters. E.g. `<1,2,3>` , `<#10:3,#10:4>` and `<(name:"Luca")>` . Unable to find mention of this special use: There is a special case when use LINKSET type reported in detail in Special use of LINKSET types section. --> Before rel.15 SET type was stored as a list (see upon).
- **Maps** (as a collection of entries with key/value) must be enclosed in `{` and `}` characters. E.g. `rules: {"database":2,"database.cluster.internal":2</code>}` (NB. to set a value part of a key/value pair, set it to the text "null", without quotation marks. Eg. `rules:{"database_name":"fred","database_alias":null}`)
- **RidBags** a special collection for link management. Represented as `%(content:binary);` where the content is binary data encoded in base64. Take a look at the main page for more details.
- **Null** fields have an empty value part of the field. E.g. `salary_cloned:,salary:`

```
[<class>@][,][<field-name>:<field-value>]*
```

Simple example (line breaks introduced so it's visible on this page):

```
Profile@nick:"ThePresident",follows:[],followers:[#10:5,#10:6],name:"Barack",surname:"Obama",
location:#3:2,invitedBy:,salary_cloned:,salary:120.3f
```

Complex example used in schema (line breaks introduced so it's visible on this page):

```
name:"ORole",id:0,defaultClusterId:3,clusterIds:[3],properties:[(name:"mode",type:17,offset:0,
mandatory:false,notNull:false,min:,max:,linkedClass:,
linkedType:,index:),(name:"rules",type:12,offset:1,mandatory:false,notNull:false,min:,
max:,linkedClass:,linkedType:17,index:)]
```

Other example of ORole that uses a map (line breaks introduced so it's visible on this page):

```
ORole@name:"reader",inheritedRole:,mode:0,rules:{"database":2,"database.cluster.internal":2,"database.cluster.orole":2,"databa
se.cluster.ouser":2,
"database.class.*":2,"database.cluster.*":2,"database.query":2,"database.command":2,
"database.hook.record":2}
```

Serialization

Below the serialization of types in JSON and Binary format (always refers to latest version of the protocol).

Type	JSON format	Binary descriptor
String	0	Value ends with 'b'. Example: 23b
Short	10000	Value ends with 's'. Example: 23s
Integer	1000000	Just the value. Example: 234392
Long	1000000000	Value ends with 'l'. Example: 234392231
Float	100000.33333	Value ends with 'f'. Example: 234392.23f
Double	100.33	Value ends with 'd'. Example: 10020.2302d
Decimal	1000.3333	Value ends with 'c'. Example: 234.923c
Boolean	true	'true' or 'false'. Example: true
Date	1436983328000	Value in milliseconds ends with 'a'. Example: 1436983328000a
Datetime	1436983328000	Value in milliseconds ends with 't'. Example: 1436983328000t
Binary	base64 encoded binary, like: "A3ERjRFdc0023Kc"	Bytes surrounded with `_` characters. Example: `_ 2332322 _`
Link	#10:3	Just the RID. Example: #10:232
Link list	`[#10:3, #10:4]`	Collections values separated by commas and surrounded by brackets "[ ]". Example: [#10:3, #10:6]
Link set	Example: `[#10:3, #10:6]`	Example: `<#10:3, #10:4>`
Link map	Example: `{ "name" : "#10:3" }`	Map entries separated by commas and surrounded by curly braces "{ }". Example: `{"Jay":#10:3,"Mike":#10:6}`
Embedded	`{"Jay":"#10:3","Mike":"#10:6"}`	Embedded document serialized surrounded by parenthesis "( )". Example: `({"Jay":#10:3,"Mike":#10:6})`
Embedded list	Example: `[20, 30]`	Collections of values separated by commas and surrounded by brackets "[ ]". Example: `[20, 30]`
Embedded set	`['is', 'a', 'test']`	Collections of values separated by commas and surrounded by minor and major "<>". Example:
Embedded map	`{ "name" : "Luca" }`	Map of values separated by commas and surrounded by curly braces "{ }". Example: `{"key1":23,"key2":2332}`
Custom	base64 encoded binary, like: "A3ERjRFdc0023Kc"	-

Schemaless Serialization

The binary schemaless serialization is an attempt to define a serialization format that can serialize a document containing all the information about the structure and the data with support for partial serialization/deserialization.

The types described here are different from the types used in the binary protocol.

A serialized record has the following shape:

```
(serialization-version:byte)(class-name:string)(header:byte[])(data:byte[])
```

Version

1 byte that contains the version of the current serialization (in order to allow progressive serialization upgrades).

Class Name

A string containing the name of the class of the record. If the record has no class, `class-name` will be just an empty string.

Header

The header contains a list of fields of the serialized records, along with their position in the `data` field. The header has the following shape:

```
(field:field-definition)+
```

Where `field-definition` has this shape:

```
(field-type:varint)(field-contents:byte[])
```

The field contents depend on the value of the `field-type` varint. Once decoded to an integer, its value can be:

- `0` - signals the end of the header
- a positive number `i` - signals that what follows is a named field (and it's the length of the field name, see below)
- a negative number `i` - signals that what follows is a property (and it encodes the property id, see below)

Named fields

Named fields are encoded as:

```
(field-name:string)(pointer-to-data-structure:int32)(data-type:byte)
```

- `field-name` - the name of the field. The `field-type` varint is included in `field-name` (as per the string encoding) as mentioned above.
- `pointer-to-data-structure` - a pointer to the data structure for the current field in the `data` segment. It's `0` if the field is null.
- `data-type` - the type id of the type of the value for the current field. The supported types (with their ids) are defined in this section.

Properties

Properties are encoded as:

```
(property-id:varint)(pointer-to-data-structure:int32)
```

- `property-id` - is the `field-type` described above. It's a negative value that encodes a property id. Decoding of this value into the corresponding property id can be done using this formula: `(property-id * -1) - 1`. The property identified by this id will be found in the schema (with its name and its type), stored in the `globalProperties` field at the root of the document that represents the schema definition.
- `pointer-to-data-structure` - a pointer to the data structure for the current field in the `data` segment. It's `0` if the field is null.

Data

The data segment is where the values of the fields are stored. It's an array of data structures (each with a type described by the corresponding field).

```
(data:data-structure[])
```

Each type is serialized differently, as described below.

Type serialization

SHORT, INTEGER, LONG

Integers are encoded using the varint (with ZigZag) algorithm used by Google's ProtoBuf (and specified here).

BYTE

Bytes is stored raw, as single bytes.

BOOLEAN

Booleans are serialized using a single byte, `0` meaning false and `1` meaning true.

FLOAT

This is stored as flat byte array copying the memory from the float memory.

```
(float:byte[4])
```

DOUBLE

This is stored as flat byte array copying the memory from the double memory.

```
(double:byte[8])
```

DATETIME

The date is converted to milliseconds (from the Unix epoch) and stored as the type LONG.

DATE

The date is converted to seconds (from the Unix epoch), moved to midnight UTC+0, divided by 86400 (the number of seconds in a day) and stored as the type LONG.

STRING

Strings are stored as binary structures (encoded as UTF-8). Strings are preceded by their size (in bytes).

```
(size:varint)(string:byte[])
```

- **size** - the number of the bytes in the string stored as a varint
- **string** - the bytes of the string encoded as UTF-8

BINARY

Byte arrays are stored like STRINGs.

```
(size:varint)(bytes:byte[])
```

- **size** - the number of the bytes to store
- **bytes** - the raw bytes

EMBEDDED

Embedded documents are serialized using the protocol described in this document (recursively). The serialized document is just a byte array.

```
(serialized-document:bytes[])
```

EMBEDDEDLIST, EMBEDDEDSET

The embedded collections are stored as an array of bytes that contain the serialized document in the embedded mode.

```
(size:varint)(collection-type:byte)(items:item[])
```

- **size** - the number of items in the list/set
- **collection-type** - the type of the elements in the list or ANY if the type is unknown.
- **items** an array of values serialized by type or if the type of the collection is ANY the item will have it's own structure.

The `item` data structure has the following shape:

```
(data-type:byte)(data:byte)
```

- **data-type** - the type id of the data in the item
- **data** - the data in the item serialized according to the **data-type**

EMBEDDEDMAP

Maps can have keys with the following types:

- STRING

As of now though, **all keys are converted to STRINGs**.

An EMBEDDEDMAP is serialized as an header and a list of values.

```
(size:varint)(header:header-structure)(values:byte[][])
```

- **size** - the number of key-value pairs in the map
- **header** - serialized as `(key-type:byte)(key-value:byte[])(pointer-to-data:int32)(value-type:byte)` (where `pointer-to-data` is the same as the one in the header, offsetting from the start of the top-level document).
- **values** - the values serialized according to their type.

LINK

The link is stored as two 64 bit integers: the cluster id and the record's position in the cluster.

```
(cluster-id:varint)(record-position:varint)
```

LINKLIST, LINKSET

Link collections (lists and sets) are serialized as the size of the collection and then a list of LINKs.

```
(size:varint)(links:LINK[])
```

LINKMAP

Maps of links can have keys with the following types:

- STRING

As of now though, **all keys are converted to STRINGs**.

A LINKMAP is serialized as the number of key-value pairs and then the list of key-value pairs.

```
(size:varint)(key-value-pairs:key-value[])
```

A `key-value` pair is serialized as:

```
(key-type:byte)(key-value:byte[])(link:LINK)
```

- **key-type** - the type id of the type of the key
- **key-value** - the value of the key, serialized according to **key-type**
- **link** - the link value

DECIMAL

Decimals are converted to integers and stored as the scale and the value. For example, `10234.546` is storead as scale `3` and value `10234546`.

```
(scale:int32)(value-size:int32)(value:byte[])
```

- **scale** - the scale of the decimal.
- **value-size** - the number of bytes that form the `value`.
- **value** - the bytes representing the value of the decimal (in big-endian order).

LINKBAG

LinkBag is a dynamic collection of links that change the way to store link from inside a document to an outside structure following the number of entries in the collection.

The collection is made by two parts an header and a content:

```
  (config:byte)(uuid-low:int64)?(uuid-high:int64)?(content:rid-bag-content)
```

The header formate by:

- **config** is a byte where the first bit if 1 means embedded content if 0 means link to external collection and the second bit if is 1 are present uuid-low and uuid-high parts if 0 are missing
- **uuid-low** to ignore
- **uuid-hig** to ignore

in case RidBag Embedded the content is:

```
  (n-entries:int32)[(cluster-id:int16)(cluster-position:int64)]*
```

where:

- **n-entries** is the number of link present
- **cluster-id** is the clusterId for the link
- **cluster-position** is the clusterPosition for the link

RidBag Tree

```
  (filed-id:int64)(page-index:int64)(page-offset:int32)(n-changes)[(cluster-id:int16)(cluster-position:int64)(change-type:byte
)(change:int32)]*
```

where:

- **filed-id** part of extern collection identifier
- **page-index** part of extern collection identifier
- **page-offeset** part of extern collection identifier
- **n-changes** number of changes happened for the collection in update
- **cluster-id** the cluster id of the change
- **cluster-position** the cluster position of the change
- **change-type** the type of the change 1 abusolute change, 0 diff change
- **chage** an int that define the change type

Network Binary Protocol Commands

This is the guide to the commands you can send through the binary protocol.

See also

- List of SQL Commands
- Network Binary Protocol Specification

the commands are divided in three main groups:

- SQL (select) Query
- SQL Commands
- Script commands

SQL (Select) Query

```
(text:string)(non-text-limit:int)[(fetch-plan:string)](serialized-params:bytes[])
```

text text of the select query
non-text-limit Limit can be set in query's text, or here. This field had priority. Send -1 to use limit from query's text
fetch-plan used only for select queries, otherwise empty
serialized-params the byte[] result of the serialization of a ODocument.

Serialized Parameters ODocument content

The ODocument have to contain a field called "params" of type Map.
the Map should have as key, in case of positional perameters the numeric position of the parameter, in case of named parameters the name of the parameter and as value the value of the parameter.

SQL Commands

```
(text:string)(has-simple-parameters:boolean)(simple-paremeters:bytes[])(has-complex-parameters:boolean)(complex-parameters:byt
es[])
```

text text of the sql command
has-simple-parameters boolean flag for determine if the **simple-parameters** byte array is present or not
simple-parameters the byte[] result of the serialization of a ODocument.
has-complex-parameters boolean flag for determine if the **complex-parameters** byte array is present or not
complex-parameters the byte[] result of the serialization of a ODocument.

Serialized Simple Parameters ODocument content

The ODocument have to contain a field called "parameters" of type Map.
the Map should have as key, in case of positional perameters the numeric position of the parameter, in case of named parameters the name of the parameter and as value the value of the parameter.

Serialized Complex Parameters ODocument content

The ODocument have to contain a field called "compositeKeyParams" of type Map.
the Map should have as key, in case of positional perameters the numeric position of the parameter, in case of named parameters the name of the parameter and as value a List that is the list of composite parameters.

Script

```
(language:string)(text:string)(has-simple-parameters:boolean)(simple-paremeters:bytes[])(has-complex-parameters:boolean)(compl
ex-parameters:bytes[])
```

language the language of the script present in the text field. All the others paramenters are serialized as the SQL Commands

SQL

When it comes to query languages, SQL is the mostly widely recognized standard. The majority of developers have experience and are comfortable with SQL. For this reason Orient DB uses SQL as its query language and adds some extensions to enable graph functionality. There are a few differences between the standard SQL syntax and that supported by OrientDB, but for the most part, it should feel very natural. The differences are covered in the OrientDB SQL dialect section of this page.

If you are looking for the most efficient way to traverse a graph, we suggest using the SQL-Match instead.

Many SQL commands share the WHERE condition. Keywords and class names in OrientDB SQL are case insensitive. Field names (properties) and values are case sensitive. In the following examples keywords are in uppercase but this is not strictly required.

If you are not yet familiar with SQL, we suggest that you get the course at KhanAcademy.

For example, if you have a class `MyClass` with a field named `id`, then the following SQL statements are equivalent:

```
SELECT FROM MyClass WHERE id = 1
select from myclass where id = 1
```

The following is NOT equivalent. Notice that the field name 'ID' is not the same as 'id'.

```
SELECT FROM MyClass WHERE ID = 1
```

Automatic usage of indexes

OrientDB allows you to execute queries against any field, indexed or not-indexed. The SQL engine automatically recognizes if any indexes can be used to speed up execution. You can also query any indexes directly by using `INDEX:<index-name>` as a target. Example:

```
SELECT FROM INDEX:myIndex WHERE key = 'Jay'
```

Extra resources

- SQL expression syntax
  - Where clause
  - Operators
  - Functions
- Pagination
- Pivoting-With-Query
- SQL batch
- SQL-Match for pattern matching.

OrientDB SQL dialect

OrientDB supports SQL as a query language with some differences compared with SQL. OrientDB decided to avoid creating Yet-Another-Query-Language. Instead we started from familiar SQL with extensions to work with graphs. We prefer to focus on standards.

If you want learn SQL, there are many online courses such as:

- Online course Introduction to Databases by Jennifer Widom from Stanford university
- Introduction to SQL at W3 Schools
- Beginner guide to SQL
- SQLCourse.com
- YouTube channel Basic SQL Training by Joey Blue

To know more, look to OrientDB SQL Syntax.

Or order any book like these

JOINs

The most important difference between OrientDB and a Relational Database is that relationships are represented by `LINKS` instead of JOINs.

For this reason, the classic JOIN syntax is not supported. OrientDB uses the "dot (`.`) notation" to navigate `LINKS`. Example 1 : In SQL you might create a join such as:

```
SELECT *
FROM Employee A, City B
WHERE A.city = B.id
AND B.name = 'Rome'
```

In OrientDB an equivalent operation would be:

```
SELECT * FROM Employee WHERE city.name = 'Rome'
```

This is much more straight forward and powerful! If you use multiple JOINs, the OrientDB SQL equivalent will be an even larger benefit. Example 2: In SQL you might create a join such as:

```
SELECT *
FROM Employee A, City B, Country C,
WHERE A.city = B.id
AND B.country = C.id
AND C.name = 'Italy'
```

In OrientDB an equivalent operation would be:

```
SELECT * FROM Employee WHERE city.country.name = 'Italy'
```

Projections

In SQL projections are mandatory and you can use the star character `*` to include all of the fields. With OrientDB this type of projection is optional. Example: In SQL to select all of the columns of Customer you would write:

```
SELECT * FROM Customer
```

In OrientDB the `*` is optional:

```
SELECT FROM Customer
```

DISTINCT

In SQL, `DISTINCT` is a keyword but in OrientDB it is a function, so if your query is:

```
SELECT DISTINCT name FROM City
```

In OrientDB you would write:

```
SELECT DISTINCT(name) FROM City
```

HAVING

OrientDB does not support the `HAVING` keyword, but with a nested query it's easy to obtain the same result. Example in SQL:

```
SELECT city, sum(salary) AS salary
FROM Employee
GROUP BY city
HAVING salary > 1000
```

This groups all of the salaries by city and extracts the result of aggregates with the total salary greater than 1,000 dollars. In OrientDB the `HAVING` conditions go in a select statement in the predicate:

```
SELECT FROM ( SELECT city, SUM(salary) AS salary FROM Employee GROUP BY city ) WHERE salary > 1000
```

Select from multiple targets

OrientDB allows only one class (classes are equivalent to tables in this discussion) as opposed to SQL, which allows for many tables as the target. If you want to select from 2 classes, you have to execute 2 sub queries and join them with the `UNIONALL` function:

```
SELECT FROM E, V
```

In OrientDB, you can accomplish this with a few variable definitions and by using the `expand` function to the union:

```
SELECT EXPAND( $c ) LET $a = ( SELECT FROM E ), $b = ( SELECT FROM V ), $c = UNIONALL( $a, $b )
```

Query metadata

OrientDB provides the `metadata:` target to retrieve information about OrientDB's metadata:

- `schema` , to get classes and properties
- `indexmanager` , to get information about indexes

Query the schema

Get all the configured classes:

```
select expand(classes) from metadata:schema

----+-----------+---------+---------------+----------+--------+--------+----------+----------+------------+----------
#   |name       |shortName|defaultClusterId|strictMode|abstract|overSize|clusterIds|properties|customFields|superClass
----+-----------+---------+---------------+----------+--------+--------+----------+----------+------------+----------
0   |UserGroup  |null     |13             |false     |false   |0.0     |[1]       |[2]       |null        |V
1   |WallPost   |null     |15             |false     |false   |0.0     |[1]       |[4]       |null        |V
2   |Owner      |null     |12             |false     |false   |0.0     |[1]       |[1]       |null        |E
3   |OTriggered |null     |-1             |false     |true    |0.0     |[1]       |[0]       |null        |null
4   |E          |E        |10             |false     |false   |0.0     |[1]       |[0]       |null        |null
5   |OUser      |null     |5              |false     |false   |0.0     |[1]       |[4]       |null        |OIdentity
6   |OSchedule  |null     |7              |false     |false   |0.0     |[1]       |[7]       |null        |null
7   |ORestricted|null     |-1             |false     |true    |0.0     |[1]       |[4]       |null        |null
8   |AssignedTo |null     |11             |false     |false   |0.0     |[1]       |[1]       |null        |E
9   |V          |null     |9              |false     |false   |2.0     |[1]       |[0]       |null        |null
10  |OFunction  |null     |6              |false     |false   |0.0     |[1]       |[5]       |null        |null
11  |ORole      |null     |4              |false     |false   |0.0     |[1]       |[4]       |null        |OIdentity
12  |ORIDs      |null     |8              |false     |false   |0.0     |[1]       |[0]       |null        |null
13  |OIdentity  |null     |-1             |false     |true    |0.0     |[1]       |[0]       |null        |null
14  |User       |null     |14             |false     |false   |0.0     |[1]       |[2]       |null        |V
----+-----------+---------+---------------+----------+--------+--------+----------+----------+------------+----------
```

Get all the configured properties for the class OUser:

```
select expand(properties) from (
   select expand(classes) from metadata:schema
) where name = 'OUser'

----+--------+----+---------+--------+-------+----+----+------+------------+-----------
#   |name    |type|mandatory|readonly|notNull|min |max |regexp|customFields|linkedClass
----+--------+----+---------+--------+-------+----+----+------+------------+-----------
0   |status  |7   |true     |false   |true   |null|null|null  |null        |null
1   |roles   |15  |false    |false   |false  |null|null|null  |null        |ORole
2   |password|7   |true     |false   |true   |null|null|null  |null        |null
3   |name    |7   |true     |false   |true   |null|null|null  |null        |null
----+--------+----+---------+--------+-------+----+----+------+------------+-----------
```

Get only the configured `customFields` properties for OUser (assuming you added CUSTOM metadata like foo=bar):

```
select customFields from (
    select expand(classes) from metadata:schema
) where name="OUser"


----+------+------------
#   |@CLASS|customFields
----+------+------------
0   |null  |{foo=bar}
----+------+------------
```

Or, if you wish to get only the configured `customFields` of an attribute, like if you had a comment for the password attribute for the OUser class.

```
select customFields from (
  select expand(properties) from (
     select expand(classes) from metadata:schema
  ) where name="OUser"
) where name="password"


----+------+---------------------------------------------------
#   |@CLASS|customFields
----+------+---------------------------------------------------
0   |null  |{comment=Foo Bar your password to keep it secure!}
----+------+---------------------------------------------------
```

Query the available indexes

Get all the configured indexes:

```
select expand(indexes) from metadata:indexmanager

----+------+------+--------+---------+---------+-----------------------------------+-----------------------------------------
-------------
#   |@RID  |mapRid|clusters|type     |name     |indexDefinition                    |indexDefinitionClass
----+------+------+--------+---------+---------+-----------------------------------+-----------------------------------------
-------------
0   |#-1:-1|#2:0  |[0]     |DICTIO...|dictio...|{keyTypes:[1]}                     |com.orientechnologies.orient.core.index.O
SimpleKeyI...
1   |#-1:-1|#1:1  |[1]     |UNIQUE   |OUser....|{className:OUser,field:name,keyTy...|com.orientechnologies.orient.core.index.O
PropertyIn...
2   |#-1:-1|#1:0  |[1]     |UNIQUE   |ORole....|{className:ORole,field:name,keyTy...|com.orientechnologies.orient.core.index.O
PropertyIn...
----+------+------+--------+---------+---------+-----------------------------------+-----------------------------------------
```

SQL Commands

CRUD	Graph	Schema & Indexes	Database	Utility
SELECT	CREATE VERTEX	CREATE CLASS	CREATE CLUSTER	CREATE LINK
INSERT	CREATE EDGE	ALTER CLASS	ALTER CLUSTER	FIND REFERENCES
UPDATE	DELETE VERTEX	DROP CLASS	DROP CLUSTER	EXPLAIN
DELETE	DELETE EDGE	CREATE PROPERTY	ALTER DATABASE	CREATE FUNCTION
TRAVERSE	UPDATE EDGE	ALTER PROPERTY	CREATE DATABASE (console only)	HA STATUS
TRUNCATE CLASS	MATCH	DROP PROPERTY	DROP DATABASE (console only)	HA REMOVE SERVER
TRUNCATE CLUSTER		CREATE INDEX	OPTIMIZE DATABASE	HA SYNC DATABASE
TRUNCATE RECORD		REBUILD INDEX	CREATE USER	HA SYNC CLUSTER
		DROP INDEX	DROP USER	HA SET
		CREATE SEQUENCE	GRANT	
		ALTER SEQUENCE	REVOKE	
		DROP SEQUENCE		

SQL - `ALTER CLASS`

Updates attributes on an existing class in the schema.

Syntax

```
ALTER CLASS <class> <attribute-name> <attribute-value>
```

- **`<class>`** Defines the class you want to change.
- **`<attribute-name>`** Defines the attribute you want to change. For a list of supported attributes, see the table below.
- **`<attribute-value>`** Defines the value you want to set.

Examples

- Define a super-class:

  ```
  orientdb> ALTER CLASS Employee SUPERCLASS Person
  ```

- Define multiple inheritances:

  ```
  orientdb> ALTER CLASS Employee SUPERCLASS Person, ORestricted
  ```

  This feature was introduced in version 2.1.

- Add a super-class:

  ```
  orientdb> ALTER CLASS Employee SUPERCLASS +Person
  ```

  This feature was introduced in version 2.1.

- Remove a super-class:

  ```
  orientdb> ALTER CLASS Employee SUPERCLASS -Person
  ```

  This feature was introduced in version 2.1.

- Update the class name from `Account` to `Seller` :

  ```
  orientdb> ALTER CLASS Account NAME Seller
  ```

- Update the oversize factor on the class `Account` :

  ```
  orientdb> ALTER CLASS Account OVERSIZE 2
  ```

- Add a cluster to the class `Account` .

  ```
  orientdb> ALTER CLASS Account ADDCLUSTER account2
  ```

  In the event that the cluster does not exist, it automatically creates it.

- Remove a cluster from the class `Account` with the ID `34` :

  ```
  orientdb> ALTER CLASS Account REMOVECLUSTER 34
  ```

- Add custom properties:

```
orientdb> ALTER CLASS Post CUSTOM `onCreate.fields`="_allowRead,_allowUpdate"
orientdb> ALTER CLASS Post CUSTOM `onCreate.identityType`="role"
```

- Create a new cluster for the class `Employee` , then set the cluster selection strategy to `balanced` :

```
orientdb> CREATE CLUSTER employee_1
orientdb> ALTER CLASS Employee ADDCLUSTER employee_1
orientdb> ALTER CLASS Employee CLUSTERSELECTION balanced
```

- Convert the class `TheClass` to an abstract class:

```
orientdb> ALTER CLASS TheClass ABSTRACT true
```

> For more information see `CREATE CLASS` , `DROP CLASS` , `ALTER CLUSTER` commands. For more information on other commands, see `Console` and SQL commands.

Supported Attributes

Attribute	Type	Support	Description
`NAME`	String		Changes the class name.
`SHORTNAME`	String		Defines a short name, (that is, an alias), for the class. Use `NULL` to remove a short name assignment.
`SUPERCLASS`	String		Defines a super-class for the class. Use `NULL` to remove a super-class assignment. Beginning with version 2.1, it supports multiple inheritances. To add a new class, you can use the syntax `+<class>` , to remove it use `-<class>` .
`OVERSIZE`	Decimal number		Defines the oversize factor.
`ADDCLUSTER`	String		Adds a cluster to the class. If the cluster doesn't exist, it creates a physical cluster. Adding clusters to a class is also useful in storing records in distributed servers. For more information, see Distributed Sharding.
`REMOVECLUSTER`	String		Removes a cluster from a class. It does not delete the cluster, only removes it from the class.
`STRICTMODE`			Enalbes or disables strict mode. When in strict mode, you work in schema-full mode and cannot add new properties to a record if they're part of the class' schema definition.
`CLUSTERSELECTION`		1.7	Defines the selection strategy in choosing which cluster it uses for new records. On class creation it inherits the setting from the database. For more information, see Cluster Selection.
`CUSTOM`			Defines custom properties. Property names and values must follow the syntax `<property-name>=<value>` without spaces between the name and value. The attribute name is an indentifier, so it has to be back-tick quoted if it contains special characters (eg. dots); the value is a string, so it has to be quoted with single or double quotes.
`ABSTRACT`	Boolean		Converts class to an abstract class or the opposite.

Java API

In addition to updating a class through the console or SQL, you can also change it through the Java API, using either the Graph or Document API.

- **Graph API**:

```
// ADD A CLUSTER TO A VERTEX CLASS
graph.getVertexType("Customer").addCluster("customer_usa");

// ADD A CLUSTER TO AN EDGE CLASS
graph.getEdgeType("WorksAt").addCluster("WorksAt_2015");
```

- **Document API**

```
db.getMetadata().getSchema().getClass("Customer").addCluster("customer_usa")
```

History

2.1

- Added support for multiple inheritance.

1.7

- Added support for `CLUSTERSELECTION` that sets the strategy used on selecting the cluster to use when creating new records.

SQL - `ALTER CLUSTER`

Updates attributes on an existing cluster.

Syntax

```
ALTER CLUSTER <cluster> <attribute-name> <attribute-value>
```

- **`<cluster>`** Defines the cluster you want to change. You can use its logical name or ID. Beginning with version 2.2, you can use the wildcard `*` to update multiple clusters together.
- **`<attribute-name>`** Defines the attribute you want to change. For a list of supported attributes, see the table below.
- **`<attribute-value>`** Defines the value you want to set.

Examples

- Change the name of a cluster, using its name:

```
orientdb> ALTER CLUSTER profile NAME profile2
```

- Change the name of a cluster, using its ID:

```
orientdb> ALTER CLUSTER 9 NAME profile2
```

- Update the cluster conflict strategy to `automerge`:

```
orientdb> ALTER CLUSTER V CONFLICTSTRATEGY automerge
```

- Put cluster `V_2012` offline:

```
orientdb> ALTER CLUSTER V_2012 STATUS OFFLINE
```

- Update multiple clusters with a similar name:

```
orientdb> ALTER CLUSTER employee* status offline
```

> For more information see, `CREATE CLUSTER`, `DROP CLUSTER`, `ALTER CLUSTER` commands. For more information on other commands, see Console and SQL commands.

Supported Attributes

Name	Type	Support	Description
`NAME`	String		Changes the cluster name.
`STATUS`	String		Changes the cluster status. Allowed values are `ONLINE` and `OFFLINE` . By default, clusters are online. When offline, OrientDB no longer opens the physical files for the cluster. You may find this useful when you want to archive old data elsewhere and restore when needed.
`COMPRESSION`	String		Defines the compression type to use. Allowed values are `NOTHING` , `SNAPPY` , `GZIP` , and any other compression types registered in the `OCompressionFactory` class. OrientDB class the `compress()` method each time it saves the record to the storage, and the `uncompress()` method each time it loads the record from storage. You can also use the `OCompression` interface to manage encryption.
`USE_WAL`	Boolean		Defines whether it uses the Journal (Write Ahead Log) when OrientDB operates against the cluster.
`RECORD_GROW_FACTOR`	Integer		Defines the grow factor to save more space on record creation. You may find this useful when you update the record with additional information. In larger records, this avoids defragmentation, as OrientDB doesn't have to find new space in the event of updates with more data.
`RECORD_OVERFLOW_GROW_FACTOR`	Integer		Defines grow factor on updates. When it reaches the size limit, is uses this setting to get more space, (factor > 1).
`CONFLICTSTRATEGY`	String	2.0+	Defines the strategy it uses to handle conflicts in the event that OrientDB MVCC finds an update or a delete operation it executes against an old record. If you don't define a strategy at the cluster-level, it uses the database-level configuration. For more information on supported strategies, see the section below.

Supported Conflict Strategies

Strategy	Description
`version`	Throws an exception when versions are different. This is the default setting.
`content`	In the event that the versions are different, it checks for changes in the content, otherwise it uses the highest version to avoid throwing an exception.
`automerge`	Merges the changes.

To know more about other SQL commands, take a look at SQL commands.

SQL - `ALTER DATABASE`

Updates attributes on the current database.

Syntax

```
ALTER DATABASE <attribute-name> <attribute-value>
```

- `<attribute-name>` Defines the attribute that you want to change. For a list of supported attributes, see the section below.
- `<attribute-value>` Defines the value you want to set.

Examples

- Disable new SQL strict parser:

```
orientdb> ALTER DATABASE CUSTOM strictSQL=false
```

- Update a Graph database that was created before version 1.4:

```
orientdb> ALTER DATABASE CUSTOM useLightweightEdges=FALSE
orientdb> ALTER DATABASE CUSTOM useClassForEdgeLabel=FALSE
orientdb> ALTER DATABASE CUSTOM useClassForVertexLabel=FALSE
orientdb> ALTER DATABASE CUSTOM useVertexFieldsForEdgeLabel=FALSE
```

> Version 1.4 introduced Lightweight Edges, which was disabled by default beginning in version 2.0. Use the above commands to maintain compatibility when using older databases with newer versions of OrientDB.

> To create a database, see the `CREATE DATABASE`. To remove a database, see the `DROP DATABASE` command. For more information on other commands, see Console and SQL commands.

Supported Attributes

- **STATUS** database's status between:
  - **CLOSED** to set closed status
  - **IMPORTING** to set importing status
  - **OPEN** to set open status
- **DEFAULTCLUSTERID** to set the default cluster. By default is 2 = "default"
- **DATEFORMAT** sets the default date format. Look at Java Date Format for more information. Default is "yyyy-MM-dd"
- **DATETIMEFORMAT** sets the default date time format. Look at Java Date Format for more information. Default is "yyyy-MM-dd HH:mm:ss"
- **TIMEZONE** set the default timezone. Look at Java Date TimeZones for more information. Default is the JVM's default timezone
- **LOCALECOUNTRY** sets the default locale country. Look at Java Locales for more information. Default is the JVM's default locale country. Example: "GB"
- **LOCALELANGUAGE** sets the default locale language. Look at Java Locales for more information. Default is the JVM's default locale language. Example: "en"
- **CHARSET** set the default charset charset. Look at Java Charset for more information. Default is the JVM's default charset. Example: "utf8"
- **CLUSTERSELECTION** sets the default strategy used on selecting the cluster where to create new records. This setting is read on class creation. After creation, each class has own modifiable strategy. Supported strategies are:
  - **default**, uses always the Class's `defaultClusterId` property. This was the default before 1.7
  - **round-robin**, put the Class's configured clusters in a ring and returns a different cluster every time restarting from the first when the ring is completed
  - **balanced**, checks the records in all the clusters and returns the smaller cluster. This allows the cluster to have all the underlying clusters balanced on size. On adding a new cluster to an existent class, the new empty cluster will be filled before

the others because more empty then the others. In distributed configuration when configure clusters on different servers this setting allows to keep the server balanced with the same amount of data. Calculation of cluster size is made every 5 or more seconds to avoid to slow down insertion

- **MINIMUMCLUSTERS**, as the minimum number of clusters to create automatically when a new class is created. By default is 1, but on multi CPU/core having multiple clusters/files improves read/write performance
- **CONFLICTSTRATEGY**, (since v2.0) is the name of the strategy used to handle conflicts when OrientDB's MVCC finds an update or delete operation executed against an old record. The strategy is applied for the entire database, but single clusters can have own strategy (use ALTER CLUSTER command for this). While it's possible to inject custom logic by writing a Java class, the out of the box modes are:
    - `version`, the default, throw an exception when versions are different
    - `content`, in case the version is different checks if the content is changed, otherwise use the highest version and avoid throwing exception
    - `automerge`, merges the changes
- **CUSTOM** sets custom properties
- **VALIDATION**, (Since v2.2) disable or enable the validation for the entire database. This setting is not persistent, so at the next restart the validation is active (Default). Disabling the validation sometimes is needed in case of remote import database.

History

1.7

- Adds support for `CLUSTERSELECTION` that sets the strategy used on selecting the cluster where to create new records.
- Adds `MINIMUMCLUSTERS` to pre-create X clusters every time a new class is created.

SQL - `ALTER PROPERTY`

Updates attributes on the existing property and class in the schema.

Syntax

```
ALTER PROPERTY <class>.<property> <attribute-name> <attribute-value>
```

- **`<class>`** Defines the class to which the property belongs.
- **`<property>`** Defines the property you want to update.
- **`<attribute-name>`** Defines the attribute you want to change.
- **`<attribute-value>`** Defines the value you want to set on the attribute.

Examples

- Change the name of the property `age` in the class `Account` to `born` :

  ```
  orientdb> ALTER PROPERTY Account.age NAME born
  ```

- Update a property to make it mandatory:

  ```
  orientdb>ALTER PROPERTY Account.age MANDATORY TRUE
  ```

- Define a Regular Expression as constraint:

  ```
  orientdb> ALTER PROPERTY Account.gender REGEXP "[M|F]"
  ```

- Define a field as case-insensitive to comparisons:

  ```
  orientdb> ALTER PROPERTY Employee.name COLLATE ci
  ```

- Define a custom field on a property:

  ```
  orientdb> ALTER PROPERTY Foo.bar1 custom stereotype="visible"
  ```

- Set the default value for the current date:

  ```
  orientdb> ALTER PROPERTY Client.created DEFAULT "sysdate()"
  ```

- Define a unique id that cannot be changed after creation:

  ```
  orientdb> ALTER PROPERTY Client.id DEFAULT "uuid()" READONLY
  orientdb> ALTER PROPERTY Client.id READONLY TRUE
  ```

Supported Attributes

Attribute	Type	Support	Description
`LINKEDCLASS`	String		Defines the linked class name. Use `NULL` to remove an existing value.
`LINKEDTYPE`	String		Defines the link type. Use `NULL` to remove an existing value.
`MIN`	Integer		Defines the minimum value as a constraint. Use `NULL` to remove an existing constraint. On String attributes, it defines the minimum length of the string. On Integer attributes, it defines the minimum value for the number. On Date attributes, the earliest date accepted. For multi-value attributes (lists, sets and maps), it defines the fewest number of entries.
`MANDATORY`	Boolean		Defines whether the proprety requires a value.
`MAX`	Integer		Defines the maximum value as a constraint. Use `NULL` to remove an existing constraint. On String attributes, it defines the greatest length of the string. On Integer attributes, it defines the maximum value for the number. On Date attributes, the last date accepted. For multi-value attributes (lists, sets and maps), it defines the highest number of entries.
`NAME`	String		Defines the property name.
`NOTNULL`	Boolean		Defines whether the property can have a null value.
`REGEX`	String		Defines a Regular Expression as constraint. Use `NULL` to remove an existing constraint.
`TYPE`	String		Defines a property type.
`COLLATE`	String		Sets collate to one of the defined comparison strategies. By default, it is set to case-sensitive (`cs`). You can also set it to case-insensitive (`ci`).
`READONLY`	Boolean		Defines whether the property value is immutable. That is, if it is possible to change it after the first assignment. Use with `DEFAULT` to have immutable values on creation.
`CUSTOM`	String		Defines custom properties. The syntax for custom properties is `<custom-name> = <custom-value>` , such as `stereotype = icon` . The custom name is an identifier, so it has to be back-tick quoted if it contains special characters (eg. dots); the value is a string, so it has to be quoted with single or double quotes.
`DEFAULT`			Defines the default value or function. Feature introduced in version 2.1, (see the section above for examples). Use `NULL` to remove an existing constraint.

When altering `NAME` or `TYPE` this command runs a data update that may take some time, depending on the amount of data. Don't shut the database down during this migration. When altering the property name, the old property value is copied to the new property name.

> To create a property, use the `CREATE PROPERTY` command, to remove a property the `DROP PROPERTY` command. For more information on other commands, see Console and SQL commands.

SQL - `ALTER SEQUENCE`

Changes the sequence. Using this parameter you can change all sequence options, except for the sequence type.

This feature was introduced in version 2.2.

Syntax

```
ALTER SEQUENCE <sequence> [START <start-point>] [INCREMENT <increment>] [CACHE <cache>]
```

- **`<sequence>`** Defines the sequence you want to change.
- **`START`** Defines the initial sequence value.
- **`INCREMENT`** Defines the value to increment when it calls `.next()`.
- **`CACHE`** Defines the number of values to cache, in the event that the sequence is of the type `CACHED`.

Examples

- Alter a sequence, resetting the start value to `1000`:

  ```
  orientdb> ALTER SEQUENCE idseq START 1000
  ```

> For more information, see
>
> - `CREATE SEQUENCE`
> - `DROP SEQUENCE`
> - Sequences and Auto-increment
> - SQL Commands.

SQL - `CREATE CLASS`

Creates a new class in the schema.

Syntax

```
CREATE CLASS <class> [IF NOT EXISTS] [EXTENDS <super-class>] [CLUSTER <cluster-id>*] [CLUSTERS <total-cluster-number>] [ABSTRA
CT]
```

- **`<class>`** Defines the name of the class you want to create. You must use a letter, underscore or dollar for the first character, for all other characters you can use alphanumeric characters, underscores and dollar.
- **`IF NOT EXISTS`** (since v 2.2.13) creates the class only if it does not exist yet
- **`<super-class>`** Defines the super-class you want to extend with this class.
- **`<cluster-id>`** Defines in a comma-separated list the ID's of the clusters you want this class to use.
- **`<total-cluster-number>`** Defines the total number of clusters you want to create for this class. The default value is `1`. This feature was introduced in version 2.1.
- **`ABSTRACT`** Defines whether the class is abstract. For abstract classes, you cannot create instances of the class.

In the event that a cluster of the same name exists in the cluster, the new class uses this cluster by default. If you do not define a cluster in the command and a cluster of this name does not exist, OrientDB creates one. The new cluster has the same name as the class, but in lower-case.

When working with multiple cores, it is recommended that you use multiple clusters to improve concurrency during inserts. To change the number of clusters created by default, `ALTER DATABASE` command to update the `minimumclusters` property. Beginning with version 2.1, you can also define the number of clusters you want to create using the `CLUSTERS` option when you create the class.

Examples

- Create the class `Account`:

```
orientdb> CREATE CLASS Account
```

- Create the class `Car` to extend `Vehicle`:

```
orientdb> CREATE CLASS Car EXTENDS Vehicle
```

- Create the class `Car`, using the cluster ID of `10`:

```
orientdb> CREATE CLASS Car CLUSTER 10
```

- Create the class `Person` as an abstract class:

```
orientdb> CREATE CLASS Person ABSTRACT
```

Cluster Selection Strategies

When you create a class, it inherits the cluster selection strategy defined at the database-level. By default this is set to round-robin. You can change the database default using the `ALTER DATABASE` command and the selection strategy for the class using the `ALTER CLASS` command.

Supported Strategies:

Strategy	Description
`default`	Selects the cluster using the class property `defaultClusterId`. This was the default selection strategy before version 1.7.
`round-robin`	Selects the next cluster in a circular order, restarting once complete.
`balanced`	Selects the smallest cluster. Allows the class to have all underlying clusters balanced on size. When adding a new cluster to an existing class, it fills the new cluster first. When using a distributed database, this keeps the servers balanced with the same amount of data. It calculates the cluster size every five seconds or more to avoid slow-downs on insertion.

> For more information, see
>
> - `ALTER CLASS`
> - `DROP CLASS`
> - `CREATE CLUSTER`
> - SQL Commands
> - Console Commands

SQL - `CREATE CLUSTER`

Creates a new cluster in the database. Once created, you can use the cluster to save records by specifying its name during saves. If you want to add the new cluster to a class, follow its creation with the `ALTER CLASS` command, using the `ADDCLUSTER` option.

Syntax

```
CREATE CLUSTER <cluster> [ID <cluster-id>]
```

- **`<cluster>`** Defines the name of the cluster you want to create. You must use a letter for the first character, for all other characters, you can use alphanumeric characters, underscores and dashes.
- **`<cluster-id>`** Defines the numeric ID you want to use for the cluster.

Examples

- Create the cluster `account` :

  ```
  orientdb> CREATE CLUSTER account
  ```

> For more information see,
>
> - `DROP CLUSTER`
> - SQL Commands
> - Console Commands

SQL - `CREATE EDGE`

Creates a new edge in the database.

Syntax

```
CREATE EDGE <class> [CLUSTER <cluster>]
FROM <rid> | ( <query> ) | [ <rid> (, <rid>)* ]
TO <rid> | ( <query> ) | [ <rid> (, <rid>)* ]
[ SET <field> = <expression>[,]* ] | CONTENT {<JSON>}
[RETRY <retry> [WAIT <pauseBetweenRetriesInMs]] [BATCH <batch-size>]
```

- **`<class>`** Defines the class name for the edge. Use the default edge class `E` in the event that you don't want to use sub-types.
- **`<cluster>`** Defines the cluster in which you want to store the edge.
- **`JSON`** Provides JSON content to set as the record. Use this instead of entering data field by field.
- **`RETRY`** Define the number of retries to attempt in the event of conflict, (optimistic approach).
- **`WAIT`** Defines the time to delay between retries in milliseconds.
- **`BATCH`** Defines whether it breaks the command down into smaller blocks and the size of the batches. This helps to avoid memory issues when the number of vertices is too high. By default, it is set to `100`. This feature was introduced in version 2.1.3.

Edges and Vertices form the main components of a Graph database. OrientDB supports polymorphism on edges. The base class for an edge is `E`.

Beginning with version 2.1, when no edges are created OrientDB throws a `OComm andExecutionException` error. This makes it easier to integrate edge creation in transactions. In such cases, if the source or target vertices don't exist, it rolls back the transaction. (Prior to 2.1, no such error is thrown.)

Examples

- Create an edge of the class `E` between two vertices:

  ```
  orientdb> CREATE EDGE FROM #10:3 TO #11:4
  ```

- Create an edge of the class `E` between multiple vertices:

  ```
  orientdb> CREATE EDGE FROM [#10:3, #10:4] TO [#11:4, #11:5]
  ```

- Create a new edge type and an edge of the new type:

  ```
  orientdb> CREATE CLASS E1 EXTENDS E
  orientdb> CREATE EDGE E1 FROM #10:3 TO #11:4
  ```

- Create an edge in a specific cluster:

  ```
  orientdb> CREATE EDGE E1 CLUSTER EuropeEdges FROM #10:3 TO #11:4
  ```

- Create an edge and define its properties:

  ```
  orientdb> CREATE EDGE FROM #10:3 TO #11:4 SET brand = 'fiat'
  ```

- Create an edge of the type `E1` and define its properties:

  ```
  orientdb> CREATE EDGE E1 FROM #10:3 TO #11:4 SET brand = 'fiat', name = 'wow'
  ```

- Create edges of the type `Watched` between all action movies in the database and the user Luca, using sub-queries:

```
orientdb> CREATE EDGE Watched FROM (SELECT FROM account WHERE name = 'Luca') TO
          (SELECT FROM movies WHERE type.name = 'action')
```

- Create an edge using JSON content:

```
orientdb> CREATE EDGE E FROM #22:33 TO #22:55 CONTENT { "name": "Jay",
          "surname": "Miner" }
```

> For more information, see
>
> - `CREATE VERTEX`

Control Vertices Version Increment

Creating and deleting edges causes OrientDB to update versions involved in the vertices. To avoid this behavior, use the Bonsai Structure.

By default, OrientDB uses Bonsai as soon as it reaches the threshold to optimize operation. To always use Bonsai on your database, either set this configuration on the JVM or in the `orientdb-server-config.xml` configuration file.

```
$ java -DridBag.embeddedToSbtreeBonsaiThreshold=-1
```

Alternatively, in your Java application use the following call before opening the database:

```
OGlobalConfiguration.RID_BAG_EMBEDDED_TO_SBTREEBONSAI_THRESHOLD.setValue(-1);
```

> For more information, see Concurrency on Adding Edges.

❗	**When running a distributed database, the SB Tree index algorithm is not supported.**

History

2.0

- Disables Lightweight Edges in new databases by default. This command now creates a regular edge.

1.4

- Command uses the Blueprints API. If you are in Java using the `OGraphDatabase` API, you may experience some differences in how OrientDB manages edges.

  > To force the command to work with the older API, change the GraphDB settings, as described in Graph backwards compatibility.

1.2

- Implements support for query and collection of Record ID's in the `FROM...TO` clause.

1.1

- Initial version.

SQL - `CREATE FUNCTION`

Creates a new Server-side function. You can execute Functions from SQL, HTTP and Java.

Syntax

```
CREATE FUNCTION <name> <code>
                [PARAMETERS [<comma-separated list of parameters' name>]]
                [IDEMPOTENT true|false]
                [LANGUAGE <language>]
```

- **`<name>`** Defines the function name.
- **`<code>`** Defines the function code.
- **`PARAMETERS`** Defines a comma-separated list of parameters bound to the execution heap. You must wrap your parameters list in square brackets [].
- **`IDEMPOTENT`** Defines whether the function can change the database status. This is useful given that HTTP GET can call `IDEMPOTENT` functions, while others are called by HTTP POST. By default, it is set to `FALSE`.
- **`LANGUAGE`** Defines the language to use. By default, it is set to JavaScript.

Examples

- Create a function `test()` in JavaScript, which takes no parameters:

  ```
  orientdb> CREATE FUNCTION test "print('\nTest!')"
  ```

- Create a function `test(a,b)` in JavaScript, which takes 2 parameters:

  ```
  orientdb> CREATE FUNCTION test "return a + b;" PARAMETERS [a,b]
  ```

- Create a function `allUsersButAdmin` in SQL, which takes with no parameters:

  ```
  orientdb> CREATE FUNCTION allUsersButAdmin "SELECT FROM ouser WHERE name <>
            'admin'" LANGUAGE SQL
  ```

> For more information, see
>
> - Functions
> - SQL Commands
> - Console Commands

SQL - `CREATE INDEX`

Creates a new index. Indexes can be

- **Unique** Where they don't allow duplicates.
- **Not Unique** Where they allow duplicates.
- **Full Text** Where they index any single word of text.

> There are several index algorithms available to determine how OrientDB indexes your database. For more information on these, see Indexes.

Syntax

```
CREATE INDEX <name> [ON <class> (<property>)] <index-type> [<key-type>]
             METADATA [{<json>}]
```

- **`<name>`** Defines the logical name for the index. If a schema already exists, you can use `<class>.<property>` to create automatic indexes bound to the schema property. Because of this, you cannot use the period " `.` " character in index names.
- **`<class>`** Defines the class to create an automatic index for. The class must already exist.
- **`<property>`** Defines the property you want to automatically index. The property must already exist.

  > If the property is one of the Map types, such as `LINKMAP` or `EMBEDDEDMAP` , you can specify the keys or values to use in index generation, using the `BY KEY` or `BY VALUE` clause.

- **`<index-type>`** Defines the index type you want to use. For a complete list, see Indexes.

- **`<key-type>`** Defines the key type. With automatic indexes, the key type is automatically selected when the database reads the target schema property. For manual indexes, when not specified, it selects the key at run-time during the first insertion by reading the type of the class. In creating composite indexes, it uses a comma-separated list of types.
- **`METADATA`** Defines additional metadata through JSON.

To create an automatic index bound to the schema property, use the `ON` clause, or use a `<class>.<property>` name for the index. In order to create an index, the schema must already exist in your database.

In the event that the `ON` and `<key-type>` clauses both exist, the database validates the specified property types. If the property types don't equal those specified in the key type list, it throws an exception.

> You can use list key types when creating manual composite indexes, but bear in mind that such indexes are not yet fully supported.

Examples

- Create a manual index to store dates:

  ```
  orientdb> CREATE INDEX mostRecentRecords UNIQUE DATE
  ```

- Create an automatic index bound to the new property `id` in the class `User` :

  ```
  orientdb> CREATE PROPERTY User.id BINARY
  orientdb> CREATE INDEX User.id UNIQUE
  ```

- Create a series automatic indexes for the `thumbs` property in the class `Movie` :

  ```
  orientdb> CREATE INDEX thumbsAuthor ON Movie (thumbs) UNIQUE
  orientdb> CREATE INDEX thumbsAuthor ON Movie (thumbs BY KEY) UNIQUE
  orientdb> CREATE INDEX thumbsValue ON Movie (thumbs BY VALUE) UNIQUE
  ```

- Create a series of properties and on them create a composite index:

```
orientdb> CREATE PROPERTY Book.author STRING
orientdb> CREATE PROPERTY Book.title STRING
orientdb> CREATE PROPERTY Book.publicationYears EMBEDDEDLIST INTEGER
orientdb> CREATE INDEX books ON Book (author, title, publicationYears) UNIQUE
```

- Create an index on an edge's date range:

```
orientdb> CREATE CLASS File EXTENDS V
orientdb> CREATE CLASS Has EXTENDS E
orientdb> CREATE PROPERTY Has.started DATETIME
orientdb> CREATE PROPERTY Has.ended DATETIME
orientdb> CREATE INDEX Has.started_ended ON Has (started, ended) NOTUNIQUE
```

> You can create indexes on edge classes only if they contain the begin and end date range of validity. This is use case is very common with historical graphs, such as the example above.

- Using the above index, retrieve all the edges that existed in the year 2014:

```
orientdb> SELECT FROM Has WHERE started >= '2014-01-01 00:00:00.000' AND
          ended < '2015-01-01 00:00:00.000'
```

- Using the above index, retrieve all edges that existed in 2014 and write them to the parent file:

```
orientdb> SELECT outV() FROM Has WHERE started >= '2014-01-01 00:00:00.000'
          AND ended < '2015-01-01 00:00:00.000'
```

- Using the above index, retrieve all the 2014 edges and connect them to children files:

```
orientdb> SELECT inV() FROM Has WHERE started >= '2014-01-01 00:00:00.000'
          AND ended < '2015-01-01 00:00:00.000'
```

- Create an index that includes null values.

  Before version 2.1 OrientDB indexes ignored null values by default. In V 2.2 we changed this default, so now all the null values are indexed by default, that means that null values are also candidates for unique key checks. To exclude indexing of null values, you can use `{ ignoreNullValues: true }` as metadata.

```
orientdb> CREATE INDEX addresses ON Employee (address) NOTUNIQUE
           METADATA { ignoreNullValues : true }
```

> For more information, see
>
> - `DROP INDEX`
> - Indexes
> - SQL commands

SQL - `CREATE LINK`

Creates a link between two simple values.

Syntax

```
CREATE LINK <link> TYPE [<link-type>] FROM <source-class>.<source-property> TO <destination-class>.<destination-property> [INV
ERSE]
```

- `<link` > Defines the property for the link. When not expressed, the link overwrites the `<destination-property>` field.
- `<link-type>` Defines the type for the link. In the event of an inverse relationship, (the most common), you can specify `LINKSET` or `LINKLIST` for 1-*n* relationships.
- `<source-class>` Defines the class to link from.
- `<source-property>` Defines the property to link from.
- `<destination-class>` Defines the class to link to.
- `<destination-property>` Defines the property to link to.
- `INVERSE` Defines whether to create a connection on the opposite direction. This option is common when creating 1-*n* relationships from a Relational database, where they are mapped at the opposite direction.

Example

- Create an inverse link between the classes `Comments` and `Post` :

```
orientdb> CREATE LINK comments TYPE LINKSET FROM Comments.PostId TO Posts.Id
          INVERSE
```

> For more information, see
>
> - Relationships
> - Importing from Relational Databases
> - SQL Commands

Conversion from Relational Databases

You may find this useful when imported data from a Relational database. In the Relational world, the database uses links to resolve foreign keys. In general, this is not the way to create links, but rather a way to convert two values in two different classes into a link.

As an example, consider a Relational database where the table `Post` has a 1-*n* relationship with the table `Comment` . That is `Post 1 --->  * Comment` , such as:

```
reldb> SELECT * FROM Post;

+----+----------------+
| Id | Title          |
+----+----------------+
| 10 | NoSQL movement |
+----+----------------+
| 20 | New OrientDB   |
+----+----------------+

reldb> SELECT * FROM Comment;

+----+--------+--------------+
| Id | PostID | Text         |
+----+--------+--------------+
|  0 | 10     | First        |
+----+--------+--------------+
|  1 | 10     | Second       |
+----+--------+--------------+
| 21 | 10     | Another      |
+----+--------+--------------+
| 41 | 20     | First again  |
+----+--------+--------------+
| 82 | 20     | Second Again |
+----+--------+--------------+
```

In OrientDB, instead of a separate table for the relationship, you use a direct relationship as your object model. Meaning that the database navigates from `Post` to `Comment` and not vice versa, as with Relational databases. To do so, you would also need to create the link with the `INVERSE` option.

SQL - `CREATE PROPERTY`

Creates a new property in the schema. It requires that the class for the property already exist on the database.

Syntax

```
CREATE PROPERTY
<class>.<property>
[IF NOT EXISTS]
<type>
[<link-type>|<link-class>]
( <property constraint> [, <property-constraint>]* )
[UNSAFE]
```

- **`<class>`** Defines the class for the new property.
- **`IF NOT EXISTS`** (since v 2.2.13) Creates the property only if it does not exist. If it does, the statement just does nothing.
- **`<property>`** Defines the logical name for the property.
- **`<type>`** Defines the property data type. For supported types, see the table below.
- **`<link-type>`** Defines the contained type for container property data types. For supported link types, see the table below.
- **`<link-class>`** Defines the contained class for container property data types. For supported link types, see the table below.
- **`<property-constraint>`** See `ALTER PROPERTY` `<attribute-name> [ <attribute-value> ]` (since V2.2.3)
- **`UNSAFE`** Defines whether it checks existing records. On larger databases, with millions of records, this could take a great deal of time. Skip the check when you are sure the property is new. Introduced in version 2.0.

> When you create a property, OrientDB checks the data for property and type. In the event that persistent data contains incompatible values for the specified type, the property creation fails. It applies no other constraints on the persistent data.

Examples

- Create the property `name` of the string type in the class `User` :

  ```
  orientdb> CREATE PROPERTY User.name STRING
  ```

- Create a property formed from a list of strings called `tags` in the class `Profile` :

  ```
  orientdb> CREATE PROPERTY Profile.tags EMBEDDEDLIST STRING
  ```

- Create the property `friends` , as an embedded map in a circular reference:

  ```
  orientdb> CREATE PROPERTY Profile.friends EMBEDDEDMAP Profile
  ```

- Create the property `name` of the string type in the class `User` , mandatory, with minimum and maximum length (since V2.2.3):

  ```
  orientdb> CREATE PROPERTY User.name STRING (MANDATORY TRUE, MIN 5, MAX 25)
  ```

> For more information, see
>
> - `DROP PROPERTY`
> - SQL Commands
> - Console Commands

Supported Types

OrientDB supports the following data types for standard properties:

`BOOLEAN`	`SHORT`	`DATE`	`DATETIME`	`BYTE`
`INTEGER`	`LONG`	`STRING`	`LINK`	`DECIMAL`
`DOUBLE`	`FLOAT`	`BINARY`	`EMBEDDED`	`LINKBAG`

It supports the following data types for container properties.

`EMBEDDEDLIST`	`EMBEDDEDSET`	`EMBEDDEDMAP`
`LINKLIST`	`LINKSET`	`LINKMAP`

For these data types, you can optionally define the contained type and class. The supported link types are the same as the standard property data types above.

SQL - `CREATE SEQUENCE`

Creates a new sequence. Command introduced in version 2.2.

Syntax

```
CREATE SEQUENCE <sequence> TYPE <CACHED|ORDERED> [START <start>]
[INCREMENT <increment>] [CACHE <cache>]
```

- **`<sequence>`** Logical name for the sequence to cache.
- **`TYPE`** Defines the sequence type. Supported types are,
  - `CACHED` For sequences where it caches N items on each node to improve performance when you require many calls to the `.next()` method. (Bear in mind, this may create holes with numeration).
  - `ORDERED` For sequences where it draws on a new value with each call to the `.next()` method.
- **`START`** Defines the initial value of the sequence.
- **`INCREMENT`** Defines the increment for each call of the `.next()` method.
- **`CACHE`** Defines the number of value to pre-cache, in the event that you use the cached sequence type.

Examples

- Create a new sequence to handle id numbers:

```
orientdb> CREATE SEQUENCE idseq TYPE ORDERED
```

- Use the new sequence to insert id values

```
orientdb> INSERT INTO Account SET id = sequence('idseq').next()
```

> For more information, see
>
> - `ALTER SEQUENCE`
> - DROP SEQUENCE
> - Sequences and Auto-increment
> - SQL commands.

SQL - `CREATE USER`

Creates a user in the current database, using the specified password and an optional role. When the role is unspecified, it defaults to `writer`.

The command was introduced in version 2.2. It is a simple wrapper around the `OUser` and `ORole` classes. More information is available at Security.

Syntax

```
CREATE USER <user> IDENTIFIED BY <password> [ROLE <role>]
```

- **`<user>`** Defines the logical name of the user you want to create.
- **`<password>`** Defines the password to use for this user.
- **`ROLE`** Defines the role you want to set for the user. For multiple roles, use the following syntax: `['author', 'writer']`.

Examples

- Create a new admin user called `Foo` with the password `bar`:

  ```
  orientdb> CREATE USER Foo IDENTIFIED BY bar ROLE admin
  ```

- Create a new user called `Bar` with the password `foo`:

  ```
  orientdb> CREATE USER Bar IDENTIFIED BY Foo
  ```

> For more information, see
>
> - Security
> - `DROP USER`
> - SQL Commands

SQL - `CREATE VERTEX`

Creates a new vertex in the database.

The Vertex and Edge are the main components of a Graph database. OrientDB supports polymorphism on vertices. The base class for a vertex is `V`.

Syntax

```
CREATE VERTEX [<class>] [CLUSTER <cluster>] [SET <field> = <expression>[,]*]
```

- **`<class>`** Defines the class to which the vertex belongs.
- **`<cluster>`** Defines the cluster in which it stores the vertex.
- **`<field>`** Defines the field you want to set.
- **`<expression>`** Defines the express to set for the field.

Examples

- Create a new vertex on the base class `V`:

```
orientdb> CREATE VERTEX
```

- Create a new vertex class, then create a vertex in that class:

```
orientdb> CREATE CLASS V1 EXTENDS V
orientdb> CREATE VERTEX V1
```

- Create a new vertex within a particular cluster:

```
orientdb> CREATE VERTEX V1 CLUSTER recent
```

- Create a new vertex, defining its properties:

```
orientdb> CREATE VERTEX SET brand = 'fiat'
```

- Create a new vertex of the class `V1`, defining its properties:

```
orientdb> CREATE VERTEX V1 SET brand = 'fiat', name = 'wow'
```

- Create a vertex using JSON content:

```
orientdb> CREATE VERTEX Employee CONTENT { "name" : "Jay", "surname" : "Miner" }
```

> For more information, see
>
> - `CREATE EDGE`
> - SQL Commands

History

1.4

- Command begins using the Blueprints API. When using Java with the OGraphDatabase API, you may experience unexpected results in how it manages edges.

To force the command to work with the older API, update the GraphDB settings, use the `ALTER DATABASE` command.

1.1

- Initial implementation of feature.

SQL - `DELETE`

Removes one or more records from the database. You can refine the set of records that it removes using the `WHERE` clause.

> **NOTE**: Don't use `DELETE` to remove Vertices or Edges. Instead use the `DELETE VERTEX` or `DELETE EDGE` commands, which ensures the integrity of the graph.

Syntax:

```
DELETE FROM <Class>|CLUSTER:<cluster>|INDEX:<index> [LOCK <default|record>] [RETURN <returning>]
  [WHERE <Condition>*] [LIMIT <MaxRecords>] [TIMEOUT <timeout>]
```

- **`LOCK`** Determines how the database locks the record between load and delete. It takes one of the following values:
  - `DEFAULT` Defines no locks during the delete. In the case of concurrent deletes, the MVCC throws an exception.
  - `RECORD` Defines record locks during the delete.
- **`RETURN`** Defines what values the database returns. It takes one of the following values:
  - `COUNT` Returns the number of deleted records. This is the default option.
  - `BEFORE` Returns the number of records before the removal.
- **`WHERE`** Filters to the records you want to delete.
- **`LIMIT`** Defines the maximum number of records to delete.
- **`TIMEOUT`** Defines the time period to allow the operation to run, before it times out.

Examples:

- Delete all recods with the surname `unknown`, ignoring case:

  ```
  orientdb> DELETE FROM Profile WHERE surname.toLowerCase() = 'unknown'
  ```

For more information, see SQL commands.

SQL - `DELETE EDGE`

Removes edges from the database. This is the equivalent of the `DELETE` command, with the addition of checking and maintaining consistency with vertices by removing all cross-references to the edge from both the `in` and `out` vertex properties.

Syntax

```
DELETE EDGE
    ( <rid>
      |
      [<rid> (, <rid>)*]
      |
      ( [ FROM (<rid> | <select_statement> ) ] [ TO ( <rid> | <select_statement> ) ] )
      |
      [<class>]
    (
    [WHERE <conditions>]
    [LIMIT <MaxRecords>]
    [BATCH <batch-size>]
```

- **`FROM`** Defines the starting point vertex of the edge to delete.
- **`TO`** Defines the ending point vertex of the edge to delete.
- **`WHERE`** Defines the filtering conditions.
- **`LIMIT`** Defines the maximum number of edges to delete.
- **`BATCH`** Defines the block size for the operation, allowing you to break large transactions down into smaller units to reduce resource demands. Its default is `100`. Feature introduced in 2.1.

Examples

- Delete an edge by its RID:

  ```
  orientdb> DELETE EDGE #22:38482
  ```

- Delete edges by RIDs:

  ```
  orientdb> DELETE EDGE [#22:38482,#23:232,#33:2332]
  ```

- Delete edges where the data is a property that might exist in one or more edges between two vertices:

  ```
  orientdb> DELETE EDGE FROM #11:101 TO #11:117 WHERE date >= "2012-01-15"
  ```

- Delete edges filtering by the edge class:

  ```
  orientdb> DELETE EDGE FROM #11:101 TO #11:117 WHERE @class = 'Owns' AND comment
            LIKE "regex of forbidden words"
  ```

- Delete edge filtering by the edge class and date:

  ```
  orientdb> DELETE EDGE Owns WHERE date < "2011-11"
  ```

  Note that this syntax is faster than filtering the class through the `WHERE` clause.

- Delete edges where `in.price` shows the condition on the `to` vertex for the edge:

  ```
  orientdb> DELETE EDGE Owns WHERE date < "2011-11" AND in.price >= 202.43
  ```

- Delete edges in blocks of one thousand per transaction.

```
orientdb> DELETE EDGE Owns WHERE date < "2011-11" BATCH 1000
```

This feature was introduced in version 2.1.

> For more information, see
>
> - `DELETE`
> - SQL Commands

Use Cases

Controling Vertex Version Increments

Creating and deleting edges causes OrientDB to increment versions on the involved vertices. You can prevent this operation by implementing the Bonsai Structure.

By default, OrientDB only uses Bonsai as soon as it reaches the threshold, in order to optimize operation. To always use Bonsai, configure it on the JVM or in the `orientdb-server-config.xml` configuration file.

```
$ javac ... -DridBag.embeddedToSbtreeBonsaiThreshold=-1
```

To implement it in Java, add the following line to your application at a point before opening the database:

```
OGlobalConfiguration.RID_BAG_EMBEDDED_TO_SBTREEBONSAI_THRESHOLD.setValue(-1);
```

For more information, see Concurrency on Adding Edges.

NOTE: When using a distributed database, OrientDB does not support SBTree indexes. In these environments, you must set `ridBag.embeddedToSbtreeBonsaiThreshold=Integer.MAX\_VALUE` to avoid replication errors._

Deleting Edges from a Sub-query

Consider a situation where you have an edge with a Record ID of `#11:0` that you want to delete. In attempting to do so, you run the following query:

```
orientdb> DELETE EDGE FROM (SELECT FROM #11:0)
```

This does **not** delete the edge. To delete edges using sub-queries, you have to use a somewhat different syntax. For instance,

```
orientdb> DELETE EDGE E WHERE @rid IN (SELECT FROM #11:0)
```

This removes the edge from your database.

To delete edges from sub-query given a class:

```
orientdb> DELETE EDGE E WHERE @rid IN (SELECT @rid FROM E)
```

Deleting Edges through Java

When a `User` node follows a `company` node, we create an edge between the user and the company of the type `followCompany` and `CompanyFollowedBy` classes. We can then remove the relevant edges through Java.

```
node1 is User node,
node2 is company node

OGraphDatabase rawGraph = orientGraph.getRawGraph();
String[] arg={"followCompany,"CompanyFollowedBy"};
Set<OIdentifiable> edges=rawGraph.getEdgesBetweenVertexes(node1, node2,null,arg);
for (OIdentifiable oIdentifiable : edges) {
    **rawGraph.removeEdge(oIdentifiable);
}
```

History

2.1

- Implements support for the option `BATCH` clause

1.4

- Command implements the Blueprints API. In the event that you are working in Java using the OGraphDatabase API, you may experience some unexpected results in how edges are managed between versions. To force the command to use the older API, change the GraphDB settings, as described on the [`ALTER DATABASE`])SQL-Alter-Database.md) command examples.

1.1

- First implementation of the feature.

SQL - `DELETE VERTEX`

Removes vertices from the database. This is the equivalent of the `DELETE` command, with the addition of checking and maintaining consistency with edges, removing all cross-references to the deleted vertex in all edges involved.

Syntax

```
DELETE VERTEX <vertex> [WHERE <conditions>] [LIMIT <MaxRecords>>] [BATCH <batch-size>]
```

- **`<vertex>`** Defines the vertex that you want to remove, using its Class, Record ID, or through a sub-query using the `FROM (<sub-query)` clause.
- **`WHERE`** Filter condition to determine which records the command removes.
- **`LIMIT`** Defines the maximum number of records to remove.
- **`BATCH`** Defines how many records the command removes at a time, allowing you to break large transactions into smaller blocks to save on memory usage. By default, it operates on blocks of 100.

Example

- Remove the vertex and disconnect all vertices that point towards it:

```
orientdb> DELETE VERTEX #10:231
```

- Remove all user accounts marked with an incoming edge on the class `BadBehaviorInForum`:

```
orientdb> DELETE VERTEX Account WHERE in.@Class CONTAINS
          'BadBehaviorInForum'
```

- Remove all vertices from the class `EmailMessages` marked with the property `isSpam`:

```
orientdb> DELETE VERTEX EMailMessage WHERE isSpam = TRUE
```

- Remove vertices of the class `Attachment`, where the vertex has an edge of the class `HasAttachment` where the property `date` is set before 1990 and the vertex `Email` connected to class `Attachment` with the condition that its property `from` is set to `bob@example.com`:

```
orientdb> DELETE VERTEX Attachment WHERE in[@Class = 'HasAttachment'].date
          <= "1990" AND in.out[@Class = "Email"].from = 'some...@example.com'
```

- Remove vertices in blocks of one thousand:

```
orientdb> DELETE VERTEX v BATCH 1000
```

  This feature was introduced in version 2.1.

Quick deletion of an entire class

In the case you want to delete one or more classes of vertices and all the connected edges resides only on particular classes, you could use the `TRUNCATE CLASS` command against both vertex and edge classes by specifying the `UNSAFE` keyword. `TRUNCATE CLASS` is much faster than `DELETE VERTEX`, because it doesn't take in consideration the removal of the edges. Use `TRUNCATE CLASS` only when you are certain that there will not be broken edges on other vertices instances. Example of deleting all the instances of vertices classes `Email` and `Attachment` and the edge class that connect them `HasAttachment`:

```
orientdb> TRUNCATE CLASS Email UNSAFE
orientdb> TRUNCATE CLASS HasAttachment UNSAFE
orientdb> TRUNCATE CLASS Attachment UNSAFE
```

History

Version 2.1

- Introduces the optional `BATCH` clause for managing batch size on the operation.

Version 1.4

- Command begins using the Blueprints API. When working in Java using the OGraphDatabase API, you may experience differences in how the database manages edges. To force the command to work with the older API, change the Graph DB settings using `ALTER DATABASE`.

Version 1.1

- Initial version.

SQL - `DROP CLASS`

Removes a class from the schema.

Syntax

```
DROP CLASS <class> [IF EXISTS] [ UNSAFE ]
```

- **`<class>`** Defines the class you want to remove.
- **`IF EXISTS`** (since v 2.2.13) Drops the class only if it exists (does nothing if it doesn't)
- **`UNSAFE`** Defines whether the command drops non-empty edge and vertex classes. Note, this can disrupt data consistency. Be sure to create a backup before running it.

> **NOTE**: Bear in mind, that the schema must remain coherent. For instance, avoid removing calsses that are super-classes to others. This operation won't delete the associated cluster.

Examples

- Remove the class `Account` :

  ```
  orientdb> DROP CLASS Account
  ```

> For more information, see
>
> - `CREATE CLASS`
> - `ALTER CLASS`
> - `ALTER CLUSTER`
> - SQL Commands
> - Console Commands

SQL - `DROP CLUSTER`

Removes the cluster and all of its content. This operation is permanent and cannot be rolled back.

Syntax

```
DROP CLUSTER <cluster-name>|<cluster-id>
```

- **`<cluster-name>`** Defines the name of the cluster you want to remove.
- **`<cluster-id>`** Defines the ID of the cluster you want to remove.

Examples

- Remove the cluster `Account` :

  ```
  orientdb> DROP CLUSTER Account
  ```

> For more information, see
>
> - `CREATE CLUSTER`
> - `ALTER CLUSTER`
> - `DROP CLASS`
> - SQL Commands
> - Console Commands

SQL - `DROP INDEX`

Removes an index from a property defined in the schema.

If the index does not exist, this call just returns with no errors.

Syntax

```
DROP INDEX <index>|<class>.<property>
```

- `<index>` Defines the name of the index.
- `<class>` Defines the class the index uses.
- `<property>` Defines the property the index uses.

Examples

- Remove the index on the `Id` property of the `Users` class:

  ```
  orientdb> DROP INDEX Users.Id
  ```

> For more information, see
>
> - `CREATE INDEX`
> - Indexes
> - SQL Commands

SQL - `DROP PROPERTY`

Removes a property from the schema. Does not remove the property values in the records, it just changes the schema information. Records continue to have the property values, if any.

Syntax

```
DROP PROPERTY <class>.<property> [IF EXISTS] [FORCE]
```

- **`<class>`** Defines the class where the property exists.
- **`IF EXISTS`** (since 2.2.13) Drops the property only if it exists. If it doesn't, the statement does nothing
- **`<property>`** Defines the property you want to remove.
- **FORCE** In case one or more indexes are defined on the property, the command will throw an exception. Use FORCE to drop indexes together with the property

Examples

- Remove the `name` property from the class `User`:

  ```
  orientdb> DROP PROPERTY User.name
  ```

> For more information, see
>
> - `CREATE PROPERTY`
> - SQL Commands
> - Console Commands

SQL - `DROP SEQUENCE`

Removes a sequence. This feature was introduced in version 2.2.

Syntax

```
DROP SEQUENCE <sequence>
```

- `<sequence>` Defines the name of the sequence you want to remove.

Examples

- Remove the sequence `idseq`:

  ```
  orientdb> DROP SEQUENCE idseq
  ```

> For more information, see
>
> - `CREATE SEQUENCE`
> - `DROP SEQUENCE`
> - Sequences and auto increment
> - SQL commands

SQL - `DROP USER`

Removes a user from the current database. This feature was introduced in version 2.2

Syntax

```
DROP USER <user>
```

- `<user>` Defines the user you want to remove.

> **NOTE**: This is a wrapper on the class `OUser`. For more information, see Security.

Examples

- Remove the user `Foo`:

  ```
  orientdb> DROP USER Foo
  ```

> For more information, see,
>
> - `CREATE USER`
> - SQL commands

SQL - `EXPLAIN`

Profiles any command and returns a JSON data on the result of its execution. You may find this useful to see why queries are running slow. Use it as a keyword before any command that you want to profile.

Syntax

```
EXPLAIN <command>
```

- `<command>` Defines the command that you want to profile.

Examples

- Profile a query that executes on a class without indexes:

```
orientdb> EXPLAIN SELECT FROM Account

Profiled command '{documentReads:1126, documentReadsCompatibleClass:1126,
recordReads:1126, elapsed:209, resultType:collection, resultSize:1126}'
in 0,212000 sec(s).
```

- Profile a query that executes on a class with indexes:

```
orientdb> EXPLAIN SELECT FROM Profile WHERE name = 'Luca'

Profiled command '{involvedIndexes:[1], indexReads:1, resultType:collection
resultSize:1, documentAnalyzedCompatibleClass:1, elapsed:1}'
in 0,002000 sec(s).
```

> For more information,s ee
>
> - SQL Commands

Understanding the Profile

When you run this command, it returns JSON data containing all of the following profile metrics:

Metric	Description
`elapsed`	Time to execute in seconds. The precision is the nanosecond.
`resultType`	The result-type: `collection` , `document` , or `number` .
`resultSize`	Number of records retrieved, in cases where the result-type is `collection` .
`recordReads`	Number of records read from disk.
`documentReads`	Number of documents read from disk. This metric may differ from `recordReads` in the event that other kinds of records are present in the command target. For instance, if you have documents and recordbytes in the same cluster it may skip many records. That said, in case of scans, it is recommended that you store different records in separate clusters.
`documentAnalyzedCompatibleClass`	Number of documents analyzed in the class. For instance, if you use the same cluster in documents for the classes `Account` and `Invoice` , it would skip records of the class `Invoice` when you target the class `Account` . In case of scans, it is recommended that you store different classes in separate clusters.
`involvedIndexes`	Indexes involved in the command.
`indexReads`	Number of records read from the index.

SQL - `FIND REFERENCES`

Searches records in the database that contain links to the given Record ID in the database or a subset of the specified class and cluster, returning the matching Record ID's.

Syntax

```
FIND REFERENCES <record-id>|(<sub-query>) [class-list]
```

- **`<record-id>`** Defines the Record ID you want to find links to in the database.
- **`<sub-query>`** Defines a sub-query for the Record ID's you want to find links to in the database. This feature was introduced in version 1.0rc9.
- **`<class-list>`** Defines a comma-separated list of classes or clusters that you want to search.

This command returns a document containing two fields:

Field	Description
`rid`	Record ID searched.
`referredBy`	Set of Record ID's referenced by the Record ID searched, if any. In the event that no records reference the searched Record ID, it returns an empty set.

Examples

- Find records that contain a link to `#5:0` :

```
orientdb> FIND REFERENCES 5:0

RESULT:
------+-----------------
 rid  | referredBy
------+-----------------
 #5:0 | [#10:23, #30:4]
------+-----------------
```

- Find references to the default cluster record

```
orientdb> FIND REFERENCES (SELECT FROM CLUSTER:default)
```

- Find all records in the classes `Profile` and `AnimalType` that contain a link to `#5:0` :

```
orientdb>  FIND REFERENCES 5:0 [Profile, AnimalType]
```

- Find all records in the cluster `profile` and class `AnimalType` that contain a link to `#5:0` :

```
orientdb> FIND REFERENCES 5:0 [CLUSTER:profile, AnimalType]
```

> For more information, see
>
> - SQL Commands

SQL - `GRANT`

Changes the permission of a role, granting it access to one or more resources. To remove access to a resource from the role, see the `REVOKE` command.

Syntax

```
GRANT <permission> ON <resource> TO <role>
```

- **`<permission>`** Defines the permission you want to grant to the role.
- **`<resource>`** Defines the resource on which you want to grant the permissions.
- **`<role>`** Defines the role you want to grant the permissions.

Examples

- Grant permission to update any record in the cluster `account` to the role `backoffice`:

  ```
  orientdb> GRANT UPDATE ON database.cluster.account TO backoffice
  ```

> For more information, see
>
> - `REVOKE
> - SQL Commands

Supported Permissions

Using this command, you can grant the following permissions to a role.

Permission	Description
`NONE`	Grants no permissions on the resource.
`CREATE`	Grants create permissions on the resource, such as the `CREATE CLASS` or `CREATE CLUSTER` commands.
`READ`	Grants read permissions on the resource, such as the `SELECT` query.
`UPDATE`	Grants update permissions on the resource, such as the `UPDATE` or `UPDATE EDGE` commands.
`DELETE`	Grants delete permissions on the resource, such as the `DROP INDEX` or `DROP SEQUENCE` commands.
`ALL`	Grants all permissions on the resource.

Supported Resources

Using this command, you can grant permissions on the following resources.

Resource	Description
`database`	Grants access on the current database.
`database.class.<class>`	Grants access on records contained in the indicated class. Use `**` to indicate all classes.
`database.cluster.<cluster>`	Grants access to records contained in the indicated cluster. Use `**` to indicate all clusters.
`database.query`	Grants the ability to execute a query, (`READ` is sufficient).
`database.command.<command>`	Grants the ability to execute the given command. Use `CREATE` for `INSERT` , `READ` for `SELECT` , `UPDATE` for `UPDATE` and `DELETE` for `DELETE` .
`database.config.<permission>`	Grants access to the configuration. Valid permissions are `READ` and `UPDATE` .
`database.hook.record`	Grants the ability to set hooks.
`server.admin`	Grants the ability to access server resources.

SQL - `HA REMOVE SERVER`

(Since v2.2) Removes a server from distributed configuration. It returns `true` if the server was found, otherwise `false`.

Syntax

```
HA REMOVE SERVER <server-name>
```

- **`<server-name>`** Defines the name of the server to remove.

Examples

- Removes the server `europe` from the distributed configuration:

  ```
  orientdb> HA REMOVE SERVER europe
  ```

Upgrading

Before v2.2, the list of servers running in HA configuration, was updated with the real situation. This could cause consistency problems in case of split brain network, because the two isolated network partitions could agree with a quorum based on the lower number of servers.

Example: if you have 5 servers with a `writeQuorum:"majority"`, means that if a node is unavailable (crash, network errors, etc.), the quorum is always on base 5, so 4 available nodes are ok. If you've lost also another node, you're still ok, because 3 is still the majority.

With OrientDB v2.2, if 3 nodes of 5 are out, you cannot reach the quorum, so all write operations are forbidden. Why? This is to keep the cluster consistent.

In facts, if you loose 3 servers of 5, it could happen a split brain network, so you have 2 networks with 2 servers and 3 servers. That's why in v2.2 we keep the servers in the distributed configuration, even if they are offline. Without such mechanism, both network would be able to write and as soon as both networks merge into one (the network problem is fixed), you could have tons of conflicts.

The correct way to remove a server from the configuration is running this command. In this way OrientDB HA will remove it from the list and the base for the quorum would not consider it anymore.

> For more information, see
>
> - Distributed Architecture
> - SQL Commands
> - Console Commands

SQL - `HA STATUS`

(Since v2.2) Retrieves information about HA.

Syntax

```
HA STATUS [-servers] [-db] [-latency] [-messages] [-all] [-output=text]
```

- `-servers` Dumps the configuration of servers
- `-db` Dumps the configuration of the database
- `-latency` Dumps the replication latency between servers (since v2.2.6)
- `-messages` Dumps the statistics about replication messages between servers (since v2.2.6)
- `-all` Dumps all the information
- `-output=text` Write the output as text formatted in readable tables

Examples

- Display the configuration of servers

```
orientdb> HA STATUS -servers -output=text

Executed '
+--------+------+-----------------------+-----+---------+----------------+----------
-----+----------------------+
|Name    |Status|Databases              |Conns|StartedOn|Binary          |HTTP
|UsedMemory            |
+--------+------+-----------------------+-----+---------+----------------+----------
-----+----------------------+
|europe1*|ONLINE|testdb01=ONLINE (MASTER)|0    |16:31:44
|192.168.1.5:2425|192.168.1.5:2481|183.06MB/3.56GB (5.03%)|
+--------+------+-----------------------+-----+---------+----------------+----------
-----+----------------------+
' in 0.002000 sec(s).
```

- Display the configuration of the current database

```
orientdb> HA STATUS -db -output=text
Executed '
LEGEND: X = Owner, o = Copy
+-----------+-----------+----------+-------+-------+
|           |           |          |MASTER |MASTER |
|           |           |          |ONLINE |ONLINE |
+-----------+-----------+----------+-------+-------+
|CLUSTER    |writeQuorum|readQuorum|europe1|europe0|
+-----------+-----------+----------+-------+-------+
|*          |     2     |    1     |   X   |       |
|data_1     |     2     |    1     |   o   |   X   |
|data_2     |     2     |    1     |   o   |   X   |
|data_3     |     2     |    1     |   o   |   X   |
|data_4     |     2     |    1     |   o   |   X   |
|e_4        |     2     |    1     |   o   |   X   |
|e_5        |     2     |    1     |   o   |   X   |
|e_6        |     2     |    1     |   o   |   X   |
|e_7        |     2     |    1     |   o   |   X   |
|internal   |     2     |    1     |       |       |
|ofunction_0|     2     |    1     |   o   |   X   |
|orole_0    |     2     |    1     |   o   |   X   |
|oschedule_0|     2     |    1     |   o   |   X   |
|osequence_0|     2     |    1     |   o   |   X   |
|ouser_0    |     2     |    1     |   o   |   X   |
|person     |     2     |    1     |   o   |   X   |
|person_1   |     2     |    1     |   o   |   X   |
|person_6   |     2     |    1     |   o   |   X   |
|person_7   |     2     |    1     |   o   |   X   |
|v_3        |     2     |    1     |   o   |   X   |
|v_5        |     2     |    1     |   o   |   X   |
|v_6        |     2     |    1     |   o   |   X   |
|v_7        |     2     |    1     |   o   |   X   |
+-----------+-----------+----------+-------+-------+
' in 0.050000 sec(s).
```

- Display the replication latency between servers (Since v2.2.6)

```
orientdb> HA STATUS -latency -output=text

Executed '
REPLICATION LATENCY AVERAGE (in milliseconds)
+-------+-----+------+-----+
|Servers|node1|node2*|node3|
+-------+-----+------+-----+
|node1  |     |  0.60| 0.43|
|node2* | 0.39|      | 0.38|
|node3  | 0.35|  0.53|     |
+-------+-----+------+-----+
' in 0.023000 sec(s).
```

- Display the statistics about replication messages between servers (Since v2.2.6)

```
orientdb> HA STATUS -messages -output=text

Executed '
REPLICATION MESSAGE COUNTERS
+-------+------+------+------+
|Servers| node1|node2*| node3|
+-------+------+------+------+
|node1  |      |     9|     5|
|node2* |64,476|      |53,593|
|node3  |     5|     5|      |
+-------+------+------+------+

REPLICATION MESSAGE COORDINATOR STATS
+-------+------+---------+------------+---------------+
|Servers|tx    |heartbeat|tx-completed|deploy_delta_db|
+-------+------+---------+------------+---------------+
|node1  |2     |28       |1           |               |
|node3* |56,379|20       |56,379      |2              |
+-------+------+---------+------------+---------------+
' in 0.005000 sec(s).
```

For more information, see

- Distributed Architecture
- SQL Commands
- Console Commands

SQL - `HA SYNC CLUSTER`

(Since v2.2) Asks for a re-synchronization of a cluster when running in HA. OrientDB will select the best server to provide the cluster.

Warning: starting from version 2.2.25, `HA SYNC CLUSTER <cluster-name>` will trigger an automatic rebuild of the indices defined on the Class the cluster `<cluster-name>` belongs to. Depending on the number of defined indices and amount of data included in the Class this operation can take some time. Furthermore it is important to wait that the sync cluster operation has finished before performing any database operations that involve that cluster.

Syntax

```
HA SYNC CLUSTER <cluster-name>
```

- **`<cluster-name>`** Defines the cluster name to re-synchronize.

Examples

- Re-synchronize the cluster `profile` :

  ```
  orientdb> HA SYNC CLUSTER profile
  ```

> For more information, see
>
> - `HA SYNC DATABASE`
> - Distributed Architecture
> - SQL Commands
> - Console Commands

SQL - `HA SYNC DATABASE`

(Since v2.2) Asks for a re-synchronization of the current database when running in HA. OrientDB will select the best server where to synchronize the database.

Syntax

```
HA SYNC DATABASE
```

Examples

- Re-synchronize the database:

  ```
  orientdb> HA SYNC DATABASE
  ```

> For more information, see
>
> - `HA SYNC CLUSTER`
> - Distributed Architecture
> - SQL Commands
> - Console Commands

SQL - `HA SET`

(Since v2.2.22, Enterprise Edition only). Update the server configuration in High Availability setting.

Syntax

```
HA SET [DBSTATUS <server>=<status>] [ROLE <server>=MASTER|REPLICA] [OWNER <cluster>=<server>]
```

NOTE: *the key/value pairs must not contain any space. This is valid* `HA SET ROLE europe=REPLICA` *, this is not:* `HA SET ROLE europe = REPLICA`

- **DBSTATUS** Changes the status of the database. This operation must be executed only if recommended by OrientDB Support Team
  - **`<server>`** Server name.
  - **`<status>`** The new status to set between `[NOT_AVAILABLE, OFFLINE, SYNCHRONIZING, ONLINE, BACKUP]` .
- **ROLE** Changes the role of the server between `MASTER` and `REPLICA`
  - **`<server>`** Server name.
- **OWNER** Changes cluster's owner
  - **`<cluster>`** The name of the cluster to change.
  - **`<server>`** Name of the server to become the owner of the cluster. The server name must be already present in the server list for that cluster.

Examples

- Change the role of the server `europe` to be a `REPLICA` only:

  ```
  orientdb> HA SET ROLE europe=REPLICA
  ```

- Set the server `usa0` as the owner of cluster "customer":

  ```
  orientdb> HA SET OWNER customer=usa0
  ```

- Set the status of database `crm` to OFFLINE for server `china` :

  ```
  orientdb> HA SET DBSTATUS china=OFFLINE
  ```

> For more information, see
>
> - Distributed Architecture
> - SQL Commands
> - Console Commands

SQL - `INSERT`

The `INSERT` command creates a new record in the database. Records can be schema-less or follow rules specified in your model.

Syntax:

```
INSERT INTO [CLASS:]<class>|CLUSTER:<cluster>|INDEX:<index>
  [(<field>[,]*) VALUES (<expression>[,]*)[,]*]|
  [SET <field> = <expression>|<sub-command>[,]*]|
  [CONTENT {<JSON>}]
  [RETURN <expression>]
  [FROM <query>]
```

- **`CONTENT`** Defines JSON data as an option to set field values.
- **`RETURN`** Defines an expression to return instead of the number of inserted records. Valid expressions are:
  - `@rid` Returns the Record ID of the new record.
  - `@this` Returns the entire new record.
- **`FROM`** Defines where you want to insert the result-set. Introduced in version 1.7.

Examples:

- Inserts a new record with the name `Jay` and surname `Miner` .

  As an example, in the SQL-92 standard, such as with a Relational database, you might use:

  ```
  orientdb> INSERT INTO Profile (name, surname)
            VALUES ('Jay', 'Miner')
  ```

  Alternatively, in the OrientDB abbreviated syntax, the query would be written as,

  ```
  orientdb> INSERT INTO Profile SET name = 'Jay', surname = 'Miner'
  ```

  In JSON content syntax, it would be written as this,

  ```
  orientdb> INSERT INTO Profile CONTENT {"name": "Jay", "surname": "Miner"}
  ```

- Insert a new record of the class `Profile` , but in a different cluster from the default.

  In SQL-92 syntax:

  ```
  orientdb> INSERT INTO Profile CLUSTER profile_recent (name, surname) VALUES
            ('Jay', 'Miner')
  ```

  Alternative, in the OrientDB abbreviated syntax:

  ```
  orientdb> INSERT INTO Profile CLUSTER profile_recent SET name = 'Jay',
            surname = 'Miner'
  ```

- Insert several records at the same time:

  ```
  orientdb> INSERT INTO Profile(name, surname) VALUES ('Jay', 'Miner'),
            ('Frank', 'Hermier'), ('Emily', 'Sout')
  ```

- Insert a new record, adding a relationship.

  In SQL-93 syntax:

```
orientdb> INSERT INTO Employee (name, boss) VALUES ('jack', #11:09)
```

In the OrientDB abbreviated syntax:

```
orientdb> INSERT INTO Employee SET name = 'jack', boss = #11:99
```

- Insert a new record, add a collection of relationships.

  In SQL-93 syntax:

```
orientdb> INSERT INTO Profile (name, friends) VALUES ('Luca', [#10:3, #10:4])
```

  In the OrientDB abbreviated syntax:

```
orientdb> INSERT INTO Profiles SET name = 'Luca', friends = [#10:3, #10:4]
```

- Inserts using `SELECT` sub-queries

```
orientdb> INSERT INTO Diver SET name = 'Luca', buddy = (SELECT FROM Diver
          WHERE name = 'Marko')
```

- Inserts using `INSERT` sub-queries:

```
orientdb> INSERT INTO Diver SET name = 'Luca', buddy = (INSERT INTO Diver
          SET name = 'Marko')
```

- Inserting into a different cluster:

```
orientdb> INSERT INTO CLUSTER:asiaemployee (name) VALUES ('Matthew')
```

  However, note that the document has no assigned class. To create a document of a certain class, but in a different cluster than the default, instead use:

```
orientdb> INSERT INTO CLUSTER:asiaemployee (@class, content) VALUES
          ('Employee', 'Matthew')
```

  That inserts the document of the class `Employee` into the cluster `asiaemployee`.

- Insert a new record, adding it as an embedded document:

```
orientdb> INSERT INTO Profile (name, address) VALUES ('Luca', { "@type": "d",
          "street": "Melrose Avenue", "@version": 0 })
```

- Insert from a query.

  To copy records from another class, use:

```
orientdb> INSERT INTO GermanyClient FROM SELECT FROM Client WHERE
          country = 'Germany'
```

  This inserts all the records from the class `Client` where the country is Germany, in the class `GermanyClient`.

  To copy records from one class into another, while adding a field:

```
orientdb> INSERT INTO GermanyClient FROM SELECT *, true AS copied FROM Client
          WHERE country = 'Germany'
```

This inserts all records from the class `Client` where the country is Germany into the class `GermanClient`, with the addition field `copied` to the value `true`.

For more information on SQL, see SQL commands.

SQL - `LIVE SELECT`

Enables a Live Query, returning a unique identifier token. Through this token, your application can receive updates whenever `INSERT`, `DELETE`, or `UPDATE` commands are issued against the given records. This feature was introduced in version 2.1 of OrientDB.

> **NOTE**: Currently, Live Queries are only supported in Java through the Java API, and in Node.js through the OrientJS Driver. It is not currently supported through the Console. For more general information, see Live Queries.

Syntax

```
LIVE SELECT FROM <target>
```

- **`FROM <target>`** Designates the object to register for live queries. THis can be a class, cluster, single Record ID, set of Record ID's, or index values sorted by ascending or descending key order.

SQL - `LIVE UNSUBSCRIBE`

Disables a Live Query token so that it no longer receives updates from OrientDB. To enable Live Queries, use `LIVE SELECT`. This feature was introduced in version 2.1 of OrientDB.

> **NOTE**: Currently, Live Queries are only supported in Java through the Java API and in Node.js through the OrientJS Driver. The commands are not available through the Console. For more general information, see Live Queries.

Syntax

```
LIVE UNSUBSCRIBE <token>
```

- **`<token>`** Unique identifier for the Live Query you want to disable.

SQL - `MATCH`

Queries the database in a declarative manner, using pattern matching. This feature was introduced in version 2.2.

Simplified Syntax

```
MATCH
  {
    [class: <class>],
    [as: <alias>],
    [where: (<whereCondition>)]
  }
  .<functionName>(){
    [class: <className>],
    [as: <alias>],
    [where: (<whereCondition>)],
    [while: (<whileCondition>)],
    [maxDepth: <number>],
    [optional: (true | false)]
  }*
RETURN <expression> [ AS <alias> ] [, <expression> [ AS <alias> ]]*
LIMIT <number>
```

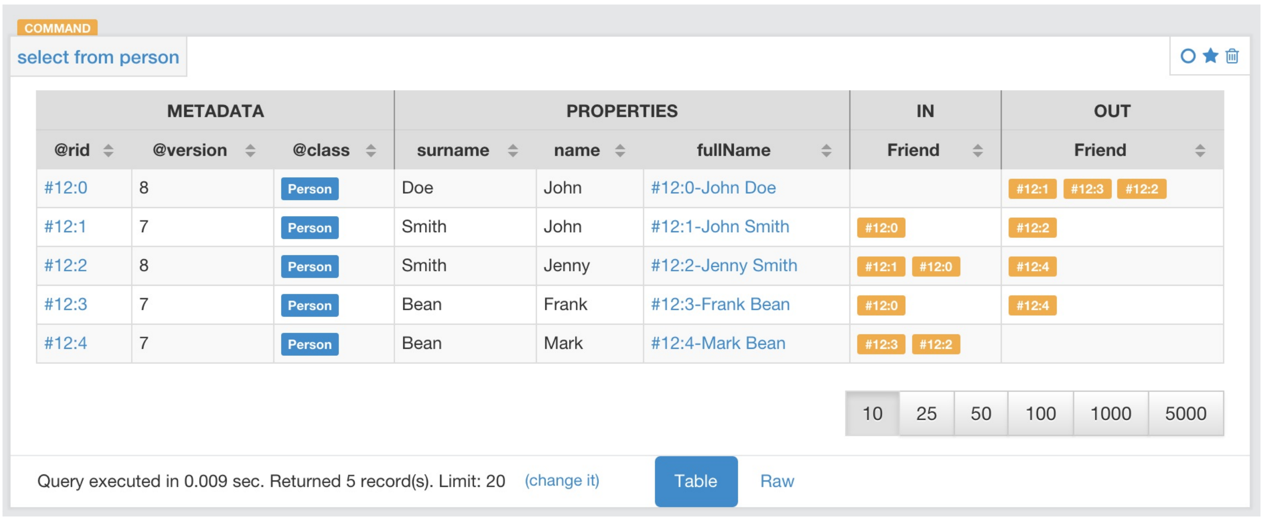
- **`<class>`** Defines a valid target class.
- **`as: <alias>`** Defines an alias for a node in the pattern.
- **`<whereCondition>`** Defines a filter condition to match a node in the pattern. It supports the normal SQL `WHERE` clause. You can also use the `$currentMatch` and `$matched` context variables.
- **`<functionName>`** Defines a graph function to represent the connection between two nodes. For instance, `out()` , `in()` , `outE()` , `inE()` , etc. For out(), in(), both() also a shortened *arrow* syntax is supported:
  - `{...}.out(){...}` can be written as `{...}-->{...}`
  - `{...}.out("EdgeClass"){...}` can be written as `{...}-EdgeClass->{...}`
  - `{...}.in(){...}` can be written as `{...}<--{...}`
  - `{...}.in("EdgeClass"){...}` can be written as `{...}<-EdgeClass-{...}`
  - `{...}.both(){...}` can be written as `{...}--{...}`
  - `{...}.both("EdgeClass"){...}` can be written as `{...}-EdgeClass-{...}`
- **`<whileCondition>`** Defines a condition that the statement must meet to allow the traversal of this path. It supports the normal SQL `WHERE` clause. You can also use the `$currentMatch` , `$matched` and `$depth` context variables. For more information, see Deep Traversal, below.
- **`<maxDepth>`** Defines the maximum depth for this single path.
- **`RETURN <expression> [ AS <alias> ]`** Defines elements in the pattern that you want returned. It can use one of the following:
  - Aliases defined in the `as:` block.
  - `$matches` Indicating all defined aliases.
  - `$paths` Indicating the full traversed paths.
  - `$elements` (since 2.2.1) Indicating that all the elements that would be returned by the $matches have to be returned flattened, without duplicates.
  - `$pathElements` (since 2.2.1) Indicating that all the elements that would be returned by the $paths have to be returned flattened, without duplicates.
- **`optional`** (since 2.2.4) if set to true, allows to evaluate and return a pattern even if that particular node does not match the pattern itself (ie. there is no value for that node in the pattern). In current version, optional nodes are allowed only on right terminal nodes, eg. `{} --> {optional:true}` is allowed, `{optional:true} <-- {}` is not.

BNF Syntax

```
MatchStatement     := ( <MATCH> MatchExpression ( <COMMA> MatchExpression )* <RETURN> Expression (<AS> Identifier)? ( <COMMA>
Expression (<AS> Identifier)? )* ( Limit )? )

MatchExpression       := ( MatchFilter ( ( MatchPathItem | MultiMatchPathItem ) )* )

MatchPathItem       := ( MethodCall ( MatchFilter )? )

MatchPathItemFirst := ( FunctionCall ( MatchFilter )? )

MultiMatchPathItem := ( <DOT> <LPAREN> MatchPathItemFirst ( MatchPathItem )* <RPAREN> ( MatchFilter )? )

MatchFilter        := ( <LBRACE> ( MatchFilterItem ( <COMMA> MatchFilterItem )* )? <RBRACE> )

MatchFilterItem    := ( ( <CLASS> <COLON> Expression )  | ( <AS> <COLON> Identifier ) | ( <WHERE> <COLON> <LPAREN> ( WhereClau
se ) <RPAREN> ) | ( <WHILE> <COLON> <LPAREN> ( WhereClause ) <RPAREN> ) | ( <MAXDEPTH> <COLON> Integer ) )
```

Examples

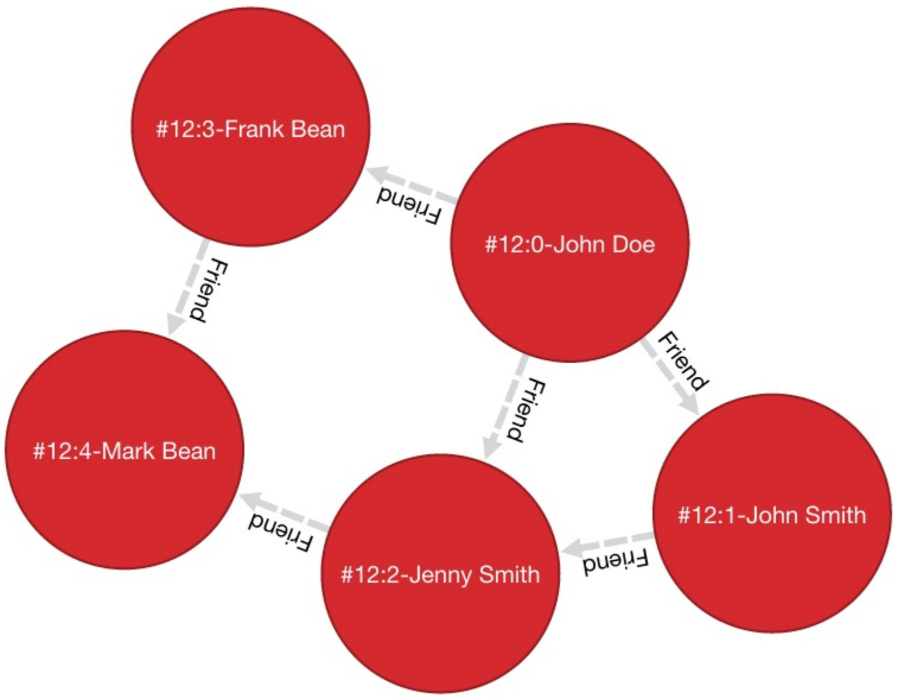
The following examples are based on this sample data-set from the class `People` :


COMMAND
select from person

METADATA			PROPERTIES			IN	OUT
@rid	@version	@class	surname	name	fullName	Friend	Friend
#12:0	8	Person	Doe	John	#12:0-John Doe		#12:1 #12:3 #12:2
#12:1	7	Person	Smith	John	#12:1-John Smith	#12:0	#12:2
#12:2	8	Person	Smith	Jenny	#12:2-Jenny Smith	#12:1 #12:0	#12:4
#12:3	7	Person	Bean	Frank	#12:3-Frank Bean	#12:0	#12:4
#12:4	7	Person	Bean	Mark	#12:4-Mark Bean	#12:3 #12:2	

10 25 50 100 1000 5000

Query executed in 0.009 sec. Returned 5 record(s). Limit: 20 (change it) Table Raw



- Find all people with the name John:

```
orientdb> MATCH {class: Person, as: people, where: (name = 'John')}
          RETURN people

---------
  people
---------
  #12:0
  #12:1
---------
```

- Find all people with the name John and the surname Smith:

```
orientdb> MATCH {class: Person, as: people, where: (name = 'John' AND
          surname = 'Smith')} RETURN people

-------
people
-------
 #12:1
-------
```

- Find people named John with their friends:

```
orientdb> MATCH {class: Person, as: person, where:
          (name = 'John')}.both('Friend') {as: friend}
          RETURN person, friend

--------+---------
 person | friend
--------+---------
 #12:0  | #12:1
 #12:0  | #12:2
 #12:0  | #12:3
 #12:1  | #12:0
 #12:1  | #12:2
--------+---------
```

- Find friends of friends:

```
orientdb> MATCH {class: Person, as: person, where: (name = 'John' AND
          surname = 'Doe')}.both('Friend').both('Friend')
          {as: friendOfFriend} RETURN person, friendOfFriend

--------+----------------
 person | friendOfFriend
--------+----------------
 #12:0  | #12:0
 #12:0  | #12:1
 #12:0  | #12:2
 #12:0  | #12:3
 #12:0  | #12:4
--------+----------------
```

- Find people, excluding the current user:

```
orientdb> MATCH {class: Person, as: person, where: (name = 'John' AND
          surname = 'Doe')}.both('Friend').both('Friend'){as: friendOfFriend,
          where: ($matched.person != $currentMatch)}
          RETURN person, friendOfFriend

--------+----------------
 person | friendOfFriend
--------+----------------
 #12:0  | #12:1
 #12:0  | #12:2
 #12:0  | #12:3
 #12:0  | #12:4
--------+----------------
```

- Find friends of friends to the sixth degree of separation:

```
orientdb> MATCH {class: Person, as: person, where: (name = 'John' AND
          surname = 'Doe')}.both('Friend'){as: friend,
          where: ($matched.person != $currentMatch) while: ($depth < 6)}
          RETURN person, friend

--------+---------
 person | friend
--------+---------
 #12:0  | #12:0
 #12:0  | #12:1
 #12:0  | #12:2
 #12:0  | #12:3
 #12:0  | #12:4
--------+---------
```

- Finding friends of friends to six degrees of separation, since a particular date:

```
orientdb> MATCH {class: Person, as: person,
          where: (name = 'John')}.(bothE('Friend'){
          where: (date < ?)}.bothV()){as: friend,
          while: ($depth < 6)} RETURN person, friend
```

In this case, the condition `$depth < 6` refers to traversing the block `bothE('Friend')` six times.

- Find friends of my friends who are also my friends, using multiple paths:

```
orientdb> MATCH {class: Person, as: person, where: (name = 'John' AND
          surname = 'Doe')}.both('Friend').both('Friend'){as: friend},
          { as: person }.both('Friend'){ as: friend }
          RETURN person, friend

--------+--------
 person | friend
--------+--------
 #12:0  | #12:1
 #12:0  | #12:2
--------+--------
```

In this case, the statement matches two expression: the first to friends of friends, the second to direct friends. Each expression shares the common aliases (`person` and `friend`). To match the whole statement, the result must match both expressions, where the alias values for the first expression are the same as that of the second.

- Find common friends of John and Jenny:

```
orientdb> MATCH {class: Person, where: (name = 'John' AND
         surname = 'Doe')}.both('Friend'){as: friend}.both('Friend')
         {class: Person, where: (name = 'Jenny')} RETURN friend


--------
 friend
--------
 #12:1
--------
```

The same, with two match expressions:

```
orientdb> MATCH {class: Person, where: (name = 'John' AND
         surname = 'Doe')}.both('Friend'){as: friend},
         {class: Person, where: (name = 'Jenny')}.both('Friend')
         {as: friend} RETURN friend
```

Context Variables

When running these queries, you can use any of the following context variables:

Variable	Description
`$matched`	Gives the current matched record. You must explicitly define the attributes for this record in order to access them. You can use this in the `where:` and `while:` conditions to refer to current partial matches or as part of the `RETURN` value.
`$currentMatch`	Gives the current complete node during the match.
`$depth`	Gives the traversal depth, following a single path item where a `while:` condition is defined.

Use Cases

Expanding Attributes

You can run this statement as a sub-query inside of another statement. Doing this allows you to obtain details and aggregate data from the inner `SELECT` query.

```
orientdb> SELECT person.name AS name, person.surname AS surname,
          friend.name AS friendName, friend.surname AS friendSurname
          FROM (MATCH {class: Person, as: person,
          where: (name = 'John')}.both('Friend'){as: friend}
          RETURN person, friend)

--------+----------+------------+---------------
 name   | surname  | friendName | friendSurname
--------+----------+------------+---------------
 John   | Doe      | John       | Smith
 John   | Doe      | Jenny      | Smith
 John   | Doe      | Frank      | Bean
 John   | Smith    | John       | Doe
 John   | Smith    | Jenny      | Smith
--------+----------+------------+---------------
```

As an alternative, you can use the following:

```
orientdb> MATCH {class: Person, as: person,
          where: (name = 'John')}.both('Friend'){as: friend}
          RETURN
          person.name as name, person.surname as surname,
          friend.name as firendName, friend.surname as friendSurname

--------+----------+------------+---------------
 name   | surname  | friendName | friendSurname
--------+----------+------------+---------------
 John   | Doe      | John       | Smith
 John   | Doe      | Jenny      | Smith
 John   | Doe      | Frank      | Bean
 John   | Smith    | John       | Doe
 John   | Smith    | Jenny      | Smith
--------+----------+------------+---------------
```

Incomplete Hierarchy

Consider building a database for a company that shows a hierarchy of departments within the company. For instance,

```
            [manager] department
           (employees in department)


                  [m0]0
                   (e1)
                   /   \
                  /     \
                 /       \
            [m1]1        [m2]2
           (e2, e3)     (e4, e5)
              / \         / \
             3   4       5   6
           (e6) (e7)   (e8)  (e9)
           /  \
       [m3]7    8
       (e10)   (e11)
        /
       9
   (e12, e13)
```

This loosely shows that,

- Department `0` is the company itself, manager 0 (`m0`) is the CEO

- `e10` works at department `7`, his manager is `m3`
- `e12` works at department `9`, this department has no direct manager, so `e12`'s manager is `m3` (the upper manager)

In this case, you would use the following query to find out who's the manager to a particular employee:

```
orientdb> SELECT EXPAND(manager) FROM (MATCH {class:Employee,
          where: (name = ?)}.out('WorksAt').out('ParentDepartment')
          {while: (out('Manager').size() == 0),
          where: (out('Manager').size() > 0)}.out('Manager')
          {as: manager} RETURN manager)
```

Deep Traversal

Match path items act in a different manners, depending on whether or not you use `while:` conditions in the statement.

For instance, consider the following graph:

```
[name='a'] -FriendOf-> [name='b'] -FriendOf-> [name='c']
```

Running the following statement on this graph only returns `b`:

```
orientdb> MATCH {class: Person, where: (name = 'a')}.out("FriendOf")
          {as: friend} RETURN friend

--------
 friend
--------
 b
--------
```

What this means is that it traverses the path item `out("FriendOf")` exactly once. It only returns the result of that traversal.

If you add a `while` condition:

```
orientdb> MATCH {class: Person, where: (name = 'a')}.out("FriendOf")
          {as: friend, while: ($depth < 2)} RETURN friend

---------
 friend
---------
 a
 b
---------
```

Including a `while:` condition on the match path item causes OrientDB to evaluate this item as zero to *n* times. That means that it returns the starting node, (`a`, in this case), as the result of zero traversal.

To exclude the starting point, you need to add a `where:` condition, such as:

```
orientdb> MATCH {class: Person, where: (name = 'a')}.out("FriendOf")
          {as: friend, while: ($depth < 2) where: ($depth > 0)}
          RETURN friend
```

As a general rule,

- **`while` Conditions:** Define this if it must execute the next traversal, (it evaluates at level zero, on the origin node).
- **`where` Condition:** Define this if the current element, (the origin node at the zero iteration the right node on the iteration is greater than zero), must be returned as a result of the traversal.

For instance, suppose that you have a genealogical tree. In the tree, you want to show a person, grandparent and the grandparent of that grandparent, and so on. The result: saying that the person is at level zero, parents at level one, grandparents at level two, etc., you would see all ancestors on even levels. That is, `level % 2 == 0`.

To get this, you might use the following query:

```
orientdb> MATCH {class: Person, where: (name = 'a')}.out("Parent")
          {as: ancestor, while: (true) where: ($depth % 2 = 0)}
          RETURN ancestor
```

Best practices

Queries can involve multiple operations, based on the domain model and use case. In some cases, like projection and aggregation, you can easily manage them with a `SELECT` query. With others, such as pattern matching and deep traversal, `MATCH` statements are more appropriate.

Use `SELECT` and `MATCH` statements together (that is, through sub-queries), to give each statement the correct responsibilities. Here,

Filtering Record Attributes for a Single Class

Filtering based on record attributes for a single class is a trivial operation through both statements. That is, finding all people named John can be written as:

```
orientdb> SELECT FROM Person WHERE name = 'John'
```

You can also write it as,

```
orientdb> MATCH {class: Person, as: person, where: (name = 'John')}
          RETURN person
```

The efficiency remains the same. Both queries use an index. With `SELECT`, you obtain expanded records, while with `MATCH`, you only obtain the Record ID's.

Filtering on Record Attributes of Connected Elements

Filtering based on the record attributes of connected elements, such as neighboring vertices, can grow trick when using `SELECT`, while with `MATCH` it is simple.

For instance, find all people living in Rome that have a friend called John. There are three different ways you can write this, using `SELECT`:

```
orientdb> SELECT FROM Person WHERE BOTH('Friend').name CONTAINS 'John'
          AND out('LivesIn').name CONTAINS 'Rome'

orientdb> SELECT FROM (SELECT BOTH('Friend') FROM Person WHERE name
          'John') WHERE out('LivesIn').name CONTAINS 'Rome'

orientdb> SELECT FROM (SELECT in('LivesIn') FROM City WHERE name = 'Rome')
          WHERE BOTH('Friend').name CONTAINS 'John'
```

In the first version, the query is more readable, but it does not use indexes, so it is less optimal in terms of execution time. The second and third use indexes if they exist, (on `Person.name` or `City.name`, both in the sub-query), but they're harder to read. Which index they use depends only on the way you write the query. That is, if you only have an index on `City.name` and not `Person.name`, the second version doesn't use an index.

Using a `MATCH` statement, the query becomes:

```
orientdb> MATCH {class: Person, where: (name = 'John')}.both("Friend")
          {as: result}.out('LivesIn'){class: City, where: (name = 'Rome')}
          RETURN result
```

Here, the query executor optimizes the query for you, choosing indexes where they exist. Moreover, the query becomes more readable, especially in complex cases, such as multiple nested `SELECT` queries.

`TRAVERSE` Alternative

There are similar limitations to using `TRAVERSE`. You may benefit from using `MATCH` as an alternative.

For instance, consider a simple `TRAVERSE` statement, like:

```
orientdb> TRAVERSE out('Friend') FROM (SELECT FROM Person WHERE name = 'John')
          WHILE $depth < 3
```

Using a `MATCH` statement, you can write the same query as:

```
orientdb> MATCH {class: Person, where: (name = 'John')}.both("Friend")
          {as: friend, while: ($depth < 3)} RETURN friend
```

Consider a case where you have a `since` date property on the edge `Friend`. You want to traverse the relationship only for edges where the `since` value is greater than a given date. In a `TRAVERSE` statement, you might write the query as:

```
orientdb> TRAVERSE bothE('Friend')[since > date('2012-07-02', 'yyyy-MM-dd')].bothV()
          FROM (SELECT FROM Person WHERE name = 'John') WHILE $depth < 3
```

Unforunately, this statement DOESN"T WORK in the current release. However, you can get the results you want using a `MATCH` statement:

```
orientdb> MATCH {class: Person, where: (name = 'John')}.(bothE("Friend")
          {where: (since > date('2012-07-02', 'yyyy-MM-dd'))}.bothV())
          {as: friend, while: ($depth < 3)} RETURN friend
```

Projections and Grouping Operations

Projections and grouping operations are better expressed with a `SELECT` query. If you need to filter and do projection or aggregation in the same query, you can use `SELECT` and `MATCH` in the same statement.

This is particular important when you expect a result that contains attributes from different connected records (cartesian product). For instance, to retrieve names, their friends and the date since they became friends:

```
orientdb> SELECT person.name AS name, friendship.since AS since, friend.name
          AS friend FROM (MATCH {class: Person, as: person}.bothE('Friend')
          {as: friendship}.bothV(){as: friend,
          where: ($matched.person != $currentMatch)}
          RETURN person, friendship, friend)
```

The same can be also achieved with the MATCH only:

```
orientdb> MATCH {class: Person, as: person}.bothE('Friend')
          {as: friendship}.bothV(){as: friend,
          where: ($matched.person != $currentMatch)}
          RETURN person.name as name, friendship.since as since, friend.name as friend
```

RETURN expressions

In the RETURN section you can use:

multiple expressions, with or without an alias (if no alias is defined, OrientDB will generate a default alias for you), comma separated

```
MATCH
  {class: Person, as: person}
  .bothE('Friend'){as: friendship}
  .bothV(){as: friend, where: ($matched.person != $currentMatch)}
RETURN person, friendship, friend

result:

| person | friendship | friend |
--------------------------------
| #12:0  | #13:0      | #12:2  |
| #12:0  | #13:1      | #12:3  |
| #12:1  | #13:2      | #12:3  |
```

```
MATCH
  {class: Person, as: person}
  .bothE('Friend'){as: friendship}
  .bothV(){as: friend, where: ($matched.person != $currentMatch)}
RETURN person.name as name, friendship.since as since, friend.name as friend

result:

| name | since | friend |
-------------------------
| John | 2015  | Frank  |
| John | 2015  | Jenny  |
| Joe  | 2016  | Jenny  |
```

```
MATCH
  {class: Person, as: person}
  .bothE('Friend'){as: friendship}
  .bothV(){as: friend, where: ($matched.person != $currentMatch)}
RETURN person.name + " is a friend of " + friend.name as friends

result:

| friends                    |
------------------------------
| John is a friend of Frank  |
| John is a friend of Jenny  |
| Joe is a friend of Jenny   |
```

$matches, to return all the patterns that match current statement. Each row in the result set will be a single pattern, containing only nodes in the statement that have an `as:` defined

```
MATCH
  {class: Person, as: person}
  .bothE('Friend'){}                                                // no 'as:friendship' in this case
  .bothV(){as: friend, where: ($matched.person != $currentMatch)}
RETURN $matches

result:

| person |  friend |
--------------------
| #12:0  |  #12:2  |
| #12:0  |  #12:3  |
| #12:1  |  #12:3  |
```

$paths, to return all the patterns that match current statement. Each row in the result set will be a single pattern, containing all th nodes in the statement. For nodes that have an `as:`, the alias will be returned, for the others a default alias is generated (automatically generated aliases start with `$ORIENT_DEFAULT_ALIAS_`)

```
MATCH
  {class: Person, as: person}
  .bothE('Friend'){}                                              // no 'as:friendship' in this case
  .bothV(){as: friend, where: ($matched.person != $currentMatch)}
RETURN $paths

result:

| person | friend | $ORIENT_DEFAULT_ALIAS_0 |
---------------------------------------------
| #12:0  | #12:2  | #13:0                   |
| #12:0  | #12:3  | #13:1                   |
| #12:1  | #12:3  | #13:2                   |
```

$elements (since 2.2.1), the same as `$matches`, but for each node present in the pattern, a single row is created in the result set (no duplicates)

```
MATCH
  {class: Person, as: person}
  .bothE('Friend'){}                                              // no 'as:friendship' in this case
  .bothV(){as: friend, where: ($matched.person != $currentMatch)}
RETURN $elements

result:

| @rid   |  @class | name   |  .....   |
----------------------------------------
| #12:0  |  Person | John   |  .....   |
| #12:1  |  Person | Joe    |  .....   |
| #12:2  |  Person | Frank  |  .....   |
| #12:3  |  Person | Jenny  |  .....   |
```

$pathElements (since 2.2.1), the same as `$paths`, but for each node present in the pattern, a single row is created in the result set (no duplicates)

```
MATCH
  {class: Person, as: person}
  .bothE('Friend'){}                                              // no 'as:friendship' in this case
  .bothV(){as: friend, where: ($matched.person != $currentMatch)}
RETURN $pathElements

result:

| @rid   |  @class | name   | since  |  .....   |
-------------------------------------------------
| #12:0  |  Person | John   |        |  .....   |
| #12:1  |  Person | Joe    |        |  .....   |
| #12:2  |  Person | Frank  |        |  .....   |
| #12:3  |  Person | Jenny  |        |  .....   |
| #13:0  |  Friend |        |  2015  |  .....   |
| #13:1  |  Friend |        |  2015  |  .....   |
| #13:2  |  Friend |        |  2016  |  .....   |
```

IMPORTANT: When using MATCH statemet in OrientDB Studio Graph panel you have to use $elements or $pathElements as return type, to let the Graph panel render the matched patterns correctly

Arrow notation

`out()` , `in()` and `both()` operators can be replaced with arrow notation `-->` , `<--` and `--`

Eg. the query

```

MATCH {class: V, as: a}.out(){}.out(){}.out(){as:b}
RETURN a, b

```

can be written as

```
MATCH {class: V, as: a} --> {} --> {} --> {as:b}
RETURN a, b
```

Eg. the query (things that belong to friends)

```
MATCH {class: Person, as: a}.out('Friend'){as:friend}.in('BelongsTo'){as:b}
RETURN a, b
```

can be written as

```
MATCH {class: Person, as: a}  -Friend-> {as:friend}
```

Using arrow notation the curly braces are mandatory on both sides. eg:

```
MATCH {class: Person, as: a} --> {} --> {as:b} RETURN a, b  //is allowed

MATCH {class: Person, as: a} --> --> {as:b} RETURN a, b  //is NOT allowed

MATCH {class: Person, as: a}.out().out(){as:b} RETURN a, b  //is allowed

MATCH {class: Person, as: a}.out(){}.out(){as:b} RETURN a, b  //is allowed
```

SQL - `MOVE VERTEX`

Moves one or more vertices into a different class or cluster.

Following the move, the vertices use a different Record ID. The command updates all edges to use the moved vertices. When using a distributed database, if you specify a cluster, it moves the vertices to the server owner of the target cluster.

Syntax

```
MOVE VERTEX <source> TO <destination> [SET [<field>=<value>]* [,]] [MERGE <JSON>]
[BATCH <batch-size>]
```

- **`<source>`** Defines the vertex you want to move. It supports the following values,
  - *Vertex* Using the Record ID of a single vertex.
  - *Array* Using an array of record ID's for vertices you want to move.
- **`<destination>`** Defines where you want to move the vertex to. It supports the following values,
  - *Class* Using `CLASS:<class>` with the class you want to move the vertex into.
  - *Cluster* Using `CLUSTER:<cluster>` with the cluster you want to move the vertex into.
- **`SET`** Clause to set values on fields during the transition.
- **`MERGE`** Clause to set values on fields during the transition, through JSON.
- **`BATCH`** Defines the batch size, allowing you to execute the command in smaller blocks to avoid memory problems when moving a large number of vertices.

WARNING: This command updates all connected edges, but not the links. When using the Graph API, it is recommend that you always use edges connected to vertices and never links.

Examples

- Move a single vertex from its current position to the class `Provider`:

  ```
  orientdb> MOVE VERTEX #34:232 TO CLASS:Provider
  ```

- Move an array of vertices by their record ID's to the class `Provider`:

  ```
  orientdb> MOVE VERTEX [#34:232,#34:444] TO CLASS:Provider
  ```

- Move a set of vertices to the class `Provider`, defining those you want to move with a subquery:

  ```
  orientdb> MOVE VERTEX (SELECT FROM V WHERE city = 'Rome') TO CLASS:Provider
  ```

- Move a vertex from its current position to the European cluster

  ```
  orientdb> MOVE VERTEX #3:33 TO CLUSTER:providers_europe
  ```

  You may find this useful when using a distributed database, where you can move vertices onto different servers.

- Move a set of vertices to the class `Provider`, while doing so update the property `movedOn` to the current date:

  ```
  orientdb> MOVE VERTEX (SELECT FROM V WHERE type = 'provider') TO CLASS:Provider
            SET movedOn = Date()
  ```

  Note the similarities this syntax has with the `UPDATE` command.

- Move the vertex using a subquery, using JSON update the properties during the transition:

```
orientdb> MOVE VERTEX (SELECT FROM User) TO CLUSTER:users_europe BATCH 50
```

- Move the same vertices as above using only one transaction:

```
orientdb> MOVE VERTEX (SELECT FROM User) TO CLUSTER:users_europe BATCH -1
```

> For more information, see
>
> - `CREATE VERTEX`
> - `CREATE EDGE`
> - SQL Commands

Use Cases

Refactoring Graphs through Sub-types

It's a very common situation where you begin modeling your domain one way, but find later that you need more flexibility.

For instance, say that you start out with a vertex class called `Person`. After using the database for several months, populating it with new vertices, you decide that you need to split these vertices into two new classes, or sub-types, called `Customer` and `Provider`, (rendering `Person` into an abstract class).

- Create the new classes for your sub-types:

```
orientdb> CREATE CLASS Customer EXTENDS Person
orientdb> CREATE CLASS Provider EXTENDS Person
```

- Move the providers and customers from `Person` into their respective sub-types:

```
orientdb> MOVE VERTEX (SELECT FROM Person WHERE type = 'Customer') TO
          CLASS:Customer
orientdb> MOVE VERTEX (SELECT FROM Person WHERE type = 'Provider') TO
          CLASS:Provider
```

- Make the class `Person` an abstract class:

```
orientdb> ALTER CLASS Person ABSTRACT TRUE
```

Moving Vertices onto Different Servers

With OrientDB, you can scale your infrastructure up by adding new servers. When you add a new server, OrientDB automatically creates a new cluster with the name of the class plus the node name. For instance, `customer_europe`.

The best practice when you need to scale up is partitioning, especially on writes. If you have a graph with `Customer` vertices and you want to move some of these onto a different server, you can move them to the cluster owned by the server.

For instance, move all customers that live in Italy, Germany or the United Kingdom onto the cluster `customer_europe`, which is assigned to the node `Europe`. This means that access to European customers is faster with applications connected to the European node.

```
orientdb> MOVE VERTEX (SELECT FROM Customer WHERE ['Italy', 'Germany', 'UK'] IN
          out('city').out('country') ) TO CLUSTER:customer_europe
```

History

2.0

- Initial implementation of the feature.

SQL - `OPTIMIZE DATABASE`

Optimizes the database for particular operations.

Syntax

```
OPTIMIZE DATABASE [-lwedges] [-noverbose]
```

- **`-lwedges`** Converts regular edges into Lightweight Edges.
- **`-noverbose`** Disables output.

> Currently, this command only supports optimization for Lightweight Edges. Additional optimization options are planned for future releases of OrientDB.

Examples

- Convert regular edges into Lightweight Edges:

  ```
  orientdb> OPTIMIZE DATABASE -lwedges
  ```

> For more information, see
>
> - Lightweight Edges
> - SQL Commands
> - Console Commands

SQL - `REBUILD INDEXES`

Rebuilds automatic indexes.

Syntax

```
REBUILD INDEX <index>
```

- **`<index>`** Defines the index that you want to rebuild. Use `*` to rebuild all automatic indexes.

> **NOTE**: During the rebuild, any idempotent queries made against the index, skip the index and perform sequential scans. This means that queries run slower during this operation. Non-idempotent commands, such as `INSERT`, `UPDATE`, and `DELETE` are blocked waiting until the indexes are rebuilt.

Examples

- Rebuild an index on the `nick` property on the class `Profile`:

  ```
  orientdb> REBUILD INDEX Profile.nick
  ```

- Rebuild all indexes:

  ```
  orientdb> REBUILD INDEX *
  ```

> For more information, see
>
> - `CREATE INDEX`
> - `DROP INDEX`
> - Indexes
> - SQL commands

SQL - `REVOKE`

Changes permissions of a role, revoking access to one or more resources. To give access to a resource to the role, see the `GRANT` command.

Syntax

```
REVOKE <permission> ON <resource> FROM <role>
```

- **`<permission>`** Defines the permission you want to revoke from the role.
- **`<resource>`** Defines the resource on which you want to revoke the permissions.
- **`<role>`** Defines the role you want to revoke the permissions.

Examples

- Revoke permission to delete records on any cluster to the role `backoffice`:

  ```
  orientdb> REVOKE DELETE ON database.cluster.* FROM backoffice
  ```

> For more information, see
>
> - SQL commands.

Supported Permissions

Using this command, you can grant the following permissions to a role.

Permission	Description
`NONE`	Revokes no permissions on the resource.
`CREATE`	Revokes create permissions on the resource, such as the `CREATE CLASS` or `CREATE CLUSTER` commands.
`READ`	Revokes read permissions on the resource, such as the `SELECT` query.
`UPDATE`	Revokes update permissions on the resource, such as the `UPDATE` or `UPDATE EDGE` commands.
`DELETE`	Revokes delete permissions on the resource, such as the `DROP INDEX` or `DROP SEQUENCE` commands.
`ALL`	Revokes all permissions on the resource.

Supported Resources

Using this command, you can grant permissions on the following resources.

Resource	Description
`database`	Revokes access on the current database.
`database.class.<class>`	Revokes access on records contained in the indicated class. Use `**` to indicate all classes.
`database.cluster.<cluster>`	Revokes access to records contained in the indicated cluster. Use `**` to indicate all clusters.
`database.query`	Revokes the ability to execute a query, (`READ` is sufficient).
`database.command.<command>`	Revokes the ability to execute the given command. Use `CREATE` for `INSERT` , `READ` for `SELECT` , `UPDATE` for `UPDATE` and `DELETE` for `DELETE` .
`database.config.<permission>`	Revokes access to the configuration. Valid permissions are `READ` and `UPDATE` .
`database.hook.record`	Revokes the ability to set hooks.
`server.admin`	Revokes the ability to access server resources.

SQL - `SELECT`

OrientDB supports the SQL language to execute queries against the database engine. For more information, see operators and functions. For more information on the differences between this implementation and the SQL-92 standard, see OrientDB SQL.

Syntax:

```
SELECT [ <Projections> ] [ FROM <Target> [ LET <Assignment>* ] ]
    [ WHERE <Condition>* ]
    [ GROUP BY <Field>* ]
    [ ORDER BY <Fields>* [ ASC|DESC ] * ]
    [ UNWIND <Field>* ]
    [ SKIP <SkipRecords> ]
    [ LIMIT <MaxRecords> ]
    [ FETCHPLAN <FetchPlan> ]
    [ TIMEOUT <Timeout> [ <STRATEGY> ]
    [ LOCK default|record ]
    [ PARALLEL ]
    [ NOCACHE ]
```

- **`<Projections>`** Indicates the data you want to extract from the query as the result-set. Note: In OrientDB, this variable is optional. In the projections you can define aliases for single fields, using the `AS` keyword; in current release aliases cannot be used in the WHERE condition, GROUP BY and ORDER BY (they will be evaluated to null)
- **`FROM`** Designates the object to query. This can be a class, cluster, single Record ID, set of Record ID's, or (beginning in version 1.7.7) index values sorted by ascending or descending key order.
  - When querying a class, for `<target>` use the class name.
  - When querying a cluster, for `<target>` use `CLUSTER:<cluster-name>` (eg. `CLUSTER:person`) or `CLUSTER:<cluster-id>` (eg. `CLUSTER:12`). This causes the query to execute only on records in that cluster.
  - When querying record ID's, you can specific one or a small set of records to query. This is useful when you need to specify a starting point in navigating graphs.
  - When querying indexes, use the following prefixes:
    - `INDEXVALUES:<index>` and `INDEXVALUESASC:<index>` sorts values into an ascending order of index keys.
    - `INDEXVALUESDESC:<index>` sorts the values into a descending order of index keys.
- **`WHERE`** Designates conditions to filter the result-set.
- **`LET`** Binds context variables to use in projections, conditions or sub-queries.
- **`GROUP BY`** Designates field on which to group the result-set. In the current release, you can only group on one field.
- **`ORDER BY`** Designates the field with which to order the result-set. Use the optional `ASC` and `DESC` operators to define the direction of the order. The default is ascending. Additionally, if you are using a projection, you need to include the `ORDER BY` field in the projection. Note that ORDER BY works only on projection fields (fields that are returned in the result set) not on LET variables.
- **`UNWIND`** Designates the field on which to unwind the collection. Introduced in version 2.1.
- **`SKIP`** Defines the number of records you want to skip from the start of the result-set. You may find this useful in pagination, when using it in conjunction with `LIMIT` .
- **`LIMIT`** Defines the maximum number of records in the result-set. You may find this useful in pagination, when using it in conjunction with `SKIP` .
- **`FETCHPLAN`** Defines how you want it to fetch results. For more information, see Fetching Strategy.
- **`TIMEOUT`** Defines the maximum time in milliseconds for the query. By default, queries have no timeouts. If you don't specify a timeout strategy, it defaults to `EXCEPTION` . These are the available timeout strategies:
  - `RETURN` Truncate the result-set, returning the data collected up to the timeout.
  - `EXCEPTION` Raises an exception.
- **`LOCK`** Defines the locking strategy. These are the available locking strategies:
  - `DEFAULT` Locks the record for the read.
  - `RECORD` Locks the record in exclusive mode for the current transaction, until the transaction commits or you perform a rollback operation.
- **`PARALLEL`** Executes the query against *x* concurrent threads, where *x* refers to the number of processors or cores found on the host operating system of the query. You may find `PARALLEL` execution useful on long running queries or queries that involve multiple cluster. For simple queries, using `PARALLEL` may cause a slow down due to the overhead inherent in using multiple threads.

- `NOCACHE` Defines whether you want to avoid using the cache.

> **NOTE**: Beginning with version 1.0 rc 7, the `RANGE` operator was removed. To execute range queries, instead use the `BETWEEN` operator against `@RID`. For more information, see Pagination.

Examples:

- Return all records of the class `Person`, where the name starts with `Luk`:

  ```
  orientdb> SELECT FROM Person WHERE name LIKE 'Luk%'
  ```

  Alternatively, you might also use either of these queries:

  ```
  orientdb> SELECT FROM Person WHERE name.left(3) = 'Luk'
  orientdb> SELECT FROM Person WHERE name.substring(0,3) = 'Luk'
  ```

- Return all records of the type `!AnimalType` where the collection `races` contains at least one entry where the first character is `e`, ignoring case:

  ```
  orientdb> SELECT FROM animaltype WHERE races CONTAINS( name.toLowerCase().subString(
            0, 1) = 'e' )
  ```

- Return all records of type `!AnimalType` where the collection `races` contains at least one entry with names `European` or `Asiatic`:

  ```
  orientdb> SELECT * FROM animaltype WHERE races CONTAINS(name in ['European',
            'Asiatic'])
  ```

- Return all records in the class `Profile` where any field contains the word `danger`:

  ```
  orientdb> SELECT FROM Profile WHERE ANY() LIKE '%danger%'
  ```

- Return any record at any level that has the word `danger`:

  DEPRECATED SYNTAX

  ```
  orientdb> SELECT FROM Profile WHERE ANY() TRAVERSE( ANY() LIKE '%danger%' )
  ```

- Return any record where up to the third level of connections has some field that contains the word `danger`, ignoring case:

  ```
  orientdb> SELECT FROM Profile WHERE ANY() TRAVERSE(0, 3) (
            ANY().toUpperCase().indexOf('danger') > -1 )
  ```

- Return all results on class `Profile`, ordered by the field `name` in descending order:

  ```
  orientdb> SELECT FROM Profile ORDER BY name DESC
  ```

- Return the number of records in the class `Account` per city:

  ```
  orientdb> SELECT SUM(*) FROM Account GROUP BY city
  ```

- Traverse records from a root node:

  ```
  orientdb> SELECT FROM 11:4 WHERE ANY() TRAVERSE(0,10) (address.city = 'Rome')
  ```

- Return only a limited set of records:

```
orientdb> SELECT FROM [#10:3, #10:4, #10:5]
```

- Return three fields from the class `Profile` :

```
orientdb> SELECT nick, followings, followers FROM Profile
```

- Return the field `name` in uppercase and the field country name of the linked city of the address:

```
orientdb> SELECT name.toUppercase(), address.city.country.name FROM Profile
```

- Return records from the class `Profile` in descending order of their creation:

```
orientdb> SELECT FROM Profile ORDER BY @rid DESC
```

- Querying an index

```
orientdb> select from index:ouser.name where key = 'admin'

|   key    |  rid   |
| "admin"  |  #5:0  |
```

A query on an index returns pairs of index keys and values. You can expand the values using a `select expand(rid) from...`

Beginning in version 1.7.7, OrientDB can open an inverse cursor against clusters. This is very fast and doesn't require the classic ordering resources, CPU and RAM.

Projections

In the standard implementations of SQL, projections are mandatory. In OrientDB, the omission of projects translates to its returning the entire record. That is, it reads no projection as the equivalent of the `*` wildcard.

```
orientdb> SELECT FROM Account
```

For all projections except the wildcard `*` , it creates a new temporary document, which does not include the `@rid` and `@version` fields of the original record.

```
orientdb> SELECT name, age FROM Account
```

The naming convention for the returned document fields are:

- Field name for plain fields, like `invoice` becoming `invoice` .
- First field name for chained fields, like `invoice.customer.name` becoming `invoice` .
- Function name for functions, like `MAX(salary)` becoming `max` .

In the event that the target field exists, it uses a numeric progression. For instance,

```
orientdb> SELECT MAX(incoming), MAX(cost) FROM Balance

------+------
 max  | max2
------+------
 1342 | 2478
------+------
```

To override the display for the field names, use the `AS`.

```
orientdb> SELECT MAX(incoming) AS max_incoming, MAX(cost) AS max_cost FROM Balance

---------------+----------
 max_incoming  | max_cost
---------------+----------
 1342          | 2478
---------------+----------
```

With the dollar sign `$`, you can access the context variables. Each time you run the command, OrientDB accesses the context to read and write the variables. For instance, say you want to display the path and depth levels up to the fifth of a `TRAVERSE` on all records in the `Movie` class.

```
orientdb> SELECT $path, $depth FROM ( TRAVERSE * FROM Movie WHERE $depth
```

`LET` Block

The `LET` block contains context variables to assign each time OrientDB evaluates a record. It destroys these values once the query execution ends. You can use context variables in projections, conditions, and sub-queries.

Assigning Fields for Reuse

OrientDB allows for crossing relationships. In single queries, you need to evaluate the same branch of the nested relationship. This is better than using a context variable that refers to the full relationship.

```
orientdb> SELECT FROM Profile WHERE address.city.name LIKE '%Saint%"' AND
          ( address.city.country.name = 'Italy' OR
            address.city.country.name = 'France' )
```

Using the `LET` makes the query shorter and faster, because it traverses the relationships only once:

```
orientdb> SELECT FROM Profile LET $city = address.city WHERE $city.name LIKE
          '%Saint%"' AND ($city.country.name = 'Italy' OR $city.country.name = 'France')
```

In this case, it traverses the path till `address.city` only once.

Sub-query

The `LET` block allows you to assign a context variable to the result of a sub-query.

```
orientdb> SELECT FROM Document LET $temp = ( SELECT @rid, $depth FROM (TRAVERSE
          V.OUT, E.IN FROM $parent.current ) WHERE @class = 'Concept' AND
          ( id = 'first concept' OR id = 'second concept' )) WHERE $temp.SIZE() > 0
```

`LET` Block in Projection

You can use context variables as part of a result-set in projections. For instance, the query below displays the city name from the previous example:

```
orientdb> SELECT $temp.name FROM Profile LET $temp = address.city WHERE $city.name
          LIKE '%Saint%"' AND ( $city.country.name = 'Italy' OR
          $city.country.name = 'France' )
```

Unwinding

Beginning with version 2.1, OrientDB allows unwinding of collection fields and obtaining multiple records as a result, one for each element in the collection:

```
orientdb> SELECT name, OUT("Friend").name AS friendName FROM Person

--------+-------------------
 name   | friendName
--------+-------------------
 'John' | ['Mark', 'Steve']
--------+-------------------
```

In the event if you want one record for each element in `friendName`, you can rewrite the query using `UNWIND`:

```
orientdb> SELECT name, OUT("Friend").name AS friendName FROM Person UNWIND friendName

--------+-------------
 name   | friendName
--------+-------------
 'John' | 'Mark'
 'John' | 'Steve'
--------+-------------
```

> **NOTE**: For more information on other SQL commands, see SQL commands.

History

- **1.7.7**: New target prefixes `INDEXVALUES:`, `INDEXVALUESASC:` and `INDEXVALUESDESC:` added.
- **1.7**: `PARALLEL` keyword added to execute the query against *x* concurrent threads, where *x* is the number of processors or cores found on the operating system where the query runs. `PARALLEL` execution is useful on long running queries or queries that involve multiple clusters. On simple queries, using `PARALLEL` can cause a slow down due to the overhead of using multiple threads.

SQL - `TRAVERSE`

Retrieves connected records crossing relationships. This works with both the Document and Graph API's, meaning that you can traverse relationships between say invoices and customers on a graph, without the need to model the domain using the Graph API.

!	In many cases, you may find it more efficient to use `SELECT`, which can result in shorter and faster queries. For more information, see `TRAVERSE` versus `SELECT` below.

Syntax

```
TRAVERSE <[class.]field>|*|any()|all()
         [FROM <target>]
         [
           MAXDEPTH <number>
           |
           WHILE <condition>
         ]
         [LIMIT <max-records>]
         [STRATEGY <strategy>]
```

- `<fields>` Defines the fields you want to traverse.
- `<target>` Defines the target you want to traverse. This can be a class, one or more clusters, a single Record ID, set of Record ID's, or a sub-query.
- `MAXDEPTH` Defines the maximum depth of the traversal. `0` indicates that you only want to traverse the root node. Negative values are invalid.
- `WHILE` Defines the condition for continuing the traversal while it is true.
- `LIMIT` Defines the maximum number of results the command can return.
- `STRATEGY` Defines strategy for traversing the graph.

> **NOTE**: The use of the `WHERE` clause has been deprecated for this command.
>
> **NOTE**: There is a difference between `MAXDEPTH N` and `WHILE DEPTH <= N`: the `MAXDEPTH` will evaluate exactly N levels, while the `WHILE` will evaluate N+1 levels and will discard the N+1th, so the `MAXDEPTH` in general has better performance.

Examples

In a social network-like domain, a user profile is connected to friends through links. The following examples consider common operations on a user with the record ID `#10:1234`.

- Traverse all fields in the root record:

```
orientdb> TRAVERSE * FROM #10:1234
```

- Specify fields and depth up to the third level, using the `BREADTH_FIRST` strategy:

```
orientdb> TRAVERSE out("Friend") FROM #10:1234 MAXDEPTH 3
          STRATEGY BREADTH_FIRST
```

- Execute the same command, this time filtering for a minimum depth to exclude the first target vertex:

```
orientdb> SELECT FROM (TRAVERSE out("Friend") FROM #10:1234 MAXDEPTH 3)
          WHERE $depth >= 1
```

> **NOTE**: You can also define the maximum depth in the `SELECT` command, but it's much more efficient to set it at the inner `TRAVERSE` statement because the returning record sets are already filtered by depth.

- Combine traversal with `SELECT` command to filter the result-set. Repeat the above example, filtering for users in Rome:

```
orientdb> SELECT FROM (TRAVERSE out("Friend") FROM #10:1234 MAXDEPTH 3)
          WHERE city = 'Rome'
```

- Extract movies of actors that have worked, at least once, in any movie produced by J.J. Abrams:

```
orientdb> SELECT FROM (TRAVERSE out("Actors"), out("Movies") FROM (SELECT FROM
          Movie WHERE producer = "J.J. Abrams") MAXDEPTH 3) WHERE
          @class = 'Movie'
```

- Display the current path in the traversal:

```
orientdb> SELECT $path FROM ( TRAVERSE out() FROM V MAXDEPTH 10 )
```

Supported Variables

Fields

Defines the fields that you want to traverse. If set to `*`, `any()` or `all()` then it traverses all fields. This can prove costly to performance and resource usage, so it is recommended that you optimize the command to only traverse the pertinent fields.

In addition to his, you can specify the fields at a class-level. Polymorphism is supported. By specifying `Person.city` and the class `Customer` extends person, you also traverse fields in `Customer`.

Field names are case-sensitive, classes not.

Target

Targets for traversal can be,

- **`<class>`** Defines the class that you want to traverse.
- **`CLUSTER:<cluster>`** Defines the cluster you want to traverse.
- **`<record-id>`** Individual root Record ID that you want to traverse.
- **`[<record-id>,<record-id>,...]`** Set of Record ID's that you want to traverse. This is useful when navigating graphs starting from the same root nodes.

Context Variables

In addition to the above, you can use the following context variables in traversals:

- **`$parent`** Gives the parent context, if any. You may find this useful when traversing from a sub-query.
- **`$current`** Gives the current record in the iteration. To get the upper-level record in nested queries, you can use `$parent.$current`.
- **`$depth`** Gives the current depth of nesting.
- **`$path`** Gives a string representation of the current path. For instance, `#5:0#.out`. You can also display it through `SELECT`:

```
orientdb> SELECT $path FROM (TRAVERSE * FROM V)
```

- **`$stack`** Gives a list of operations in the stack. Use it to access the traversal history. It's a `List<OTraverseAbstractProcess<?>>`, where the process implementations are:
  - *`OTraverseRecordSetProcess`* The base target of traversal, usually the first given.
  - *`OTraverseRecordProcess`* The traversed record.
  - *`OTraverseFieldProcess`* The traversal through the record's fields.
  - *`OTraverseMultiValueProcess`* Use on fields that are multivalue, such as arrays, collections and maps.
- **`$history`** Gives a set of records traversed as `SET<ORID>`.

Use Cases

`TRAVERSE` versus `SELECT`

When you already know traversal information, such as relationship names and depth-level, consider using `SELECT` instead of `TRAVERSE` as it is faster in some cases.

For example, this query traverses the `follow` relationship on Twitter accounts, getting the second level of friendship:

```
orientdb> SELECT FROM (TRAVERSE out('follow') FROM TwitterAccounts MAXDEPTH 2) WHERE $depth = 2
```

But, you could also express this same query using `SELECT` operation, in a way that is also shorter and faster:

```
orientdb> SELECT out('follow').out('follow') FROM TwitterAccounts
```

`TRAVERSE` with the Graph Model and API

While you can use the `TRAVERSE` command with any domain model, it provides the greatest utility in [Graph Databases[(Graph-Database-Tinkerpop.md) model.

This model is based on the concepts of the Vertex (or Node) as the class `V` and the Edge (or Arc, Connection, Link, etc.) as the class `E`. If you want to traverse in a direction, you have to use the class name when declaring the traversing fields. The supported directions are:

- **Vertex to outgoing edges** Using `outE()` or `outE('EdgeClassName')`. That is, going out from a vertex and into the outgoing edges.
- **Vertex to incoming edges** Using `inE()` or `inE('EdgeClassName')`. That is, going from a vertex and into the incoming edges.
- **Vertex to all edges** Using `bothE()` or `bothE('EdgeClassName')`. That is, going from a vertex and into all the connected edges.
- **Edge to Vertex (end point)** Using `inV()`. That is, going out from an edge and into a vertex.
- **Edge to Vertex (starting point)** Using `outV()`. That is, going back from an edge and into a vertex.
- **Edge to Vertex (both sizes)** Using `bothV()`. That is, going from an edge and into connected vertices.
- **Vertex to Vertex (outgoing edges)** Using `out()` or `out('EdgeClassName')`. This is the same as `outE().inV()`
- **Vertex to Vertex (incoming edges)** Using `in()` or `in('EdgeClassName')`. This is the same as `outE().inV()`
- **Vertex to Vertex (all directions)** Using `both()` or `both('EdgeClassName')`.

For instance, traversing outgoing edges on the record `#10:3434`:

```
orientdb> TRAVERSE out() FROM #10:3434
```

In a domain for emails, to find all messages sent on January 1, 2012 from the user Luca, assuming that they are stored in the vertex class `User` and that the messages are contained in the vertex class `Message`. Sent messages are stored as `out` connections on the edge class `SentMessage`:

```
orientdb> SELECT FROM (TRAVERSE outE(), inV() FROM (SELECT FROM User WHERE
          name = 'Luca') MAXDEPTH 2 AND (@class = 'Message' or
          (@class = 'SentMessage' AND sentOn = '01/01/2012') )) WHERE
          @class = 'Message'
```

Deprecated `TRAVERSE` Operator

Before the introduction of the `TRAVERSE` command, OrientDB featured a `TRAVERSE` operator, which worked in a different manner and was applied to the `WHERE` condition.

More recent releases deprecated this operator. It is recommended that you transition to the `TRAVERSE` command with `SELECT` queries to utilize more power.

The deprecated syntax for the `TRAVERSE` operator looks like this:

	WARNING: Beginning in version 2.1, OrientDB no longer supports this syntax.

Syntax

```
SELECT FROM <target> WHERE <field> TRAVERSE[(<minDeep> [,<maxDeep> [,<fields>]])] (<conditions>)
```

- **`<target>`** Defines the query target.
- **`<field>`** Defines the field to traverse. Supported fields are,
  - *`out`* Gives outgoing edges.
  - *`in`* Gives incoming edges.
  - *`any()`* Any field, including `in` and `out` .
  - *`all()`* All fields, including `in` and `out` .
  - Any attribute of the vertex.
- *`minDeep`* Defines the minimum depth-level to begin applying the conditions. This is usually `0` for the root vertex, or `1` for only the outgoing vertices.
- **`maxDeep`** Defines the maximum depth-level to read. Default is `-1` , for infinite depth.
- **`[<field>, <field>,...]`** Defines a list of fields to traverse. Default is `any()` .
- **`<conditions>`** Defines conditions to check on any traversed vertex.

> For more information, see SQL syntax.

Examples

- Find all vertices that have at least one friend, (connected through `out`), up to the third depth, that lives in Rome:

  ```
  orientdb> SELECT FROM Profile WHERE any() TRAVERSE(0,3) (city = 'Rome')
  ```

- Alternatively, you can write the above as:

  ```
  orientdb> SELECT FROM Profile LET $temp = (SELECT FROM (TRAVERSE * FROM $current
            MAXDEPTH 3) WHERE city = 'Rome') WHERE $temp.size() > 0
  ```

- Consider an example using the Graph Query, with the following schema:

  ```
  Vertex    edge         Vertex
  User----->Friends----->User
            Label='f'
  ```

- Find the first-level friends of the user with the Record ID `#10:11` :

  ```
  orientdb> SELECT DISTINCT(in.lid) AS lid,distinct(in.fid) AS fid FROM
            (TRAVERSE outE(), inV() FROM #10:11 MAXDEPTH 1) WHERE
            @class = 'Friends'
  ```

- By changing the depth to 3, you can find the second-level friends of the user:

  ```
  orientdb> SELECT distinct(in.lid) AS lid, distinct(in.fid) AS fid FROM
            (TRAVERSE outE(), inV() FROM #10:11 WHILE MAXDEPTH 3) WHERE
            @class = 'Friends'
  ```

> For more information, see
>
> - Java-Traverse page.
> - SQL Commands

SQL - `TRUNCATE CLASS`

Deletes records of all clusters defined as part of the class.

By default, every class has an associated cluster with the same name. This command operates at a lower level than `DELETE`. This commands ignores sub-classes, (That is, their records remain in their clusters). If you want to also remove all records from the class hierarchy, you need to use the `POLYMORPHIC` keyword.

Truncation is not permitted on vertex or edge classes, but you can force its execution using the `UNSAFE` keyword. Forcing truncation is strongly discouraged, as it can leave the graph in an inconsistent state.

Syntax

```
TRUNCATE CLASS <class> [ POLYMORPHIC ] [ UNSAFE ]
```

- **`<class>`** Defines the class you want to truncate.
- **`POLYMORPHIC`** Defines whether the command also truncates the class hierarchy.
- **`UNSAFE`** Defines whether the command forces truncation on vertex or edge classes, (that is, sub-classes that extend the classes `V` or `E`).

Examples

- Remove all records of the class `Profile`:

  ```
  orientdb> TRUNCATE CLASS Profile
  ```

> For more information, see
>
> - `DELETE`
> - `TRUNCATE CLUSTER`
> - `CREATE CLASS`
> - SQL Commands
> - Console Commands

SQL - `TRUNCATE CLUSTER`

Deletes all records of a cluster. This command operates at a lower level than the standard `DELETE` command.

Syntax

```
TRUNCATE CLUSTER <cluster>
```

- **`<cluster>`** Defines the cluster to delete.

Examples

- Remove all records in the cluster `profile` :

  ```
  orientdb> TRUNCATE CLUSTER profile
  ```

> For more information, see
>
> - `DELETE`
> - `TRUNCATE CLASS`
> - SQL Commands
> - Console Commands

SQL - `TRUNCATE RECORD`

Deletes a record or records without loading them. Useful in cases where the record is corrupted in a way that prevents OrientDB from correctly loading it.

Syntax

```
TRUNCATE RECORD <record-id>*
```

- **`<record-id>`** Defines the Record ID you want to truncate. You can also truncate multiple records using a comma-separated list within brackets.

This command returns the number of records it truncates.

Examples

- Truncate a record:

  ```
  orientdb> TRUNCATE RECORD 20:3
  ```

- Truncate three records together:

  ```
  orientdb> TRUNCATE RECORD [20:0, 20:1, 20:2]
  ```

> For more information, see
>
> - `DELETE`
> - SQL Commands
> - Console Commands

SQL - `UPDATE`

Update one or more records in the current database. Remember: OrientDB can work in schema-less mode, so you can create any field on-the-fly. Furthermore, the command also supports extensions to work on collections.

Syntax:

```
UPDATE <class>|CLUSTER:<cluster>|<recordID>
  [SET|INCREMENT|ADD|REMOVE|PUT <field-name> = <field-value>[,]*]|[CONTENT|MERGE <JSON>]
  [UPSERT]
  [RETURN <returning> [<returning-expression>]]
  [WHERE <conditions>]
  [LOCK default|record]
  [LIMIT <max-records>] [TIMEOUT <timeout>]
```

- **`SET`** Defines the fields to update.
- **`INCREMENT`** Increments the field by the value.

  For instance, record at `10` with `INCREMENT value = 3` sets the new value to `13`. You may find this useful in atomic updates of counters. Use negative numbers to decrement. Additionally, you can use `INCREMENT` to implement sequences and auto-increment.

- **`ADD`** Adds a new item in collection fields.
- **`REMOVE`** Removes an item in collection and map fields.
- **`PUT`** Puts an entry into a map field.
- **`CONTENT`** Replaces the record content with a JSON document.
- **`MERGE`** Merges the record content with a JSON document.
- **`LOCK`** Specifies how to lock the record between the load and update. You can use one of the following lock strategies:
  - `DEFAULT` No lock. Use in the event of concurrent updates, the MVCC throws an exception.
  - `RECORD` Locks the record during the update.
- **`UPSERT`** Updates a record if it exists or inserts a new record if it doesn't. This avoids the need to execute two commands, (one for each condition, inserting and updating).

  `UPSERT` requires a `WHERE` clause and a class target. There are further limitations on `UPSERT`, explained below.

- **`RETURN`** Specifies an expression to return instead of the record and what to do with the result-set returned by the expression. The available return operators are:
  - `COUNT` Returns the number of updated records. This is the default return operator.
  - `BEFORE` Returns the records before the update.
  - `AFTER` Return the records after the update.
- `WHERE`
- `LIMIT` Defines the maximum number of records to update.
- `TIMEOUT` Defines the time you want to allow the update run before it times out.

> **NOTE**: The Record ID must have a `#` prefix. For instance, `#12:3`.

Examples:

- Update to change the value of a field:

  ```
  orientdb> UPDATE Profile SET nick = 'Luca' WHERE nick IS NULL

  Updated 2 record(s) in 0.008000 sec(s).
  ```

- Update to remove a field from all records:

  ```
  orientdb> UPDATE Profile REMOVE nick
  ```

- Update to add a value into a collection:

```
orientdb> UPDATE Account ADD address=#12:0
```

> **NOTE**: Beginning with version 2.0.5, the OrientDB server generates a server error if there is no space between `#` and the `=`. You must write the command as:
>
> ```
> orientdb> UPDATE Account ADD address = #12:0
> ```

- Update to remove a value from a collection, if you know the exact value that you want to remove:

  Remove an element from a link list or set:

  ```
  orientdb> UPDATE Account REMOVE address = #12:0
  ```

  Remove an element from a list or set of strings:

  ```
  orientdb> UPDATE Account REMOVE addresses = 'Foo'
  ```

- Update to remove a value, filtering on value attributes.

  Remove addresses based in the city of Rome:

  ```
  orientdb> UPDATE Account REMOVE addresses = addresses[city = 'Rome']
  ```

- Update to remove a value, filtering based on position in the collection.

  ```
  orientdb> UPDATE Account REMOVE addresses = addresses[1]
  ```

  This remove the second element from a list, (position numbers start from `0`, so `addresses[1]` is the second elelment).

- Update to put a map entry into the map:

  ```
  orientdb> UPDATE Account PUT addresses = 'Luca', #12:0
  ```

- Update to remove a value from a map

  ```
  orientdb> UPDATE Account REMOVE addresses = 'Luca'
  ```

- Update an embedded document. The `UPDATE` command can take JSON as a value to update.

  ```
  orientdb> UPDATE Account SET address={ "street": "Melrose Avenue", "city": {
            "name": "Beverly Hills" } }
  ```

- Update the first twenty records that satisfy a condition:

  ```
  orientdb> UPDATE Profile SET nick = 'Luca' WHERE nick IS NULL LIMIT 20
  ```

- Update a record or insert if it doesn't already exist:

  ```
  orientdb> UPDATE Profile SET nick = 'Luca' UPSERT WHERE nick = 'Luca'
  ```

- Update a web counter, avoiding concurrent accesses:

```
orientdb> UPDATE Counter INCREMENT views = 1 WHERE pages = '/downloads/'
          LOCK RECORD
```

- Updates using the `RETURN` keyword:

```
orientdb> UPDATE #7:0 SET gender='male' RETURN AFTER @rid
orientdb> UPDATE #7:0 SET gender='male' RETURN AFTER @version
orientdb> UPDATE #7:0 SET gender='male' RETURN AFTER @this
orientdb> UPDATE #7:0 INCREMENT Counter = 123 RETURN BEFORE $current.Counter
orientdb> UPDATE #7:0 SET gender='male' RETURN AFTER $current.exclude(
          "really_big_field")
orientdb> UPDATE #7:0 ADD out_Edge = #12:1 RETURN AFTER $current.outE("Edge")
```

In the event that a single field is returned, OrientDB wraps the result-set in a record storing the value in the field `result`. This avoids introducing a new serialization, as there is no primitive values collection serialization in the binary protocol. Additionally, it provides useful fields like `version` and `rid` from the original record in corresponding fields. The new syntax allows for optimization of client-server network traffic.

For more information on SQL syntax, see `SELECT`.

Limitations of the `UPSERT` Clause

The `UPSERT` clause only guarantees atomicity when you use a `UNIQUE` index and perform the look-up on the index through the `WHERE` condition.

```
orientdb> UPDATE Client SET id = 23 UPSERT WHERE id = 23
```

Here, you must have a unique index on `Client.id` to guarantee uniqueness on concurrent operations.

SQL - `UPDATE EDGE`

Updates edge records in the current database. This is the equivalent of the `UPDATE` command, with the addition of checking and maintaining graph consistency with vertices, in the event that you update the `out` and `in` properties.

Bear in mind that OrientDB can also work in schema-less mode, allowing you to create fields on the fly. Furthermore, that it works on collections and necessarily includes some extensions to the standard SQL for handling collections.

This command was introduced in version 2.2.

Syntax

```
UPDATE EDGE <edge>
  [SET|INCREMENT|ADD|REMOVE|PUT <field-name> = <field-value>[,]*]|[CONTENT|MERGE <JSON>]
  [RETURN <returning> [<returning-expression>]]
  [WHERE <conditions>]
  [LOCK default|record]
  [LIMIT <max-records>] [TIMEOUT <timeout>]
```

- `<edge>` Defines the edge that you want to update. You can choose between:
  - *Class* Updating edges by class.
  - *Cluster* Updating edges by cluster, using `CLUSTER` prefix.
  - *Record ID* Updating edges by Record ID.
- `SET` Updates the field to the given value.
- `INCREMENT` Increments the given field by the value.
- `ADD` Defines an item to add to a collection of fields.
- `REMOVE` Defines an item to remove from a collection of fields.
- `PUT` Defines an entry to put into a map field.
- `RETURN` Defines the expression you want to return after running the update.
  - `COUNT` Returns the number of updated records. This is the default operator.
  - `BEFORE` Returns the records before the update.
  - `AFTER` Returns the records after the update.
- `WHERE` Defines the filter conditions.
- `LOCK` Defines how the record locks between the load and update. You can choose between the following lock strategies:
  - `DEFAULT` Disables locking. Use this in the event of concurrent updates. It throws an exception in the event of conflict.
  - `RECORD` Locks the record during the update.
- `LIMIT` Defines the maximum number of records to update.

Examples

- Change the edge endpoint:

  ```
  orientdb> UPDATE EDGE Friend SET out = (SELECT FROM Person WHERE name = 'John')
            WHERE foo = 'bar'
  ```

> For more information, see
>
> - `UPDATE`
> - SQL Commands

SQL - Filtering

The Where condition is shared among many SQL commands.

Syntax

`[<item>] <operator> <item>`

Items

And `item` can be:

What	Description	Example	Available since
field	Document field	where *price* > 1000000	0.9.1
field<indexes>	Document field part. To know more about field part look at the full syntax: Document_Field_Part	where tags[name='Hi'] or tags[0-3] IN ('Hello') and employees IS NOT NULL	1.0rc5
record attribute	Record attribute name with @ as prefix	where *@class* = 'Profile'	0.9.21
column	The number of the column. Useful in Column Database	where *column(1)* > 300	0.9.1
any()	Represents any field of the Document. The condition is true if ANY of the fields matches the condition	where *any()* like 'L%'	0.9.10
all()	Represents all the fields of the Document. The condition is true if ALL the fields match the condition	where *all()* is null	0.9.10
functions	Any function between the defined ones	where distance(x, y, 52.20472, 0.14056) <= 30	0.9.25
$variable	Context variable prefixed with $	where $depth <= 3	1.2.0

Record attributes

Name	Description	Example	Available since
@this	returns the record it self	select **@this.toJSON()** from Account	0.9.25
@rid	returns the Record ID in the form <cluster:position>. It's null for embedded records. *NOTE: using @rid in where condition slow down queries. Much better to use the Record ID as target. Example: change this: select from Profile where @rid = #10:44 with this: select from #10:44*	**@rid** = #11:0	0.9.21
@class	returns Class name only for record of type Schema Aware. It's null for the others	**@class** = 'Profile'	0.9.21
@version	returns the record version as integer. Version starts from 0. Can't be null	**@version** > 0	0.9.21
@size	returns the record size in bytes	**@size** > 1024	0.9.21
@fields	returns the number of fields in document	select @fields from V	-
@type	returns the record type between: 'document', 'column', 'flat', 'bytes'	**@type** = 'flat'	0.9.21

Operators

Conditional Operators

Apply to	Operator	Description	Example	Available since
any	=	Equals to	name = 'Luke'	0.9.1
string	like	Similar to equals, but allow the wildcard '%' that means 'any'	name **like** 'Luk%'	0.9.1
any	<	Less than	age < 40	0.9.1
any	<=	Less than or equal to	age <= 40	0.9.1
any	>	Greater than	age > 40	0.9.1
any	>=	Greater than or equal to	age >= 40	0.9.1
any	<>	Not equals (same of !=)	age <> 40	0.9.1
any	BETWEEN	The value is between a range. It's equivalent to <field> >= <from-value> AND <field> <= <to-value>	price BETWEEN 10 and 30	1.0rc2
any	IS	Used to test if a value is NULL	children **is** null	0.9.6
record, string (as class name)	INSTANCEOF	Used to check if the record extends a class	@this **instanceof** 'Customer' or @class **instanceof** 'Provider'	1.0rc8
collection	IN	contains any of the elements listed	name **in** ['European','Asiatic']	
collection	CONTAINS	true if the collection contains at least one element that satisfy the next condition. Condition can be a single item: in this case the behaviour is like the IN operator	children **contains** (name = 'Luke') - map.values() **contains** (name = 'Luke')	0.9.7

collection	CONTAINSALL	true if all the elements of the collection satisfy the next condition	children *containsAll* (name = 'Luke')	0.9.7
map	CONTAINSKEY	true if the map contains at least one key equals to the requested. You can also use map.keys() CONTAINS in place of it	connections *containsKey* 'Luke'	0.9.22
map	CONTAINSVALUE	true if the map contains at least one value equals to the requested. You can also use map.values() CONTAINS in place of it	connections *containsValue* 10:3	0.9.22
string	CONTAINSTEXT	When used against an indexed field, a lookup in the index will be performed with the text specified as key. When there is no index a simple Java indexOf will be performed. So the result set could be different if you have an index or not on that field	text *containsText* 'jay'	0.9.22
string	MATCHES	Matches the string using a [http://www.regular-expressions.info/ Regular Expression]	text matches '\b[A-Z0-9.%+-]+@[A-Z0-9.-]+.[A-Z]{2,4}\b'	0.9.6
any	TRAVERSE[(<minDepth> [,<maxDepth> [,<fields>]]	*This function was born before the SQL Traverse statement and today it's pretty limited. Look at Traversing graphs to know more about traversing in better ways.* true if traversing the declared field(s) at the level from <minDepth> to <maxDepth> matches the condition. A minDepth = 0 means the root node, maxDepth = -1 means no limit: traverse all the graph recursively. If <minDepth> and <maxDepth> are not used, then (0, -1) will be taken. If <fields> is not passed, than any() will be used.	select from profile where any() **traverse(0,7,'followers,followings')** (address.city.name = 'Rome')	0.9.10 and 0.9.24 for <fields> parameter

Logical Operators

Operator	Description	Example	Available since
AND	true if both the conditions are true	name = 'Luke' **and** surname like 'Sky%'	0.9.1
OR	true if at least one of the condition is true	name = 'Luke' **or** surname like 'Sky%'	0.9.1
NOT	true if the condition is false. NOT needs parenthesis on the right with the condition to negate	**not** (name = 'Luke')	1.2

Mathematics Operators

Apply to	Operator	Description	Example	Available since
Numbers	+	Plus	age + 34	1.0rc7
Numbers	-	Minus	salary - 34	1.0rc7
Numbers	*	Multiply	factor * 1.3	1.0rc7
Numbers	/	Divide	total / 12	1.0rc7
Numbers	%	Mod	total % 3	1.0rc7

Starting from v1.4 OrientDB supports the `eval()` function to execute complex operations. Example:

```
select eval( "amount * 120 / 100 - discount" ) as finalPrice from Order
```

Methods

Also called "Field Operators", are are treated on a separate page.

Functions

All the SQL functions are treated on a separate page.

Variables

OrientDB supports variables managed in the context of the command/query. By default some variables are created. Below the table with the available variables:

Name	Description	Command(s)	Since
$parent	Get the parent context from a sub-query. Example: select from V let $type = (traverse * from $parent.$current.children)	SELECT and TRAVERSE	1.2.0
$root	Get the root context from a sub-query. Example: select from V let $type = (traverse * from $root.$current.children)	SELECT and TRAVERSE	1.2.0
$current	Current record to use in sub-queries to refer from the parent's variable	SELECT and TRAVERSE	1.2.0
$depth	The current depth of nesting	TRAVERSE	1.1.0
$path	The string representation of the current path. Example: #6:0.in.#5:0#.out. You can also display it with -> select $path from (traverse * from V)	TRAVERSE	1.1.0
$stack	The List of operation in the stack. Use it to access to the history of the traversal	TRAVERSE	1.1.0
$history	The set of all the records traversed as a Set<ORID>	TRAVERSE	1.1.0

To set custom variable use the LET keyword.

SQL - Functions

Bundled functions

Functions by category

Graph	Math	Collections	Misc
out()	eval()	set()	date()
in()	min()	map()	sysdate()
both()	max()	list()	format()
outE()	sum()	difference()	distance()
inE()	abs()	first()	ifnull()
bothE()		intersect()	coalesce()
bothV()			
outV()	avg()	distinct()	uuid()
inV()	count()	expand()	if()
traversedElement()	mode()	unionall()	
traversedVertex()	median()	flatten()	
traversedEdge()	percentile()	last()	
shortestPath()	variance()	symmetricDifference()	-
dijkstra()	stddev()		
astar()			

Functions by name

abs()	astar()	avg()	both()
bothE()	bothV()	coalesce()	count()
date()	difference()	dijkstra()	distance()
distinct()	eval()	expand()	format()
first()	flatten()	if()	ifnull()
in()	inE()	inv()	intersect()
list()	map()	min()	max()
median()	mode()	out()	outE()
outV()	percentile()	set()	shortestPath()
stddev()	sum()	symmetricDifference()	sysdate()
traversedElement()	traversedEdge()	traversedVertex()	unionall()
uuid()	variance()		

SQL Functions are all the functions bundled with OrientDB SQL engine. You can create your own Database Functions in any language supported by JVM. Look also to SQL Methods.

SQL Functions can work in 2 ways based on the fact that they can receive 1 or more parameters:

Aggregated mode

When only one parameter is passed, the function aggregates the result in only one record. The classic example is the `sum()` function:

```
SELECT SUM(salary) FROM employee
```

This will always return 1 record with the sum of salary field.

Inline mode

When two or more parameters are passed:

```
SELECT SUM(salary, extra, benefits) AS total FROM employee
```

This will return the sum of the field "salary", "extra" and "benefits" as "total".

In case you need to use a function inline, when you only have one parameter, then add "null" as the second parameter:

```
SELECT first( out('friends').name, null ) as firstFriend FROM Profiles
```

In the above example, the `first()` function doesn't aggregate everything in only one record, but rather returns one record per `Profile`, where the `firstFriend` is the first item of the collection received as the parameter.

Function Reference

out()

Get the adjacent outgoing vertices starting from the current record as Vertex.

Syntax: `out([<label-1>][,<label-n>]*)`

Available since: 1.4.0

Example

Get all the outgoing vertices from all the Vehicle vertices:

```
SELECT out() FROM V
```

Get all the incoming vertices connected with edges with label (class) "Eats" and "Favorited" from all the Restaurant vertices in Rome:

```
SELECT out('Eats','Favorited') FROM Restaurant WHERE city = 'Rome'
```

---

in()

Get the adjacent incoming vertices starting from the current record as Vertex.

Syntax:

```
in([<label-1>][,<label-n>]*)
```

Available since: 1.4.0

Example

Get all the incoming vertices from all the Vehicle vertices:

```
SELECT in() FROM V
```

Get all the incoming vertices connected with edges with label (class) "Friend" and "Brother":

```
SELECT in('Friend','Brother') FROM V
```

both()

Get the adjacent outgoing and incoming vertices starting from the current record as Vertex.

Syntax:

```
both([<label1>][,<label-n>]*)
```

Available since: 1.4.0

Example

Get all the incoming and outgoing vertices from vertex with rid #13:33:

```
SELECT both() FROM #13:33
```

Get all the incoming and outgoing vertices connected with edges with label (class) "Friend" and "Brother":

```
SELECT both('Friend','Brother') FROM V
```

outE()

Get the adjacent outgoing edges starting from the current record as Vertex.

Syntax:

```
outE([<label1>][,<label-n>]*)
```

Available since: 1.4.0

Example

Get all the outgoing edges from all the vertices:

```
SELECT outE() FROM V
```

Get all the outgoing edges of type "Eats" from all the SocialNetworkProfile vertices:

```
SELECT outE('Eats') FROM SocialNetworkProfile
```

inE()

Get the adjacent incoming edges starting from the current record as Vertex.

Syntax:

```
inE([<label1>][,<label-n>]*)
```

Example

Get all the incoming edges from all the vertices:

```
SELECT inE() FROM V
```

Get all the incoming edges of type "Eats" from the Restaurant 'Bella Napoli':

```
SELECT inE('Eats') FROM Restaurant WHERE name = 'Bella Napoli'
```

---

bothE()

Get the adjacent outgoing and incoming edges starting from the current record as Vertex.

Syntax: `bothE([<label1>][,<label-n>]*)`

Available since: 1.4.0

Example

Get both incoming and outgoing edges from all the vertices:

```
SELECT bothE() FROM V
```

Get all the incoming and outgoing edges of type "Friend" from the Profile with nick 'Jay'

```
SELECT bothE('Friend') FROM Profile WHERE nick = 'Jay'
```

---

bothV()

Get the adjacent outgoing and incoming Vertices starting from the current record as Edge.

Syntax: `bothV()`

Available since: 1.4.0

Example

Get both incoming and outgoing vertices from all the edges:

```
SELECT bothV() FROM E
```

outV()

Get outgoing vertices starting from the current record as Edge.

Syntax:

```
outV()
```

Available since: 1.4.0

Example

```
SELECT outV() FROM E
```

inV()

Get incoming vertices starting from the current record as Edge.

Syntax:

```
inV()
```

Available since: 1.4.0

Example

```
SELECT inV() FROM E
```

eval()

Syntax: `eval('<expression>')`

Evaluates the expression between quotes (or double quotes).

Available since: 1.4.0

Example

```
SELECT eval('price * 120 / 100 - discount') AS finalPrice FROM Order
```

coalesce()

Returns the first field/value not null parameter. If no field/value is not null, returns null.

Syntax:

```
coalesce(<field|value> [, <field-n|value-n>]*)
```

Available since: 1.3.0

Example

```
SELECT coalesce(amount, amount2, amount3) FROM Account
```

if()

Syntax:

```
if(<expression>, <result-if-true>, <result-if-false>)
```

Evaluates a condition (first parameters) and returns the second parameter if the condition is true, the third one otherwise

Example:

```
SELECT if(eval("name = 'John'"), "My name is John", "My name is not John") FROM Person
```

ifnull()

Returns the passed field/value (or optional parameter *return_value_if_not_null*). If field/value is not null, otherwise it returns *return_value_if_null*.

Syntax:

```
ifnull(<field/value>, <return_value_if_null>)
```

Available since: 1.3.0

Example

```
SELECT ifnull(salary, 0) FROM Account
```

---

expand()

Available since: 1.4.0

This function has two meanings:

- When used on a collection field, it unwinds the collection in the field and use it as result.
- When used on a link (RID) field, it expands the document pointed by that link.

Syntax: `expand(<field>)`

Since version 2.1 the preferred operator to unwind collections is UNWIND. Expand usage for this use case will probably be deprecated in next releases

Example

on collectinos:

```
SELECT EXPAND( addresses ) FROM Account  //where addresses is a link list
```

on RIDs

```
SELECT EXPAND( country ) FROM City //where country is a link
```

This replaces the flatten() now deprecated

---

flatten()

> Deprecated, use the EXPAND() instead.

Extracts the collection in the field and use it as result.

Syntax:

```
flatten(<field>)
```

Available since: 1.0rc1

Example

```
SELECT flatten( addresses ) FROM Account
```

---

first()

Retrieves only the first item of multi-value fields (arrays, collections and maps). For non multi-value types just returns the value.

Syntax: `first(<field>)`

Available since: 1.2.0

Example

```
select first( addresses ) from Account
```

---

last()

Retrieves only the last item of multi-value fields (arrays, collections and maps). For non multi-value types just returns the value.

Syntax: `last(<field>)`

Available since: 1.2.0

Example

```
SELECT last( addresses ) FROM Account
```

---

count()

Counts the records that match the query condition. If * is not used as a field, then the record will be counted only if the field content is not null.

Syntax: `count(<field>)`

Available since: 0.9.25

Example

```
SELECT COUNT(*) FROM Account
```

---

min()

Returns the minimum value. If invoked with more than one parameter, the function doesn't aggregate but returns the minimum value between all the arguments.

Syntax: `min(<field> [, <field-n>]* )`

Available since: 0.9.25

Example

Returns the minimum salary of all the Account records:

```
SELECT min(salary) FROM Account
```

Returns the minimum value between 'salary1', 'salary2' and 'salary3' fields.

```
SELECT min(salary1, salary2, salary3) FROM Account
```

max()

Returns the maximum value. If invoked with more than one parameter, the function doesn't aggregate, but returns the maximum value between all the arguments.

Syntax: `max(<field> [, <field-n>]* )`

Available since: 0.9.25

Example

Returns the maximum salary of all the Account records:

```
SELECT max(salary) FROM Account.
```

Returns the maximum value between 'salary1', 'salary2' and 'salary3' fields.

```
SELECT max(salary1, salary2, salary3) FROM Account
```

abs()

Returns the absolute value. It works with Integer, Long, Short, Double, Float, BigInteger, BigDecimal, null.

Syntax: `abs(<field>)`

Available since: 2.2

Example

```
SELECT abs(score) FROM Account
SELECT abs(-2332) FROM Account
SELECT abs(999) FROM Account
```

avg()

Returns the average value.

Syntax: `avg(<field>)`

Available since: 0.9.25

Example

```
SELECT avg(salary) FROM Account
```

sum()

Syntax: `sum(<field>)`

Returns the sum of all the values returned.

Available since: 0.9.25

Example

```
SELECT sum(salary) FROM Account
```

date()

Returns a date formatting a string. <date-as-string> is the date in string format, and <format> is the date format following these rules. If no format is specified, then the default database format is used. To know more about it, look at Managing Dates.

Syntax: `date( <date-as-string> [<format>] [,<timezone>] )`

Available since: 0.9.25

Example

```
SELECT FROM Account WHERE created <= date('2012-07-02', 'yyyy-MM-dd')
```

sysdate()

Returns the current date time. If executed with no parameters, it returns a Date object, otherwise a string with the requested format/timezone. To know more about it, look at Managing Dates.

Syntax: `sysdate( [<format>] [,<timezone>] )`

Available since: 0.9.25

Example

```
SELECT sysdate('dd-MM-yyyy') FROM Account
```

format()

Formats a value using the String.format() conventions. Look here for more information.

Syntax: `format( <format> [,<arg1> ](,<arg-n>]*.md)`

Available since: 0.9.25

Example

```
SELECT format("%d - Mr. %s %s (%s)", id, name, surname, address) FROM Account
```

astar()

A*'s algorithm describes how to find the cheapest path from one node to another node in a directed weighted graph with husrestic function.

The first parameter is source record. The second parameter is destination record. The third parameter is a name of property that represents 'weight' and fourth represnts the map of options.

If property is not defined in edge or is null, distance between vertexes are 0 .

Syntax: `astar(<sourceVertex>, <destinationVertex>, <weightEdgeFieldName>, [<options>])`

options:

```
{
  direction:"OUT", //the edge direction (OUT, IN, BOTH)
  edgeTypeNames:[],
  vertexAxisNames:[],
  parallel : false,
  tieBreaker:true,
  maxDepth:99999,
  dFactor:1.0,
  customHeuristicFormula:'custom_Function_Name_here'  // (MANHATAN, MAXAXIS, DIAGONAL, EUCLIDEAN, EUCLIDEANNOSQR, CUSTOM)
}
```

Example

```
SELECT astar($current, #8:10, 'weight') FROM V
```

dijkstra()

Returns the cheapest path between two vertices using the [http://en.wikipedia.org/wiki/Dijkstra's_algorithm Dijkstra algorithm] where the **weightEdgeFieldName** parameter is the field containing the weight. Direction can be OUT (default), IN or BOTH.

Syntax: `dijkstra(<sourceVertex>, <destinationVertex>, <weightEdgeFieldName> [, <direction>])`

Available since: 1.3.0

Example

```
SELECT dijkstra($current, #8:10, 'weight') FROM V
```

shortestPath()

Returns the shortest path between two vertices. Direction can be OUT (default), IN or BOTH.

Available since: 1.3.0

Syntax: `shortestPath( <sourceVertex>, <destinationVertex> [, <direction> [, <edgeClassName> [, <additionalParams>]]])`

Where:

- `sourceVertex` is the source vertex where to start the path
- `destinationVertex` is the destination vertex where the path ends
- `direction` , optional, is the direction of traversing. By default is "BOTH" (in+out). Supported values are "BOTH" (incoming and outgoing), "OUT" (outgoing) and "IN" (incoming)
- `edgeClassName` , optional, is the edge class to traverse. By default all edges are crossed. Since 2.0.9 and 2.1-rc2. This can also be a list of edge class names (eg. `["edgeType1", "edgeType2"]`)
- `additionalParams` (since v 2.1.12), optional, here you can pass a map of additional parametes (Map in Java, JSON from SQL). Currently allowed parameters are
  - 'maxDepth': integer, maximum depth for paths (ignore path longer that 'maxDepth')

Example on finding the shortest path between vertices #8:32 and #8:10

```
SELECT shortestPath(#8:32, #8:10)
```

Example on finding the shortest path between vertices #8:32 and #8:10 only crossing outgoing edges

```
SELECT shortestPath(#8:32, #8:10, 'OUT')
```

Example on finding the shortest path between vertices #8:32 and #8:10 only crossing incoming edges of type 'Friend'

```
SELECT shortestPath(#8:32, #8:10, 'IN', 'Friend')
```

Example on finding the shortest path between vertices #8:32 and #8:10 only crossing incoming edges of type 'Friend' or 'Colleague'

```
SELECT shortestPath(#8:32, #8:10, 'IN', ['Friend', 'Colleague'])
```

Example on finding the shortest path between vertices #8:32 and #8:10, long at most five hops

```
SELECT shortestPath(#8:32, #8:10, null, null, {"maxDepth": 5})
```

---

distance()

Syntax: `distance( <x-field>, <y-field>, <x-value>, <y-value> )`

Returns the distance between two points in the globe using the Haversine algorithm. Coordinates must be as degrees.

Available since: 0.9.25

Example

```
SELECT FROM POI WHERE distance(x, y, 52.20472, 0.14056 ) <= 30
```

---

distinct()

Syntax: `distinct(<field>)`

Retrieves only unique data entries depending on the field you have specified as argument. The main difference compared to standard SQL DISTINCT is that with OrientDB, a function with parenthesis and only one field can be specified.

Available since: 1.0rc2

Example

```
SELECT distinct(name) FROM City
```

---

unionall()

Syntax: `unionall(<field> [,<field-n>]*)`

Works as aggregate or inline. If only one argument is passed then aggregates, otherwise executes and returns a UNION of all the collections received as parameters. Also works with no collection values.

Available since: 1.7

Example

```
SELECT unionall(friends) FROM profile
```

```
select unionall(inEdges, outEdges) from OGraphVertex where label = 'test'
```

intersect()

Syntax: `intersect(<field> [,<field-n>]*)`

Works as aggregate or inline. If only one argument is passed then it aggregates, otherwise executes and returns the INTERSECTION of the collections received as parameters.

Available since: 1.0rc2

Example

```
SELECT intersect(friends) FROM profile WHERE jobTitle = 'programmer'
```

```
SELECT intersect(inEdges, outEdges) FROM OGraphVertex
```

difference()

Syntax: `difference(<field> [,<field-n>]*)`

Works as aggregate or inline. If only one argument is passed then it aggregates, otherwise it executes and returns the DIFFERENCE between the collections received as parameters.

Available since: 1.0rc2

Example

```
SELECT difference(tags) FROM book
```

```
SELECT difference(inEdges, outEdges) FROM OGraphVertex
```

symmetricDifference()

Syntax: `symmetricDifference(<field> [,<field-n>]*)`

Works as aggregate or inline. If only one argument is passed then it aggregates, otherwise executes and returns the SYMMETRIC DIFFERENCE between the collections received as parameters.

Available since: 2.0.7

Example

```
SELECT difference(tags) FROM book
```

```
SELECT difference(inEdges, outEdges) FROM OGraphVertex
```

set()

Adds a value to a set. The first time the set is created. If `<value>` is a collection, then is merged with the set, otherwise `<value>` is added to the set.

Syntax: `set(<field>)`

Available since: 1.2.0

Example

```
SELECT name, set(roles.name) AS roles FROM OUser
```

list()

Adds a value to a list. The first time the list is created. If `<value>` is a collection, then is merged with the list, otherwise `<value>` is added to the list.

Syntax: `list(<field>)`

Available since: 1.2.0

Example

```
SELECT name, list(roles.name) AS roles FROM OUser
```

map()

Creates a map from single values.

Syntax: `map(<key>, <value>, [<key>, <value>]*)`

- With two or less parameters, it works as an aggregate function and adds a value to a map. The first time the map is created. If `<value>` is a map, then is merged with the map, otherwise the pair `<key>` and `<value>` is added to the map as new entry.
- With more than 2 params, it creates a map from single key/value pairs, eg.

```
SELECT map("name", "foo", "surname", "bar") as theMap
```

returns

```
{
   "theMap": {
       "name": "foo",
       "surname": "bar"
    }
}
```

Available since: 1.2.0

Example

```
SELECT map(name, roles.name) FROM OUser
```

traversedElement()

Returns the traversed element(s) in Traverse commands.

Syntax: `traversedElement(<index> [,<items>])`

Where:

- `<index>` is the starting item to retrieve. Value >= 0 means absolute position in the traversed stack. 0 means the first record. Negative values are counted from the end: -1 means last one, -2 means the record before last one, etc.
- `<items>`, optional, by default is 1. If >1 a collection of items is returned

Available since: 1.7

Example

Returns last traversed item of TRAVERSE command:

```
SELECT traversedElement(-1) FROM ( TRAVERSE out() FROM #34:3232 WHILE $depth <= 10 )
```

Returns last 3 traversed items of TRAVERSE command:

```
SELECT traversedElement(-1, 3) FROM ( TRAVERSE out() FROM #34:3232 WHILE $depth <= 10 )
```

traversedEdge()

Returns the traversed edge(s) in Traverse commands.

Syntax: `traversedEdge(<index> [,<items>])`

Where:

- `<index>` is the starting edge to retrieve. Value >= 0 means absolute position in the traversed stack. 0 means the first record. Negative values are counted from the end: -1 means last one, -2 means the edge before last one, etc.
- `<items>`, optional, by default is 1. If >1 a collection of edges is returned

Available since: 1.7

Example

Returns last traversed edge(s) of TRAVERSE command:

```
SELECT traversedEdge(-1) FROM ( TRAVERSE outE(), inV() FROM #34:3232 WHILE $depth <= 10 )
```

Returns last 3 traversed edge(s) of TRAVERSE command:

```
SELECT traversedEdge(-1, 3) FROM ( TRAVERSE outE(), inV() FROM #34:3232 WHILE $depth <= 10 )
```

traversedVertex()

Returns the traversed vertex(es) in Traverse commands.

Syntax: `traversedVertex(<index> [,<items>])`

Where:

- `<index>` is the starting vertex to retrieve. Value >= 0 means absolute position in the traversed stack. 0 means the first vertex. Negative values are counted from the end: -1 means last one, -2 means the vertex before last one, etc.
- `<items>`, optional, by default is 1. If >1 a collection of vertices is returned

Available since: 1.7

Example

Returns last traversed vertex of TRAVERSE command:

```
SELECT traversedVertex(-1) FROM ( TRAVERSE out() FROM #34:3232 WHILE $depth <= 10 )
```

Returns last 3 traversed vertices of TRAVERSE command:

```
SELECT traversedVertex(-1, 3) FROM ( TRAVERSE out() FROM #34:3232 WHILE $depth <= 10 )
```

mode()

Returns the values that occur with the greatest frequency. Nulls are ignored in the calculation.

Syntax: `mode(<field>)`

Available since: 2.0-M1

Example

```
SELECT mode(salary) FROM Account
```

median()

Returns the middle value or an interpolated value that represent the middle value after the values are sorted. Nulls are ignored in the calculation.

Syntax: `median(<field>)`

Available since: 2.0-M1

Example

```
select median(salary) from Account
```

percentile()

Returns the nth percentiles (the values that cut off the first n percent of the field values when it is sorted in ascending order). Nulls are ignored in the calculation.

Syntax: `percentile(<field> [, <quantile-n>]*)`

The quantiles have to be in the range 0-1

Available since: 2.0-M1

Examples

```
SELECT percentile(salary, 0.95) FROM Account
SELECT percentile(salary, 0.25, 0.75) AS IQR FROM Account
```

variance()

Returns the middle variance: the average of the squared differences from the mean. Nulls are ignored in the calculation.

Syntax: `variance(<field>)`

Available since: 2.0-M1

Example

```
SELECT variance(salary) FROM Account
```

---

stddev()

Returns the standard deviation: the measure of how spread out values are. Nulls are ignored in the calculation.

Syntax: `stddev(<field>)`

Available since: 2.0-M1

Example

```
SELECT stddev(salary) FROM Account
```

---

uuid()

Generates a UUID as a 128-bits value using the Leach-Salz variant. For more information look at: http://docs.oracle.com/javase/6/docs/api/java/util/UUID.html.

Available since: 2.0-M1

Syntax: `uuid()`

Example

Insert a new record with an automatic generated id:

```
INSERT INTO Account SET id = UUID()
```

Custom functions

The SQL engine can be extended with custom functions written with a Scripting language or via Java.

Database's function

Look at the Functions page.

Custom functions in Java

Before to use them in your queries you need to register:

```
// REGISTER 'BIGGER' FUNCTION WITH FIXED 2 PARAMETERS (MIN/MAX=2)
OSQLEngine.getInstance().registerFunction("bigger",
                                          new OSQLFunctionAbstract("bigger", 2, 2) {
  public String getSyntax() {
    return "bigger(<first>, <second>)";
  }

  public Object execute(Object[] iParameters) {
    if (iParameters[0] == null || iParameters[1] == null)
      // CHECK BOTH EXPECTED PARAMETERS
      return null;

    if (!(iParameters[0] instanceof Number) || !(iParameters[1] instanceof Number))
      // EXCLUDE IT FROM THE RESULT SET
      return null;

    // USE DOUBLE TO AVOID LOSS OF PRECISION
    final double v1 = ((Number) iParameters[0]).doubleValue();
    final double v2 = ((Number) iParameters[1]).doubleValue();

    return Math.max(v1, v2);
  }

  public boolean aggregateResults() {
    return false;
  }
});
```

Now you can execute it:

```
List<ODocument> result = database.command(
  new OSQLSynchQuery<ODocument>("SELECT FROM Account WHERE bigger( salary, 10 ) > 10") )
  .execute();
```

SQL Methods

SQL Methods are similar to SQL functions but they apply to values. In Object Oriented paradigm they are called "methods", as functions related to a class. So what's the difference between a function and a method?

This is a SQL function:

```
SELECT FROM sum( salary ) FROM employee
```

This is a SQL method:

```
SELECT FROM salary.toJSON() FROM employee
```

As you can see the method is executed against a field/value. Methods can receive parameters, like functions. You can concatenate N operators in sequence.

> **Note**: operators are case-insensitive.

Bundled methods

Methods by category

Conversions	String manipulation	Collections	Misc
convert()	append()	[]	exclude()
asBoolean()	charAt()	size()	include()
asDate()	indexOf()	remove()	javaType()
asDatetime()	left()	removeAll()	toJSON()
asDecimal()	right()	keys()	type()
asFloat()	prefix()	values()	
asInteger()	trim()		
asList()	replace()		
asLong()	length()		
asMap()	subString()		
asSet()	toLowerCase()		
asString()	toUpperCase()		
normalize()	hash()		
	format()		

Methods by name

[]	append()	asBoolean()	asDate()	asDatetime()	
asDecimal()	asFloat()	asInteger()	asList()	asLong()	asMap()
asSet()	asString()	charAt()	convert()	exclude()	format()
hash()	include()	indexOf()	javaType()	keys()	left()
length()	normalize()	prefix()	remove()	removeAll()	replace()
right()	size()	subString()	trim()	toJSON()	toLowerCase()
toUpperCase()	type()	values()			

`[]`

Execute an expression against the item. An item can be a multi-value object like a map, a list, an array or a document. For documents and maps, the item must be a string. For lists and arrays, the index is a number.

Syntax: `<value>[<expression>]`

Applies to the following types:

- document,
- map,
- list,
- array

Examples

Get the item with key "phone" in a map:

```
SELECT FROM Profile WHERE '+39' IN contacts[phone].left(3)
```

Get the first 10 tags of posts:

```
SELECT FROM tags[0-9] FROM Posts
```

History

- 1.0rc5: First version

.append()

Appends a string to another one.

Syntax: `<value>.append(<value>)`

Applies to the following types:

- string

Examples

```
SELECT name.append(' ').append(surname) FROM Employee
```

History

- 1.0rc1: First version

.asBoolean()

Transforms the field into a Boolean type. If the origin type is a string, then "true" and "false" is checked. If it's a number then 1 means TRUE while 0 means FALSE.

Syntax: `<value>.asBoolean()`

Applies to the following types:

- string,
- short,
- int,
- long

Examples

```
SELECT FROM Users WHERE online.asBoolean() = true
```

History

- 0.9.15: First version

.asDate()

Transforms the field into a Date type. To know more about it, look at Managing Dates.

Syntax: `<value>.asDate()`

Applies to the following types:

- string,
- long

Examples

Time is stored as long type measuring milliseconds since a particular day. Returns all the records where time is before the year 2010:

```
SELECT FROM Log WHERE time.asDateTime() < '01-01-2010 00:00:00'
```

History

- 0.9.14: First version

.asDateTime()

Transforms the field into a Date type but parsing also the time information. To know more about it, look at Managing Dates.

Syntax: `<value>.asDateTime()`

Applies to the following types:

- string,
- long

Examples

Time is stored as long type measuring milliseconds since a particular day. Returns all the records where time is before the year 2010:

```
SELECT FROM Log WHERE time.asDateTime() < '01-01-2010 00:00:00'
```

History

- 0.9.14: First version

.asDecimal()

Transforms the field into an Decimal type. Use Decimal type when treat currencies.

Syntax: `<value>.asDecimal()`

Applies to the following types:

- any

Examples

```
SELECT salary.asDecimal() FROM Employee
```

History

- 1.0rc1: First version

.asFloat()

Transforms the field into a float type.

Syntax: `<value>.asFloat()`

Applies to the following types:

- any

Examples

```
SELECT ray.asFloat() > 3.14
```

History

- 0.9.14: First version

.asInteger()

Transforms the field into an integer type.

Syntax: `<value>.asInteger()`

Applies to the following types:

- any

Examples

Converts the first 3 chars of 'value' field in an integer:

```
SELECT value.left(3).asInteger() FROM Log
```

History

- 0.9.14: First version

.asList()

Transforms the value in a List. If it's a single item, a new list is created.

Syntax: `<value>.asList()`

Applies to the following types:

- any

Examples

```
SELECT tags.asList() FROM Friend
```

History

- 1.0rc2: First version

.asLong()

Transforms the field into a Long type. To know more about it, look at Managing Dates.

Syntax: `<value>.asLong()`

Applies to the following types:

- any

Examples

```
SELECT date.asLong() FROM Log
```

History

- 1.0rc1: First version

.asMap()

Transforms the value in a Map where even items are the keys and odd items are values.

Syntax: `<value>.asMap()`

Applies to the following types:

- collections

Examples

```
SELECT tags.asMap() FROM Friend
```

History

- 1.0rc2: First version

.asSet()

Transforms the value in a Set. If it's a single item, a new set is created. Sets doesn't allow duplicates.

Syntax: `<value>.asSet()`

Applies to the following types:

- any

Examples

```
SELECT tags.asSet() FROM Friend
```

History

- 1.0rc2: First version

.asString()

Transforms the field into a string type.

Syntax: `<value>.asString()`

Applies to the following types:

- any

Examples

Get all the salaries with decimals:

```
SELECT salary.asString().indexof('.') > -1
```

History

- 0.9.14: First version

.charAt()

Returns the character of the string contained in the position 'position'. 'position' starts from 0 to string length.

Syntax: `<value>.charAt(<position>)`

Applies to the following types:

- string

Examples

Get the first character of the users' name:

```
SELECT FROM User WHERE name.charAt( 0 ) = 'L'
```

History

- 0.9.7: First version

.convert()

Convert a value to another type.

Syntax: `<value>.convert(<type>)`

Applies to the following types:

- any

Examples

```
SELECT dob.convert( 'date' ) FROM User
```

History

- 1.0rc2: First version

.exclude()

Excludes some properties in the resulting document.

Syntax: `<value>.exclude(<field-name>[,]*)`

Applies to the following types:

- document record

Examples

```
SELECT EXPAND( @this.exclude( 'password' ) ) FROM OUser
```

Starting from 2.2.19 you can specify a wildcard as ending character to exclude all the fields that start with a certain string. Example to exclude all the outgoing and incloming edges:

```
SELECT EXPAND( @this.exclude( 'out_*', 'in_*' ) ) FROM V
```

.format()

Returns the value formatted using the common "printf" syntax. For the complete reference goto Java Formatter JavaDoc. To know more about it, look at Managing Dates.

Syntax: `<value>.format(<format>)`

Applies to the following types:

- any

Examples

Formats salaries as number with 11 digits filling with 0 at left:

```
SELECT salary.format("%-011d") FROM Employee
```

History

- 0.9.8: First version

.hash()

Returns the hash of the field. Supports all the algorithms available in the JVM.

Syntax: `<value>` .hash([])```

Applies to the following types:

- string

Example

Get the SHA-512 of the field "password" in the class User:

```
SELECT password.hash('SHA-512') FROM User
```

History

- 1.7: First version

.include()

Include only some properties in the resulting document.

Syntax: `<value>.include(<field-name>[,]*)`

Applies to the following types:

- document record

Examples

```
SELECT EXPAND( @this.include( 'name' ) ) FROM OUser
```

Starting from 2.2.19 you can specify a wildcard as ending character to include all the fields that start with a certain string. Example to include all the fields that starts with `amount` :

```
SELECT EXPAND( @this.include( 'amount*' ) ) FROM OUser
```

History

- 1.0rc2: First version

.indexOf()

Returns the position of the 'string-to-search' inside the value. It returns -1 if no occurrences are found. 'begin-position' is the optional position where to start, otherwise the beginning of the string is taken (=0).

Syntax: `<value>.indexOf(<string-to-search> [, <begin-position>)`

Applies to the following types:

- string

Examples

Returns all the UK numbers:

```
SELECT FROM Contact WHERE phone.indexOf('+44') > -1
```

History

- 0.9.10: First version

.javaType()

Returns the corresponding Java Type.

Syntax: `<value>.javaType()`

Applies to the following types:

- any

Examples

Prints the Java type used to store dates:

```
SELECT FROM date.javaType() FROM Events
```

History

- 1.0rc1: First version

.keys()

Returns the map's keys as a separate set. Useful to use in conjunction with IN, CONTAINS and CONTAINSALL operators.

Syntax: `<value>.keys()`

Applies to the following types:

- maps
- documents

Examples

```
SELECT FROM Actor WHERE 'Luke' IN map.keys()
```

History

- 1.0rc1: First version

.left()

Returns a substring of the original cutting from the begin and getting 'len' characters.

Syntax: `<value>.left(<length>)`

Applies to the following types:

- string

Examples

```
SELECT FROM Actors WHERE name.left( 4 ) = 'Luke'
```

History

- 0.9.7: First version

.length()

Returns the length of the string. If the string is null 0 will be returned.

Syntax: `<value>.length()`

Applies to the following types:

- string

Examples

```
SELECT FROM Providers WHERE name.length() > 0
```

History

- 0.9.7: First version

.normalize()

Form can be NDF, NFD, NFKC, NFKD. Default is NDF. pattern-matching if not defined is "\p{InCombiningDiacriticalMarks}+". For more information look at Unicode Standard.

Syntax: `<value>.normalize( [<form>] [,<pattern-matching>] )`

Applies to the following types:

- string

Examples

```
SELECT FROM V WHERE name.normalize() AND name.normalize('NFD')
```

History

- 1.4.0: First version

.prefix()

Prefixes a string to another one.

Syntax: `<value>.prefix('<string>')`

Applies to the following types:

- string

Examples

```
SELECT name.prefix('Mr. ') FROM Profile
```

History

- 1.0rc1: First version

.remove()

Removes the first occurrence of the passed items.

Syntax: `<value>.remove(<item>*)`

Applies to the following types:

- collection

Examples

```
SELECT out().in().remove( @this ) FROM V
```

History

- 1.0rc1: First version

.removeAll()

Removes all the occurrences of the passed items.

Syntax: `<value>.removeAll(<item>*)`

Applies to the following types:

- collection

Examples

```
SELECT out().in().removeAll( @this ) FROM V
```

History

- 1.0rc1: First version

.replace()

Replace a string with another one.

Syntax: `<value>.replace(<to-find>, <to-replace>)`

Applies to the following types:

- string

Examples

```
SELECT name.replace('Mr.', 'Ms.') FROM User
```

History

- 1.0rc1: First version

.right()

Returns a substring of the original cutting from the end of the string 'length' characters.

Syntax: `<value>.right(<length>)`

Applies to the following types:

- string

Examples

Returns all the vertices where the name ends by "ke".

```
SELECT FROM V WHERE name.right( 2 ) = 'ke'
```

History

- 0.9.7: First version

.size()

Returns the size of the collection.

Syntax: `<value>.size()`

Applies to the following types:

- collection

Examples

Returns all the items in a tree with children:

```
SELECT FROM TreeItem WHERE children.size() > 0
```

History

- 0.9.7: First version

.subString()

Returns a substring of the original cutting from 'begin' index up to 'end' index (not included).

Syntax: `<value>.subString(<begin> [,<end>] )`

Applies to the following types:

- string

Examples

Get all the items where the name begins with an "L":

```
SELECT name.substring( 0, 1 ) = 'L' FROM StockItems
```

Substring of `OrientDB`

```
SELECT "OrientDB".substring(0,6)
```

returns `Orient`

History

- 0.9.7: First version

.trim()

Returns the original string removing white spaces from the begin and the end.

Syntax: `<value>.trim()`

Applies to the following types:

- string

Examples

```
SELECT name.trim() == 'Luke' FROM Actors
```

History

- 0.9.7: First version

.toJSON()

Returns the record in JSON format.

Syntax: `<value>.toJSON([<format>])`

Where:

- **format** optional, allows custom formatting rules (separate multiple options by comma). Rules are the following:
  - **type** to include the fields' types in the "@fieldTypes" attribute
  - **rid** to include records's RIDs as attribute "@rid"
  - **version** to include records' versions in the attribute "@version"
  - **class** to include the class name in the attribute "@class"
  - **attribSameRow** put all the attributes in the same row
  - **indent** is the indent level as integer. By Default no ident is used
  - **fetchPlan** is the FetchPlan to use while fetching linked records
  - **alwaysFetchEmbedded** to always fetch embedded records (without considering the fetch plan)
  - **dateAsLong** to return dates (Date and Datetime types) as long numers
  - **prettyPrint** indent the returning JSON in readeable (pretty) way

Applies to the following types:

- record

Examples

```
create class Test extends V
insert into Test content {"attr1": "value 1", "attr2": "value 2"}

select @this.toJson('rid,version,fetchPlan:in_*:-2 out_*:-2') from Test
```

History

- 0.9.8: First version

.toLowerCase()

Returns the string in lower case.

Syntax: `<value>.toLowerCase()`

Applies to the following types:

- string

Examples

```
SELECT name.toLowerCase() == 'luke' FROM Actors
```

History

- 0.9.7: First version

.toUpperCase()

Returns the string in upper case.

Syntax: `<value>.toUpperCase()`

Applies to the following types:

- string

Examples

```
SELECT name.toUpperCase() == 'LUKE' FROM Actors
```

History

- 0.9.7: First version

.type()

Returns the value's OrientDB Type.

Syntax: `<value>.type()`

Applies to the following types:

- any

Examples

Prints the type used to store dates:

```
SELECT FROM date.type() FROM Events
```

History

- 1.0rc1: First version

.values()

Returns the map's values as a separate collection. Useful to use in conjunction with IN, CONTAINS and CONTAINSALL operators.

Syntax: `<value>.values()`

Applies to the following types:

- maps
- documents

Examples

```
SELECT FROM Clients WHERE map.values() CONTAINSALL ( name is not null)
```

History

- 1.0rc1: First version

SQL Batch

OrientDB allows execution of arbitrary scripts written in Javascript or any scripting language installed in the JVM. OrientDB supports a minimal SQL engine to allow a batch of commands.

Batch of commands are very useful when you have to execute multiple things at the server side avoiding the network roundtrip for each command.

SQL Batch supports all the OrientDB SQL commands, plus the following:

- `begin [isolation <isolation-level>]`, where `<isolation-level>` can be `READ_COMMITTED`, `REPEATABLE_READ`. By default is `READ_COMMITTED`
- `commit [retry <retry>]`, where:
    - is the number of retries in case of concurrent modification exception
- `let <variable> = <SQL>`, to assign the result of a SQL command to a variable. To reuse the variable prefix it with the dollar sign $
- `if(<expression>){<statememt>}`. Look at Conditional execution.
- `sleep <ms>`, put the batch in wait for `<ms>` milliseconds.
- `console.log <text>`, logs a message in the console. Context variables can be used with `${<variable>}`. Since 2.2.
- `console.error <text>`, writes a message in the console's standard output. Context variables can be used with `${<variable>}`. Since 2.2.
- `console.output <text>`, writes a message in the console's standard error. Context variables can be used with `${<variable>}`. Since 2.2.
- `return`, where value can be:
    - any value. Example: `return 3`
    - any variable with $ as prefix. Example: `return $a`
    - arrays (HTTP protocol only, see below). Example: `return [ $a, $b ]`
    - maps (HTTP protocol only, see below). Example: `return { 'first' : $a, 'second' : $b }`
    - a query. Example: `return (SELECT FROM Foo)`

  NOTE: to return arrays and maps (eg. Java or Node.js driver) it's strongly recommended to use a RETURN SELECT, eg.

```
return (SELECT $a as first, $b as second)
```

This will work on any protocol and driver.

See also

- Javascript-Command

Optimistic transaction

Example to create a new vertex in a Transaction and attach it to an existent vertex by creating a new edge between them. If a concurrent modification occurs, repeat the transaction up to 100 times:

```
begin
let account = create vertex Account set name = 'Luke'
let city = select from City where name = 'London'
let e = create edge Lives from $account to $city
commit retry 100
return $e
```

Note the usage of $account and $city in further SQL commands.

Pessimistic transaction

This script above used an Optimistic approach: in case of conflict it retries up top 100 times by re-executing the entire transaction (commit retry 100). To follow a Pessimistic approach by locking the records, try this:

```
BEGIN
let account = CREATE VERTEX Account SET name = 'Luke'
let city = SELECT FROM City WHERE name = 'London' LOCK RECORD
let e = CREATE EDGE Lives FROM $account TO $city
COMMIT
return $e
```

Note the "lock record" after the select. This means the returning records will be locked until commit (or rollback). In this way concurrent updates against London will wait for this transaction to complete.

NOTE: locks inside transactions works ONLY against MEMORY storage, we're working to provide such feature also against plocal. Stay tuned (Issue https://github.com/orientechnologies/orientdb/issues/1677)

Conditional execution

(since 2.1.7) SQL Batch provides IF constructor to allow conditional execution. The syntax is

```
if(<sql-predicate>){
   <statement>
   <statement>
   ...
}
```

`<sql-predicate>` is any valid SQL predicate (any condition that can be used in a WHERE clause). In current release it's mandatory to have `IF(){`, `<statement>` and `}` on separate lines, eg. the following is not a valid script

```
if($a.size() > 0) { ROLLBACK }
```

The right syntax is following:

```
if($a.size() > 0) {
   ROLLBACK
}
```

Java API

This can be used by Java API with:

```
database.open("admin", "admin");

String cmd = "begin\n";
cmd += "let a = CREATE VERTEX SET script = true\n";
cmd += "let b = SELECT FROM v LIMIT 1\n";
cmd += "let e = CREATE EDGE FROM $a TO $b\n";
cmd += "COMMIT RETRY 100\n";
cmd += "return $e";

OIdentifiable edge = database.command(new OCommandScript("sql", cmd)).execute();
```

Remember to put one command per line (postfix it with \n) or use the semicolon (;) as separator.

HTTP REST API

And via HTTP REST interface (https://github.com/orientechnologies/orientdb/issues/2056). Execute a POST against /batch URL by sending a payload in this format:

```
{ "transaction" : false,
  "operations" : [
    {
      "type" : "script",
      "language" : "sql",
      "script" : <text>
    }
  ]
}
```

Example:

```
{ "transaction" : false,
  "operations" : [
    {
      "type" : "script",
      "language" : "sql",
      "script" : [ "BEGIN;let account = CREATE VERTEX Account SET name = 'Luke';let city =SELECT FROM City WHERE name = 'Londo
n';CREATE EDGE Lives FROM $account TO $city;COMMIT RETRY 100" ]
    }
  ]
}
```

To separate commands use semicolon (;) or linefeed (\n). Starting from release 1.7 the "script" property can be an array of strings to put each command on separate item, example:

```
{ "transaction" : false,
  "operations" : [
    {
      "type" : "script",
      "language" : "sql",
      "script" : [ "begin",
                   "let account = CREATE VERTEX Account SET name = 'Luke'",
                   "let city = SELECT FROM City WHERE name = 'London'",
                   "CREATE EDGE Lives FROM $account TO $city",
                   "COMMIT RETRY 100" ]
    }
  ]
}
```

Hope this new feature will simplify your development improving performance.

What about having more complex constructs like IF, FOR, etc? If you need more complexity, we suggest you to use Javascript as language that already support all these concepts.

Pagination

OrientDB supports pagination natively. Pagination doesn't consume server side resources because no cursors are used. Only Record ID's are used as pointers to the physical position in the cluster.

There are 2 ways to achieve pagination:

Use the SKIP-LIMIT

The first and simpler way to do pagination is to use the `SKIP` / `LIMIT` approach. This is the slower way because OrientDB repeats the query and just skips the first X records from the result. Syntax:

```
SELECT FROM <target> [WHERE ...] SKIP <records-to-skip> LIMIT <max-records>
```

Where:

- **records-to-skip** is the number of records to skip before starting to collect them as the result set
- **max-records** is the maximum number of records returned by the query

Example

Use the RID-LIMIT

This method is faster than the `SKIP` - `LIMIT` because OrientDB will begin the scan from the starting RID. OrientDB can seek the first record in about O(1) time. The downside is that it's more complex to use.

The trick here is to execute the query multiple times setting the `LIMIT` as the page size and using the greater than `>` operator against `@rid` . The **lower-rid** is the starting point to search, for example `#10:300` .

Syntax:

```
SELECT FROM <target> WHERE @rid > <lower-rid> ... [LIMIT <max-records>]
```

Where:

- **lower-rid** is the exclusive lower bound of the range as Record ID
- **max-records** is the maximum number of records returned by the query

In this way, OrientDB will start to scan the cluster from the given position **lower-rid** + 1. After the first call, the **lower-rid** will be the rid of the last record returned by the previous call. To scan the cluster from the beginning, use `#-1:-1` as **lower-rid** .

Handle it by hand

```
database.open("admin", "admin");
final OSQLSynchQuery<ODocument> query = new OSQLSynchQuery<ODocument>("select from Customer where @rid > ? LIMIT 20");

List<ODocument> resultset = database.query(query, new ORecordId());

while (!resultset.isEmpty()) {
    ORID last = resultset.get(resultset.size() - 1).getIdentity();

    for (ODocument record : resultset) {
        // ITERATE THE PAGINATED RESULT SET
    }

    resultset = database.query(query, last);
}
database.close();
```

Automatic management

In order to simplify the pagination, the `OSQLSynchQuery` object (usually used in queries) keeps track of the current page and, if executed multiple times, it advances page to page automatically without using the `>` operator.

Example:

```
OSQLSynchQuery<ODocument> query = new OSQLSynchQuery<ODocument>("select from Customer LIMIT 20");
for (List<ODocument> resultset = database.query(query); !resultset.isEmpty(); resultset = database.query(query)) {
    ...
}
```

Usage of indexes

This is the faster way to achieve pagination with large clusters.

If you've defined an index, you can use it to paginate results. An example is to get all the names next to `Jay` limiting it to 20:

```
Collection<ODocument> indexEntries = (Collection<ODocument>) index.getEntriesMajor("Jay", true, 20);
```

Sequences and auto increment

Starting from v2.2, OrientDB supports sequences like most of RDBMS. What's a sequence? It's a structure that manage counters. Sequences are mostly used when you need a number that always increments. Sequence types can be:

- **ORDERED**: each call to `.next()` will result in a new value.
- **CACHED**: the sequence will cache N items on each node, thus improving the performance if many `.next()` calls are required. However, this may create holes.

To manipulate sequences you can use the Java API or SQL commands.

Create a sequence

Create a sequence with Java API

```
OSequenceLibrary sequenceLibrary = database.getMetadata().getSequenceLibrary();
OSequence seq = sequenceLibrary.createSequence("idseq", SEQUENCE_TYPE.ORDERED, new OSequence.CreateParams().setStart(0).setInc
rement(1));
```

SQL CREATE SEQUENCE

```
CREATE SEQUENCE idseq
INSERT INTO account SET id = sequence('idseq').next()
```

For more information look at SQL CREATE SEQUENCE.

Using a sequence

Using a sequence with Java API

```
OSequence seq = graph.getRawGraph().getMetadata().getSequenceLibrary().getSequence("idseq");
graph.addVertex("class:Account", "id", seq.next());
```

Using a sequence from SQL

You can use a sequence from SQL with the following syntax:

```
sequence('<sequence>').<method>
```

Where:

- `method` can be:
  - `next()` retrieves the next value
  - `current()` gets the current value
  - `reset()` resets the sequence value to it's initial value

Example

```
INSERT INTO Account SET id = sequence('mysequence').next()
```

Alter a sequence

Alter a sequence with Java API

```
graph.getRawGraph().getMetadata().getSequenceLibrary().getSequence("idseq").updateParams( new OSequence.CreateParams().setStar
t(1000) );
```

SQL ALTER SEQUENCE

```
ALTER SEQUENCE idseq START 1000
```

For more information look at SQL ALTER SEQUENCE.

Drop a sequence

Drop a sequence with Java API

```
graph.getRawGraph().getMetadata().getSequenceLibrary().dropSequence("idseq");
```

SQL DROP SEQUENCE

```
DROP SEQUENCE idseq
```

For more information look at SQL DROP SEQUENCE.

OrientDB before v2.2

OrientDB before v2.2 doesn't support sequences (autoincrement), so you can manage your own counter in this way (example using SQL):

```
CREATE CLASS counter
INSERT INTO counter SET name='mycounter', value=0
```

And then every time you need a new number you can do:

```
UPDATE counter INCREMENT value = 1 WHERE name = 'mycounter'
```

This works in a SQL batch in this way:

```
BEGIN
let $counter = UPDATE counter INCREMENT value = 1 return after $current WHERE name = 'mycounter'
INSERT INTO items SET id = $counter.value[0], qty = 10, price = 1000
COMMIT
```

When planning an OrientDB SELECT query, it is important to determine the model class that will be used as the pivot class of the query. This class is expressed in the FROM clause. It affects other elements in the query as follows:

- projections will be relative to the pivot class. It is possible to traverse within a projection to refer to neighboring classes by chaining edge syntax expressions (i.e. `in[label='office'].out.out[label='office'].size()`). However, consider that multiple results from a projection traversed from the pivot class will be returned as a collection within the result set (unless there is only a single value).
- filtering conditions in the WHERE clause are also relative to the pivot class. It is also possible to traverse to neighboring classes in order to compose advanced conditions by using edge syntax expressions (e.g. `and in[label='company'].out.out[label='employee'].in.id IN '0000345'`).
- the ORDER BY clause will be relative to one of the projections and must be returned as a single value per record (i.e. an attribute of the pivot class or a single attribute of a neighboring class). It will not be possible to order by traversed projections in a single query if they return multiple results (as a collection). Therefore, in queries using an ORDER BY clause, there is only one possible choice for the pivot class as it must be the class containing the attribute to order by.

Additionally, there are performance considerations that should be considered on selecting the pivot class. Assuming 2 classes as follows:

```
+--------------------+           +-------------------+
| Class: CountryType | --------> | Class: PersonType |
| attr: name         |           |  attr: name       |
| atr: code          |           |  attr: lat        |
|                    |           |  attr: long       |
+--------------------+           +-------------------+
  (tens of vertices)             (millions of vertices)
```

Queries:

```
SELECT [...] FROM CountryType WHERE [...]

SELECT [...] FROM PersonType WHERE [...]
```

The first query will apply the WHERE filtering and projections to fewer vertices, and as a result will perform faster that the second query. Therefore, it is advisable to assign the pivot class to the class that contains the most relevant items for the query to avoid unnecessary loops from the evaluation, i.e. usually the one with lower multiplicity.

Switching the pivot class within a query

Based on the previous discussion, there may be conflicting requirements on determining the pivot class. Take the case where we need to ORDER BY a class with a very high multiplicity (say, millions of vertices), but most of these vertices are not relevant for the outcome of our query.

On one hand, according to the requirements of the ORDER BY clause, we are forced to choose the class containing the attribute to order by as the pivot class. But, as we also saw, this class can not be an optimal choice from a performance point of view if only a small subset of vertices is relevant to the query. In this case, we have a choice between poor performance resulting from setting the pivot class as the class containing the attribute to order by even though it has a higher multiplicity, or good performance by taking out the ORDER BY clause and ordering results after the fact in the invoking Java code, which is more work. If we choose to execute the full operation in one query, indices can be used to improve the poor performance, but it would be usually an overkill as a consequence of a bad query planning.

A more elegant solution can be achieved by using the nested query technique, as shown below:

```
SELECT                                                                  -- outer query
  in[label='city'].out.name AS name,
  in[label='city'].out.out[label='city'].size() AS city_count,
  CityLat,
  CityLong,
  distance(CityLat, CityLong, 51.513363, -0.089178) AS distance     -- order by parameter
FROM (                                                                  -- inner query
  SELECT flatten( in[label='region'].out.out[label='city'].in )
  FROM CountryType WHERE id IN '0032'
)
WHERE CityLat <> '' AND CityLong  <> ''
ORDER BY distance
```

This nested query represents a two-fold operation, taking the best of both worlds. The inner query uses the `CountryType` class which has lower multiplicity as pivot class, so the number of required loops is smaller, and as a result delivers better performance. The set of vertices resulting from the inner query is taken as pivot class for the outer query. The `flatten()` function is required to expose items from the inner query as a flat structure to the outer query. The higher the multiplicity and number of irrelevant records in the class with the parameter to order by, the more convenient using this approach becomes.

Command Cache

Starting in release 2.2, OrientDB supports caching of command results. Caching command results has been used by other DBMSs and has proven to improve dramatically the following use cases:

- database is mostly read than write
- there are a few heavy queries that return a small result set
- you have available RAM to use for caching results

By default, the command cache is disabled. To enable it, set `command.cache.enabled=true`. Look at the Studio page about Command Cache.

Settings

There are some settings to tune the command cache. See the table below containing all the available settings.

Parameter	Description	Type	Default value
command.cache.enabled	Enable command cache	Boolean	false
command.cache.evictStrategy	Command cache strategy between: [INVALIDATE_ALL,PER_CLUSTER]	String.class	PER_CLUSTER
command.cache.minExecutionTime	Minimum execution time to consider caching result set	Integer.class	10
command.cache.maxResultsetSize	Maximum resultset time to consider caching result set	Integer	500

Eviction strategies

Using a cache that holds old data could be meaningless, unless you can accept eventual consistency. For this reason, the command cache supports 2 eviction strategies to keep the cache consistent:

- **INVALIDATE_ALL** to remove all the query results at every Create, Update, and Delete operation. This is faster than **PER_CLUSTER** if many writes occur.
- **PER_CLUSTER** to remove all the query results only related to the modified cluster. This operation is more expensive than **INVALIDATE_ALL**.

SQL Query Optimization

> **IMPORTANT**: This section refers to OrientDB v 2.2 only.
>
> Some of these tips are also valid for previous 2.x versions.
>
> V 3.0 has a completely new execution planner, so none of these tips can be considered valid on that version.

The SQL executor is a quite complex component and one of the oldest pieces in the architecture of OrientDB, writing efficient queries requires some knowledge of the internals and of the related components, like indexes and clusters. This said, following some basic guidelines allows to reach a good level of performance for most of the typical SQL queries. This section is intended to provide these guidelines and to help end users to write efficient SQL queries in OrientDB. We will concentrate on SQL `SELECT` statement, but most of these guidelines also apply to other statements.

Case A: Anatomy of an SQL query

Let's start from a basic but complete use case:

```
SELECT name, surname
from Person
WHERE age = 25
ORDER BY name ASC
SKIP 10 LIMIT 10
```

Let's start from the base situation:

- 1.000.000 Person records in the DB
- evenly distributed by age (0 - 99)
- no indexes defined

For this simple statement, the query executor will perform the following actions:

- fetch records from `Person` class
- for each record, filter by `age`
- calculate projectionis (`name` and `surname`)
- order the resulting records based on `name` property
- skip the first 10 records
- return 10 records

> NOTE: the ordering of the result is executed *after* the calculation of the projections, so the result can only be sorted by `name` or `surname` , ie. not by `age`

Some facts:

- the SQL executor will scan the whole Person class to fetch all the records that match the condition...
- ...even if only 10.000 of them will match the condition `age = 25`
- the ORDER BY takes into consideration the SKIP/LIMIT, so it keeps in memory only 20 records with the "smallest" name (then the SKIP will discard the first 10)
- the sorting is executed in HEAP memory

The most relevant operation here is the scan of 1.000.000 records (likely loading them from disk)

Case B: Index based filtering

Based on Case A, the first optimization we can do here is adding an index on `age`

```
CREATE INDEX Person.age on Person (age) NOTUNIQUE
```

With this simple optimization, the query execution plan becomes as follows:

- fetch records from `Person.age` index, where key = 25
- calculate projectionis (`name` and `surname`)
- order the resulting records based on `name` property
- skip the first 10 records
- return 10 records

Some facts:

- in this case the SQL executor will only fetch from disk the 10.000 records that match `age = 25`
- the ORDER BY still only keeps in memory 20 records with the "smallest" name (then the SKIP will discard the first 10)

Compared to Case A, OrientDB only fetches from disk 1/100 of the records, so you can expect the original query to be 100 times faster approximately.

Case C: Index based sorting

Based on Case A (suppose no other indexes are available, ie. we never performed the optimization provided with Case B), another optimization we can do is adding an index on `name` property to speed up the sorting operation.

```
CREATE INDEX Person.name on Person (name) NOTUNIQUE
```

With this optimization, the query execution plan becomes as follows:

- fetch records from `Person.name` index in ascending order
- calculate projectionis (`name` and `surname`)
- for each record, filter by `age`
- collect the first 20 valid results
- discard 10 records (SKIP)
- return 10 records

Some facts:

- There is no need for sorting here, as the index already provides records sorted by `name`
- If you are lucky enough, the first 20 records will have `age = 20` , so the query will take ~1/50.000 of the original one...
- ...but if you are particularly unlucky, to find 20 records that match `age = 20` you will have to scan all the original dataset, so the performance will be the same as the Case A

> **IMPORTANT:** Sorting based on indexes can be **only** performed on tree-based indexes (ie. UNIQUE and NOTUNIQUE indexes). All the other types of indexes (eg. NOTUNIQUE_HASH_INDEX, UNIQUE_HASH_INDEX, LUCENE) *do not* support sorting, so they will be ignored for ORDER BY operations.

Case D: Index based filtering + sorting

Let's try to mix Case B and C together and see if we can do better:

The naive approach of using both indexes together won't work:

```
//WRONG!

CREATE INDEX Person.age on Person (age) NOTUNIQUE
CREATE INDEX Person.name on Person (name) NOTUNIQUE
```

With these index definitions, OrientDB will be able to use only one index to optimize the query. In this case it will choose the index for filtering and will discard the other one.

> **IMPORTANT**: THE EXECUTION PLAN **CANNOT** MIX INDEXES FOR SORTING AND FILTERING. IT WILL ALWAYS CHOOSE THE INDEX FOR FILTERING AND WILL IGNORE THE OTHER ONE.

OrientDB can actually exploit indexes for both filtering and sorting, but it has to be the *same* index:

```
//CORRECT

CREATE INDEX Person.age_name on Person (age, name) NOTUNIQUE
```

With this index, the query execution plan becomes much more efficient:

- fetch records from `Person.age_name` index in ascending order, `where age = 25`
- discard 10 records (SKIP)
- return 10 records

This execution will ALWAYS fetch only 20 records from the storage, so the query performance is always 50.000x faster than Case A

Case E: Equality and Inequality conditions

Let's consider three different statements:

```
SELECT FROM Person WHERE age = 25

SELECT FROM Person WHERE age <> 25

SELECT FROM Person WHERE age > 25
```

The first statement has an equality expression; to execute it, OrientDB can use **any type** of index (apart from fulltext and spatial), ie. tree based and hash indexes.

The second statement has a "not equals" condition. OrientDB will *never* use indexes to optimize it. `<>` is equivalent to `!=`

The third statement has a "range" condition (range operators include `>`, `<`, `>=`, `<=`); OrientDB can only use three-based indexes (ie. UNIQUE and NOTUNIQUE) to optimize range queries. Hash indexes will be ignored.

Case F: Composite indexes - full match

A composite index is an index defined on multiple properties. Consider the following

```
CREATE CLASS Person
CREATE PROPERTY Person.name STRING
CREATE PROPERTY Person.surname STRING
CREATE PROPERTY Person.age INTEGER
CREATE PROPERTY Person.karma INTEGER
```

And an index defined as follows:

```
CREATE INDEX Person.name_surname_age_karma on Person (name, surname, age, karma) NOTUNIQUE
```

This index can of course be used for a full match, eg.

```
SELECT FROM Person WHERE name = 'foo' AND surname = 'bar' AND age = 25 AND karma = 100
```

Case G - Composite indexes - partial match

Consider the schema and the index defined in Case E. This index can also be used for partial queries, eg. the following queries can use that index to optimize the search

```
SELECT FROM Person WHERE name = 'foo' AND surname = 'bar' AND age = 25

SELECT FROM Person WHERE name = 'foo' AND surname = 'bar'

SELECT FROM Person WHERE name = 'foo'
```

The partial match is allowed only on a prefix of the index definition. The following query **won't** be optimized by the above mentioned index:

```
//NOT INDEXED
SELECT FROM Person WHERE surname = 'bar' AND age = 25 AND karma = 100

//NOT INDEXED
SELECT FROM Person WHERE age = 25

//NOT INDEXED
SELECT FROM Person WHERE karma = 100
```

> IMPORTANT: Only **tree-based** indexes (ie UNIQUE, NOTUNIQUE) can be used for partial match. Hash indexes (eg. UNIQUE_HASH_INDEX, NOTUNIQUE_HASH_INDEX) will be **ignored** for partial match.

Case H - Composite indexes - range queries

Tree-based indexes can be used to optimize both equality and range queries. The same applies to composite indexes, with the only limitation that the range condition has to be on the last property that is used for index search. Let's make it clear with an example:

Given the schema and the index defined in Case E, consider the following query:

```
SELECT FROM Person WHERE name = 'foo' AND surname = 'bar' AND age = 25 AND karma > 100
```

This query will be executed using the full index (ie. on properties `name` , `surname` , `age` and `karma`).

Now consider the following:

```
SELECT FROM Person WHERE name = 'foo' AND surname = 'bar' AND age > 25 AND karma > 100
```

now the range condition is on `age` , that is the third property in the index definition. In this case, the query will be executed as follows:

- fetch from the index, based on `name` , `surname` and `age`
- filter the resulting records by `karma`

So if you have 1000 records with the same name, surname and age, but only one has karma > 100, the query will fetch all the 1000 records and filter them one by one, based on `karma` value.

This happens because now the first range condition is `age > 25` , this condition *short-circuits the range query*

The same would have happened if the condition on `karma` was an equality condition (ie. `karma = 100`); all the conditioins after the first range condition are ignored in partial index match.

> **IMPORTANT:** range conditions short-circuit partial index usage

Case I - Composite indexes - partial match and sorting

As discussed in Case D, indexes can be used for filtering and sorting at the same time. This also applies to partial match. Consider the domain and the index defined in Case E and the following query:

```
SELECT FROM Person WHERE name = 'foo' AND surname = 'bar' ORDER BY age
```

In this case the query executor will use the index for both filtering (partial match on `name` and `surname`) and for sorting.

The conditions for this to happen are following:

- the filtering has to be done based on equality conditions (ie. no range conditions)
- the sorting has to be executed on a property that, in the index definition, is right next to the properties used for filtering

To make it clear, consider this scheme:

```
CREATE INDEX theIndex on TheClass (prop1, prop2... propN) NOTUNIQUE
```

```
SELECT FROM TheClass
WHERE
prop1 = ...
AND prop2 = ...
...
AND propX = ...
ORDER BY `propX+1`
```

```
          ALL EQUALITY CONDITIONS            NOTHING IN
                    |                        THE MIDDLE
 +------------------+--------------------+     |
 |                                       |     |
equality    equality    equality    equality   |
condition   condition   condition   condition  |  ORDER BY
    |           |           |           |      |    |
    V           V           V           V      |    V
  prop1       prop2       ...         propX    V  propX+1     ....   propN
```

> **IMPORTANT:** both partial match and sorting are allowed only on tree-based indexes

Case J - IN condition

Consider a query like this:

```
SELECT FROM Person WHERE name in ['foo', 'bar']
```

This query can be optimized with an index that is defined on `name` property only (eg. not on an index that is defined on `name` and `surname`).

> this is not a limitation in the index engine, but just a limitation in the implementation of the SQL executor, there is a chance that in next 2.2.x releases it will be addressed. This limitation does not exist in V 3.0.

The same applies to a query on a composite index, eg.

```
SELECT FROM Person WHERE name = 'foo' AND surname in ['xxx', 'yyy']
```

This query can be optimized using an index defined on `name` and `surname` .

As a general rule, `IN` conditions are optimized using indexes only when all these conditions apply:

- the index can be used for a full match (not on partial match)
- the `IN` condition is defined on the last property in the index definition

In all the other cases, the `IN` condition is not optimized using indexes

Case K - order of conditions in the WHERE clause

In v 2.2, OrientDB SQL executor tries to find the best index based on the conditions defined in the query, but in some cases if fails to find the right combination of conditions to consider for indexed execution.

An important rule to make OrientDB find the right index (and use it the right way) is to write the conditions in the same order as the properties appear in the index definition.

Consider the schema and index defined in Case E, a query like following

```
SELECT FROM Person WHERE name = 'foo' AND surname = 'bar' AND age = 25
```

Will correctly use the index on all the three properties. A query like following

```
//wrong order of properties
SELECT FROM Person WHERE surname = 'bar' AND age = 25 AND name = 'foo'
```

in some cases will only use the index to match the `name` and then will manually filter on `surname` and `age`

This is particularly relevant when using parentheses, eg. the following query

```
//wrong order of properties + parentheses
SELECT FROM Person WHERE (surname = 'bar' AND age = 25) AND (name = 'foo')
```

will likely fail to correctly use the index

> **IMPORTANT**: always try to write your WHERE clause so that the order of the conditions matches the order of the fields in the index definition, this will make it easier for OrientDB to find the right index and use it correctly.
>
> **NOTE**: this limitation is completely removed in v 3.0

Case L - AND vs OR conditions

Consider the following schema:

```
CREATE CLASS TheClass
CREATE PROPERTY TheClass.prop1 STRING
CREATE PROPERTY TheClass.prop2 STRING

CREATE INDEX TheClass.prop1 on TheClass (prop1) NOTUNIQUE
CREATE INDEX TheClass.prop2 on TheClass (prop2) NOTUNIQUE
```

In such a scenario, you can write queries that involve both `prop1` and `prop2` and OrientDB will have to choose which index (or indexes) to use.

A simple case is following:

```
SELECT FROM TheClass WHERE prop1 = 'foo' OR prop2 = 'bar'
```

in this case, OrientDB will use both indexes and will merge the results.

> OrientDB can use mutiple indexes when OR conditions are involved

Another possible case is this (please note that in this case an `AND` condition is involved):

```
SELECT FROM TheClass WHERE prop1 = 'foo' AND prop2 = 'bar'
```

in this case OrientDB will use *only one index* (eg. the one defined on `prop1`) and then will apply the rest of the conditions record by record.

The way OrientDB chooses the index depends on how many properties are involved in the indexing. Typically, OrientDB will *try* to use as many properties as possible for indexed query. When two indexes with the same number of properties are available, the choice is simply based on internals (eg. on the order the indexes appear in the schema).

As a general rule, you should not rely on this mechanism, because the general performance of the query is not completely predictable.

A better approach is to re-write the query as a chain of sub-queries, to make sure that he inner query uses the right index. Eg. if you want OrientDB to use the index on `prop2` , you can write the query as follows:

```
SELECT FROM (
   SELECT FROM TheClass WHERE prop2 = 'bar'
) WHERE prop1 = 'foo'
```

Case M - Optimization of count(*)

OrientDB can optimize a `count(*)` in some basic cases:

- when it is performed on a class or on a cluster, without further conditions, eg.

```
SELECT count(*) FROM Person
```

- when it's performed on an indexed query without further filtering:

```
--suppose an index is defined on "name"
SELECT count(*) FROM Person where name = 'a'
```

Understanding EXPLAIN command

EXPLAIN is a very useful tool to understand how a query is performing. To use it, just prefix the query with `explain` keyword, eg.

```
EXPLAIN SELECT FROM Person WHERE name = 'foo'
```

The result is a record containing statistics about the query execution, eg.

```
{
    "result": [
        {
            "@type": "d",
            "@version": 0,
            "documentReads": 2,
            "fullySortedByIndex": false,
            "documentAnalyzedCompatibleClass": 2,
            "recordReads": 2,
            "fetchingFromTargetElapsed": 0,
            "indexIsUsedInOrderBy": false,
            "compositeIndexUsed": 1,
            "current": "#74:0",
            "involvedIndexes": [
                "Person.name_surname_age_karma"
            ],
            "limit": -1,
            "evaluated": 2,
            "user": "#5:0",
            "elapsed": 0.655,
            "resultType": "collection",
            "resultSize": 1
        }
    ]
}
```

> NOTE: In v 2.2, OrientDB will have to execute the query to calculate the `explain`, so if the query takes a lot of time/memory/resources to execute, the `explain` will take a lot as well.

The first information you can get from this is whether one or more indexes were used to execute the query. Just check **involvedIndexes** property in the result. If this property is not in the result, then the executor did not use any indexes.

Unfortunately, **involvedIndexes** does not give you any information about *how* the index was used, eg. you don't know if it was used for full match or for partial match.

To have some more information, you can check **recordReads** property. It reports how many records were actually fetched an analized.

Another useful information is provided by **fullySortedByIndex**: if it returs `true` it means that no ORDER BY operations were performed in memory, but all the sorting relies on indexes. Unfortunately you do not have the opposite information, ie. if the index was used *only* for sorting and not for filtering.

Sneak peek of V 3.0

> **Please note** that all what was described above is going to change in v 3.0.

As a preview, here you can see the result of an EXPLAIN in OrientDB V 3.0

```
------------------------------
-- result of an EXPLAIN in V 3.0
------------------------------

explain select Name from Monuments
where Id < 15 and Name LIKE 'Statua%'
ORDER BY Name
SKIP 1 LIMIT 3
```

```
+ FETCH FROM INDEX Monuments.Id
  Id < 15
+ EXTRACT VALUE FROM INDEX ENTRY
+ FILTER ITEMS WHERE
  Name LIKE 'Statua%'
+ FILTER ITEMS BY CLASS
  Monuments
+ CALCULATE PROJECTIONS
  Name
+ ORDER BY Name ASC
  (buffer size: 4)
+ SKIP ( SKIP 1)
+ LIMIT ( LIMIT 3)
```

In V 3.0 the execution planner will be much smarter (and already is, in the SNAPSHOT release) than in v 2.2:

- the order of the conditions and parentheses does not matter anymore, OrientDB can figure out all the indexes that can be used for a query
- the choice of the right index for a query is based on statistics collected during query execution
- the execution planner is much more explicit, so you can exactly know how the query is being executed
- the EXPLAIN is calculated without executing the query, so it returns immediately, also for very expensive queries

Indexes

OrientDB supports four index algorithms:

- **SB-Tree Index** Provides a good mix of features available from other index types, good for general use. It is durable, transactional and supports range queries. It is the default index type.
- **Hash Index** Provides fast lookup and is very light on disk usage. It is durable and transactional, but does not support range queries. It works like a HashMap, which makes it faster on punctual lookups and it consumes less resources than other index types.
- **Auto Sharding Index** Provides an implementation of a DHT. It is durable and transactional, but does not support range queries. (Since v2.2)
- **Lucene Full Text Index** Provides good full-text indexes, but cannot be used to index other types. It is durable, transactional and supports range queries.
- **Lucene Spatial Index** Provides good spatial indexes, but cannot be used to index other types. It is durable, transactional and supports range queries.

Understanding Indexes

OrientDB can handle indexes in the same manner as classes, using the SQL language and prefixing the name with `index:` followed by the index name. An index is like a class with two properties:

- `key` The index key.
- `rid` The Record ID, which points to the record associated with the key.

Index Target

OrientDB can use two methods to update indexes:

- **Automatic** Where the index is bound to schema properties. (For example, `User.id`.) If you have a schema-less database and you want to create an automatic index, then you need to create the class and the property before using the index.

- **Manual** Where the index is handled by the application developer, using the Java API and SQL commands (see below). You can use them as Persistent Maps, where the entry's value are the records pointed to by the index.

You can rebuild automatic indexes using the `REBUILD INDEX` command.

Index Types

When you create an index, you create it as one of several available algorithm types. Once you create an index, you cannot change its type. OrientDB supports four index algorithms and several types within each. You also have the option of using any third-party index algorithms available through plugins.

- **SB-Tree Algorithm**
  - `UNIQUE` These indexes do not allow duplicate keys. For composite indexes, this refers to the uniqueness of the composite keys.
  - `NOTUNIQUE` These indexes allow duplicate keys.
  - `FULLTEXT` These indexes are based on any single word of text. You can use them in queries through the `CONTAINSTEXT` operator.
  - `DICTIONARY` These indexes are similar to those that use `UNIQUE`, but in the case of duplicate keys, they replaces the existing record with the new record.
- **HashIndex Algorithm**
  - `UNIQUE_HASH_INDEX` These indexes do not allow duplicate keys. For composite indexes, this refers to the uniqueness of the composite keys. Available since version 1.5.x.
  - `NOTUNIQUE_HASH_INDEX` These indexes allow duplicate keys. Available since version 1.5.x.
  - `FULLTEXT_HASH_INDEX` These indexes are based on any single word of text. You can use them in queries through the `CONTAINSTEXT` operator. Available since version 1.5.x.

  - `DICTIONARY_HASH_INDEX` These indexes are similar to those that use `UNIQUE_HASH_INDEX`, but in cases of duplicate keys, they replaces the existing record with the new record. Available since version 1.5.x.
- **HashIndex Algorithm** (Since v2.2)
  - `UNIQUE_HASH_INDEX` These indexes do not allow duplicate keys. For composite indexes, this refers to the uniqueness of the composite keys.
  - `NOTUNIQUE_HASH_INDEX` These indexes allow duplicate keys.
-
  - **Lucene Engine**
  - `FULLTEXT` These indexes use the Lucene engine to index string content. You can use them in queries with the `LUCENE` operator.
  - `SPATIAL` These indexes use the Lucene engine to index geospatial coordinates.

Every database has a default manual index type `DICTIONARY`, which uses strings as keys. You may find this useful in handling the root records of trees and graphs, and handling singleton records in configurations.

Indexes and Null Values

Starting from v2.2, Indexes do not ignore NULL values, but they are indexes as any other values. This means that if you have a UNIQUE index, you cannot have multiple NULL keys. This applies only to the new indexes, opening a database with indexes previously created, will all ignore NULL by default.

To create an index that expressly ignore nulls (like the default with v2.1 and earlier), look at the following examples by using SQL or Java API.

SQL:

```
orientdb> CREATE INDEX addresses ON Employee (address) NOTUNIQUE METADATA {ignoreNullValues: true}
```

And Java API:

```
schema.getClass(Employee.class).getProperty("address").createIndex(OClass.INDEX_TYPE.NOTUNIQUE, new ODocument().field("ignoreN
ullValues",true));
```

Indexes and Composite Keys

Operations that work with indexes also work with indexes formed from composite keys. By its nature, a composite key is a collection of values, so, syntactically, it is a collection.

For example, consider a case where you have a class `Book`, indexed by three fields: `author`, `title` and `publicationYear`. You might use the following query to look up an individual book:

```
orientdb> SELECT FROM INDEX:books WHERE key = ["Donald Knuth", "The Art of Computer
          Programming", 1968]
```

Alternatively, you can look for books over a range of years with the field `publicationYear`:

```
orientdb> SELECT FROM INDEX:books WHERE key BETWEEN ["Donald Knuth", "The Art of
          Computer Programming", 1960] AND ["Donald Knuth", "The Art of Computer
          Programming", 2000]
```

Partial Match Searches

Occasionally, you may need to search an index record by several fields of its composite key. In these partial match searches, the remaining fields with undefined values can match any value in the result.

Only use composite indexes for partial match searches when the declared fields in the composite index are used from left to right. For instance, from the example above searching only `title` wouldn't work with a composite index, since `title` is the second value. But, you could use it when searching `author` and `title`.

For example, consider a case where you don't care when the books in your database were published. This allows you to use a somewhat different query, to return all books with the same author and title, but from any publication year.

```
orientdb> SELECT FROM INDEX:books WHERE key = ["Donald Knuth", "The Art of Computer
          Programming"]
```

In the event that you also don't know the title of the work you want, you can further reduce it to only search all books with the same author.

```
orientdb> SELECT FROM INDEX:books WHERE key = ["Donald Knuth"]
```

Or, the equal,

```
orientdb> SELECT FROM INDEX:books WHERE key = "Donald Knuth"
```

Range Queries

Not all the indexes support range queries (check above). In the case of range queries, the field subject to the range must be the last one, (that is, the one on the far right). For example,

```
orientdb> SELECT FROM INDEX:books WHERE key BETWEEN ["Donald Knuth", "The Art of
          Computer Programming", 1900] AND ["Donald Knuth", "The Art of Computer
          Programming", 2014]
```

Operations against Indexes

Once you have a good understanding of the theoretical side of what indexes are and some of basic concepts that go into their use, it's time to consider the practical aspects of creating and using indexes with your application.

Creating Indexes

When you have created the relevant classes that you want to index, create the index. To create an automatic index, bound to a schema property, use the `ON` section or use the name in the `<class>.<property>` notation.

Syntax:

```
CREATE INDEX <name> [ON <class-name> (prop-names)] <type> [<key-type>]
                    [METADATA {<metadata>}]
```

- `<name>` Provides the logical name for the index. You can also use the `<class.property>` notation to create an automatic index bound to a schema property. In this case, for `<class>` use the class of the schema and `<property>` the property created in the class.

  Bear in mind that this means case index names cannot contain the period (`.`) symbol, as OrientDB would interpret the text after as a property.

- `<class-name>` Provides the name of the class that you are creating the automatic index to index. This class must already exist in the database.

- `<prop-names>` Provides a comma-separated list of properties, which you want the automatic index to index. These properties must already exist in the schema.

If the property belongs to one of the Map types, (such as `LINKMAP`, or `EMBEDDEDMAP`), you can specify the keys or values to use in generating indexes. You can do this with the `BY KEY` or `BY VALUE` expressions, if nothing is specified, these keys are used during index creation.

- `<type>` Provides the algorithm and type of index that you want to create. For information on the supported index types, see Index Types.
- `<key-type>` Provides the optional key type. With automatic indexes, the key type OrientDB automatically determines the key type by reading the target schema property where the index is created. With manual indexes, if not specified, OrientDB automatically determines the key type at run-time, during the first insertion by reading the type of the class.
- `<metadata>` Provides a JSON representation

Examples:

- Creating custom indexes:

```
orientdb> CREATE INDEX mostRecentRecords UNIQUE date
```

- Creating automatic indexes bound to the property `id` of the class `User`:

```
orientdb> CREATE PROPERTY User.id BINARY
orientdb> CREATE INDEX User.id UNIQUE
```

- Creating another index for the property `id` of the class `User`:

```
orientdb> CREATE INDEX indexForId ON User (id) UNIQUE
```

- Creating indexes for property `thumbs` on class `Movie`:

```
orientdb> CREATE INDEX thumbsAuthor ON Movie (thumbs) UNIQUE
orientdb> CREATE INDEX thumbsAuthor ON Movie (thumbs BY KEY) UNIQUE
orientdb> CREATE INDEX thumbsValue on Movie (thumbs BY VALUE) UNIQUE
```

- Creating composite indexes:

```
orientdb> CREATE PROPERTY Book.author STRING
orientdb> CREATE PROPERTY Book.title STRING
orientdb> CREATE PROPERTY Book.publicationYears EMBEDDEDLIST INTEGER
orientdb> CREATE INDEX books ON Book (author, title, publicationYears) UNIQUE
```

For more information on creating indexes, see the `CREATE INDEX` command.

Dropping Indexes

In the event that you have an index that you no longer want to use, you can drop it from the database. This operation does not remove linked records.

Syntax:

```
DROP INDEX <name>
```

- `<name>` provides the name of the index you want to drop.

For more information on dropping indexes, see the `DROP INDEX` command.

Querying Indexes

When you have an index created and in use, you can query records in the index using the `SELECT` command.

Syntax:

```
SELECT FROM INDEX:<index-name> WHERE key = <key>
```

Example:

- Selecting from the index `dictionary` where the key matches to `Luke`:

  ```
  orientdb> SELECT FROM INDEX:dictionary WHERE key='Luke'
  ```

Case-insensitive Matching with Indexes

In the event that you would like the index to use case-insensitive matching, set the `COLLATE` attribute of the indexed properties to `ci`. For instance,

```
orientdb> CREATE INDEX OUser.name ON OUser (name COLLATE ci) UNIQUE
```

Inserting Index Entries

You can insert new entries into the index using the `key` and `rid` pairings.

Syntax:

```
INSERT INTO INDEX:<index-name> (key,rid) VALUES (<key>,<rid>)
```

Example:

- Inserting the key `Luke` and Record ID `#10:4` into the index `dictionary`:

  ```
  orientdb> INSERT INTO INDEX:dictionary (key, rid) VALUES ('Luke', #10:4)
  ```

Querying Index Ranges

In addition to querying single results from the index, you can also query a range of results between minimum and maximum values. Bear in mind that not all index types support this operation.

Syntax:

```
SELECT FROM INDEX:<index-name> WHERE key BETWEEN <min> AND <max>
```

Example:

- Querying from the index `coordinates` and range between `10.3` and `10.7`:

  ```
  orientdb> SELECT FROM INDEX:coordinates WHERE key BETWEEN 10.3 AND 10.7
  ```

Removing Index Entries

You can delete entries by passing the `key` and `rid` values. This operation returns `TRUE` if the removal was successful and `FALSE` if the entry wasn't found.

Syntax:

```
DELETE FROM INDEX:<index-name> WHERE key = <key> AND rid = <rid>
```

Example:

- Removing an entry from the index `dictionary` :

  ```
  orientdb> DELETE FROM INDEX:dictionary WHERE key = 'Luke' AND rid = #10:4
  ```

Removing Index Entries by Key

You can delete all entries from the index through the requested key.

Syntax:

```
DELETE FROM INDEX:<index-name> WHERE key = <key>
```

Example:

- Delete entries from the index `addressbook` whee the key matches to `Luke` :

  ```
  orientdb> DELETE FROM INDEX:addressbook WHERE key = 'Luke'
  ```

Removing Index Entries by RID

You can remove all index entries to a particular record by its record ID.

Syntax:

```
DELETE FROM INDEX:<index-name> WHERE rid = <rid>
```

Example:

- Removing entries from index `dictionary` tied to the record ID `#10:4` :

  ```
  orientdb> DELETE FROM INDEX:dictionary WHERE rid = #10:4
  ```

Counting Index Entries

To see the number of entries in a given index, you can use the `COUNT()` function.

Syntax:

```
SELECT COUNT(*) AS size FROM INDEX:<index-name>
```

Example:

- Counting the entries on the index `dictionary` :

  ```
  orientdb> SELECT COUNT(*) AS size FROM INDEX:dictionary
  ```

Querying Keys from Indexes

You can query all keys in an index using the `SELECT` command.

Syntax:

```
SELECT key FROM INDEX:<index-name>
```

Example:

- Querying the keys in the index `dictionary`:

  ```
  orientdb> SELECT key FROM INDEX:dictionary
  ```

Querying Index Entries

You can query for all entries on an index as `key` and `rid` pairs.

Syntax:

```
SELECT key, value FROM INDEX:<index-name>
```

Example:

- Querying the `key` / `rid` pairs from the index `dictionary`:

  ```
  orientdb> SELECT key, value FROM INDEX:dictionary
  ```

Clearing Indexes

Remove all entries from an index. After running this command, the index is empty.

Syntax:

```
DELETE FROM INDEX:<index-name>
```

Example:

- Removing all entries from the index `dictionary`:

  ```
  orientdb> DELETE FROM INDEX:dictionary
  ```

Custom Keys

OrientDB includes support for the creation of custom keys for indexes. This feature has been available since version 1.0, and can provide you with a huge performance improvement, in the event that you would like minimize memory usage by developing your own serializer.

For example, consider a case where you want to handle SHA-256 data as binary keys without using a string to represent it, which would save on disk space, CPU and memory usage.

To implement this, begin by creating your own type,

```
public static class ComparableBinary implements Comparable<ComparableBinary> {
      private byte[] value;

      public ComparableBinary(byte[] buffer) {
            value = buffer;
      }

      public int compareTo(ComparableBinary o) {
            final int size = value.length;
            for (int i = 0; i < size; ++i) {
                  if (value[i] > o.value[i])
                        return 1;
                  else if (value[i] < o.value[i])
                        return -1;
            }
            return 0;
      }

      public byte[] toByteArray() {
            return value;
      }
}
```

With your index type created, next create a binary selector for it to use:

```
public static class OHash256Serializer implements OBinarySerializer<ComparableBinary> {

      public static final OBinaryTypeSerializer INSTANCE = new OBinaryTypeSerializer();
      public static final byte ID = 100;
      public static final int LENGTH = 32;

      public int getObjectSize(final int length) {
            return length;
      }

      public int getObjectSize(final ComparableBinary object) {
            return object.toByteArray().length;
      }

      public void serialize(
         final ComparableBinary object,
         final byte[] stream,
     final int startPosition) {
                  final byte[] buffer = object.toByteArray();
                  System.arraycopy(buffer, 0, stream, startPosition, buffer.length);
            }

      public ComparableBinary deserialize(
         final byte[] stream,
     final int startPosition) {
                 final byte[] buffer = Arrays.copyOfRange(
                  stream,
                  startPosition,
                  startPosition + LENGTH);
                  return new ComparableBinary(buffer);
      }

      public int getObjectSize(byte[] stream, int startPosition) {
            return LENGTH;
      }

      public byte getId() {
            return ID;
      }
}
```

Lastly, register the new serializer with OrientDB:

```
OBinarySerializerFactory.INSTANCE.registerSerializer(new OHash256Serializer(), null);
index = database.getMetadata().getIndexManager().createIndex("custom-hash", "UNIQUE", new ORuntimeKeyIndexDefinition(OHash256S
erializer.ID), null, null);
```

Your custom keys are now available for use in searches:

```
ComparableBinary key1 = new ComparableBinary(new byte[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 2,
3, 4, 5, 6, 7, 8, 9, 0, 1 });
ODocument doc1 = new ODocument().field("k", "key1");
index.put(key1, doc1);
```

Query the available indexes

To access to the indexes, you can use SQL.

Create your index engine

Here you can find a guide how to create a custom index engine.

Create a manual index in Java

To create a manual index in Java, you can use the following method:

```
OIndexManager.createIndex(final String iName, final String iType, final OIndexDefinition indexDefinition, final int[] clusterI
dsToIndex, final OProgressListener progressListener, final ODocument metadata)
```

- `iName` : the index name
- `iType` : the index type (UNIQUE, NOTUNIQUE, HASH_INDEX ecc.)
- `indexDefinition` : the definition of the key type. You can use OSimpleKeyIndexDefinition
- `clusterIdsToIndex` : this has to be null, because you are creating a manual index
- `progressListener` : a progress index for the index creation (it can be null)
- `metadata` : the index metadata (Eg. the index engine). For basic unique and notunique indexes it can be null

An example of its usage is following:

```
OIndexManager idxManager = db.getMetadata().getIndexManager();
idxManager.createIndex("myManualIndex", "NOTUNIQUE", new OSimpleKeyIndexDefinition(-1, OType.STRING), null, null, null);
```

SB-Tree Index Algorithm

This indexing algorithm provides a good mix of features, similar to the features available from other index types. It is good for general use and is durable, transactional and supports range queries. There are four index types that utilize the SB-Tree index algorithm:

- `UNIQUE` Does not allow duplicate keys, fails when it encounters duplicates.
- `NOTUNIQUE` Does allow duplicate keys.
- `FULLTEXT` Indexes to any single word of text.
- `DICTIONARY` Does not allow duplicate keys, overwrites when it encounters duplicates.

> For more information on `FULLTEXT_HASH_INDEX`, see FullText Index.

The SB-Tree index algorithm is based on the B-Tree index algorithm. It has been adapted with several optimizations, which relate to data insertion and range queries. As is the case with all other tree-based indexes, SB-Tree index algorithm experiences `log(N)` complexity, but the base to this logarithm is about 500.

> **NOTE**: There is an issue in the replacement of indexes based on B-Tree with those based on COLA Tree to avoid slowdowns introduced by random I/O operations. For more information see Issue #1756.

Hash Index Algorithm

This indexing algorithm provides a fast lookup and is very light on disk usage. It is durable and transactional, but does not support range queries. It is similar to a HashMap, which makes it faster on punctual lookups and it consumes less resources than other index types. The Hash index algorithm supports four index types, which have been available since version 1.5.x:

- `UNIQUE_HASH_INDEX` Does not allow duplicate keys, it fails when it encounters duplicates.
- `NOTUNIQUE_HASH_INDEX` Does allow duplicate keys.
- `FULLTEXT_HASH_INDEX` Indexes to any single word.
- `DICTIONARY` Does not allow duplicate keys, it overwrites when it encounters duplicates.

> For more information on `FULLTEXT_HASH_INDEX`, see FullText Index.

Hash indexes are able to perform index read operations in one I/O operation and write operations in a maximum of three I/O operations. The Hash Index algorithm is based on the Extendible Hashing algorithm. Despite not providing support for range queries, it is noticeably faster than SB-Tree Index Algorithms, (about twice as fast when querying through ten million records).

> **NOTE**: There is an issue relating to the enhancement of Hash indexes to avoid slowdowns introduced by random I/O operations using LSM Tree approaches. For more information, see Issue #1757.

Auto Sharding Index Algorithm

(Since v2.2)

This indexing algorithm is based on the DHT concept, where they keys are stored on different partition, based on the Murmur3 hash function.

Auto Sharding Index supports the following index types:

- `UNIQUE_HASH_INDEX` Does not allow duplicate keys, it fails when it encounters duplicates.
- `NOTUNIQUE_HASH_INDEX` Does allow duplicate keys.

Under the hood, this index creates multiple Hash Indexes, one per cluster. So if you have 8 clusters for the class "Employee", this index will create, at the beginning, 8 Hash Indexes.

Since this index is based on the Hash Index, it's able to perform index read operations in one I/O operation and write operations in a maximum of three I/O operations. The Hash Index algorithm is based on the Extendible Hashing algorithm. Despite not providing support for range queries, it is noticeably faster than SB-Tree Index Algorithms, (about twice as fast when querying through ten million records).

Usage

Create an index by passing "AUTOSHARDING" as index engine:

```
final OClass cls = db.getMetadata().getSchema().createClass("Log");
cls.createProperty("key", OType.LONG);
cls.createIndex("idx_LogKey", OClass.INDEX_TYPE.UNIQUE.toString(),
    (OProgressListener) null, (ODocument) null, "AUTOSHARDING", new String[] { "key" });
```

Performance

On multi-core hw, using this index instead of Hash Index gives about +50% more throughput on insertion on a 8 cores machine.

Distributed

The fully distributed version of this index will be supported in v3.0. In v2.2 each node has own copy of the index with all the partitions.

Internals

This is the algorithm for the `put(key,value)` :

```
int partition = Murmur3_hash(key) % partitions;
getSubIndex(partition).put(key,value);
```

This is for the `value = get(key)` :

```
int partition = Murmur3_hash(key) % partitions;
return getSubIndex(partition).get(key);
```

FullText Indexes

The SB-Tree index algorithm provides support for FullText indexes. These indexes allow you to index text as a single word and its radix. FullText indexes are like having a search engine on your database.

> **NOTE**: Bear in mind that there is a difference between `FULLTEXT` without the `LUCENE` operator, which uses a FullText index with the SB-Tree index algorithm and `FULLTEXT` with the `LUCENE` operator, which uses a FullText index through the Lucene Engine.
>
> For more information on the latter, see Lucene FullText Index.

Creating FullText Indexes

If you want to create an index using the FullText SB-Tree index algorithm, you can do so using the `CREATE INDEX` command.

```
orientdb> CREATE INDEX City.name ON City(name) FULLTEXT
```

This creates a FullText index on the property `name` of the class `City`, using the default configuration.

FullText Index Parameters

In the event that the default FullText Index configuration is not sufficient to your needs, there are a number of parameters available to fine tune how it generates the index.

Parameter	Default	Description
`indexRadix`	`TRUE`	Word prefixes will be also index
`ignoreChars`	`"`	Chars to skip when indexing
`separatorChars`	`\r\n\t:;,.&#124;+*/\=!?[](.md)`	
`minWordLength`	`3`	Minimum word length to index
`stopWords`	`the in a at as and or for his her him this that what which while up with be was were is`	Stop words escluded from indexing

To configure a FullText Index, from version 1.7 on, you can do so through the OrientDB console or the Java API. When configuring the index from the console, use the `CREATE INDEX` command with the `METADATA` operator.

```
orientdb> CREATE INDEX City.name ON City(name) FULLTEXT METADATA
          {"indexRadix": true, "ignoreChars": "&", "separatorChars": " |()",
          "minWordLength": 4, "stopWords": ["the", "of"]}
```

Alternatively, you can configure the index in Java.

```
OSchema schema = db.getMetadata().getSchema();
OClass city = schema.getClass("City");
ODocument metadata = new ODocument();
metadata.field("indexRadix", true);
metadata.field("stopWords", Arrays.asList(new String[] { "the", "in", "a", "at" }));
metadata.field("separatorChars", " :;?[](.md)");
metadata.field("ignoreChars", "$&");
metadata.field("minWordLength", 5);
city.createIndex("City.name", "FULLTEXT", null, metadata, null, new String[] { "name" });
```

Lucene FullText Index

In addition to the standard FullText Index, which uses the SB-Tree index algorithm, you can also create FullText indexes using the Lucene Engine. Beginning from version 2.0, this plugin is packaged with OrientDB distribution.

Syntax:

```
orientdb> CREATE INDEX  ON  (prop-names) FULLTEXT ENGINE LUCENE
```

The following SQL statement will create a FullText index on the property `name` for the class `City`, using the Lucene Engine.

```
orientdb> CREATE INDEX City.name ON City(name) FULLTEXT ENGINE LUCENE
```

Indexes can also be created on *n*-properties. For example, create an index on the properties `name` and `description` on the class `City`.

```
orientdb> CREATE INDEX City.name_description ON City(name, description)
          FULLTEXT ENGINE LUCENE
```

The default analyzer used by OrientDB when a Lucene index is created id the StandardAnalyzer.

Analyzer

In addition to the StandardAnalyzer, full text indexes can be configured to use different analyzer by the `METADATA` operator through `CREATE INDEX`.

Configure the index on `City.name` to use the `EnglishAnalyzer`:

```
orientdb> CREATE INDEX City.name ON City(name)
            FULLTEXT ENGINE LUCENE METADATA {
                "analyzer": "org.apache.lucene.analysis.en.EnglishAnalyzer"
            }
```

(from 2.1.16)

From version 2.1.16 it is possible to provide a set of stopwords to the analyzer to override the default set of the analyzer:

```
orientdb> CREATE INDEX City.name ON City(name) FULLTEXT ENGINE LUCENE METADATA
          {
          "analyzer": "org.apache.lucene.analysis.en.EnglishAnalyzer",
          "analyzer_stopwords": ["a", "an", "and", "are", "as", "at", "be", "but", "by" ]
          }
```

(from 2.2)

Starting from 2.2 it is possible to configure different analyzers for indexing and querying.

```
orientdb> CREATE INDEX City.name ON City(name) FULLTEXT ENGINE LUCENE METADATA
          {
          "index": "org.apache.lucene.analysis.en.EnglishAnalyzer",
          "query": "org.apache.lucene.analysis.standard.StandardAnalyzer"
          }
```

EnglishAnalyzer will be used to analyze text while indexing and the StandardAnalyzer will be used to analyze query terms.

A very detailed configuration, on multi-field index configuration, could be:

```
orientdb> CREATE INDEX City.name_description ON City(name, lyrics, title,author, description) FULLTEXT ENGINE LUCENE METAD
ATA
          {
            "default": "org.apache.lucene.analysis.standard.StandardAnalyzer",
            "index": "org.apache.lucene.analysis.core.KeywordAnalyzer",
            "query": "org.apache.lucene.analysis.standard.StandardAnalyzer",
            "name_index": "org.apache.lucene.analysis.standard.StandardAnalyzer",
            "name_query": "org.apache.lucene.analysis.core.KeywordAnalyzer",
            "lyrics_index": "org.apache.lucene.analysis.en.EnglishAnalyzer",
            "title_index": "org.apache.lucene.analysis.en.EnglishAnalyzer",
            "title_query": "org.apache.lucene.analysis.en.EnglishAnalyzer",
            "author_query": "org.apache.lucene.analysis.core.KeywordAnalyzer",
            "description_index": "org.apache.lucene.analysis.standard.StandardAnalyzer",
            "description_index_stopwords": [
              "the",
              "is"
            ]

          }
```

With this configuration, the underlying Lucene index will works in different way on each field:

- *name*: indexed with StandardAnalyzer, searched with KeywordAnalyzer (it's a strange choice, but possible)
- *lyrics*: indexed with EnglishAnalyzer, searched with default query analyzer StandardAnalyzer
- *title*: indexed and searched with EnglishAnalyzer
- *author*: indexed and searched with KeywordhAnalyzer
- *description*: indexed with StandardAnalyzer with a given set of stopwords

Java API

The FullText Index with the Lucene Engine is configurable through the Java API.

```
OSchema schema = databaseDocumentTx.getMetadata().getSchema();
OClass oClass = schema.createClass("Foo");
oClass.createProperty("name", OType.STRING);
oClass.createIndex("City.name", "FULLTEXT", null, null, "LUCENE", new String[] { "name"});
```

Query parser

It is possible to configure some behavior of the Lucene query parser

Allow Leading Wildcard

Lucene by default doesn't support leading wildcard: Lucene wildcard support

It is possible to override this behavior with a dedicated flag on meta-data:

```
{
  "allowLeadingWildcard": true
}
```

Use this flag carefully, as stated in the Lucene FAQ:

> Note that this can be an expensive operation: it requires scanning the list of tokens in the index in its entirety to look for those that match the pattern.

Disable lower case on terms

Lucene's QueryParser applies a lower case filter on expanded queries by default. It is possible to override this behavior with a dedicated flag on meta-data:

```
{
  "lowercaseExpandedTerms": false
}
```

It is useful when used in pair with keyword analyzer:

```
{
  "lowercaseExpandedTerms": false,
  "default" : "org.apache.lucene.analysis.core.KeywordAnalyzer"
}
```

With *lowercaseExpandedTerms* set to false, these two queries will return different results:

```
SELECT from Person WHERE name LUCENE "NAME"

SELECT from Person WHERE name LUCENE "name"
```

Lucene Writer fine tuning (expert)

It is possible to fine tune the behaviour of the underlying Lucene's IndexWriter

```
CREATE INDEX City.name ON City(name) FULLTEXT ENGINE LUCENE METADATA
{
  "directory_type": "nio",
  "use_compound_file": false,
  "ram_buffer_MB": "16",
  "max_buffered_docs": "-1",
  "max_buffered_delete_terms": "-1",
  "ram_per_thread_MB": "1024",
  "default": "org.apache.lucene.analysis.standard.StandardAnalyzer"
}
```

- *directory_type*: configure the acces type to the Lucene's index
  - *nio* (*default)*: the index is opened with *NIOFSDirectory*
  - *mmap*: the index is opened with *MMapDirectory*
  - *ram*: index will be created in memory with *RAMDirectory*
- *use_compound_file*: default is false
- *ram_buffer_MB*: size of the document's buffer in MB, default value is 16 MB (which means flush when buffered docs consume approximately 16 MB RAM)
- *max_buffered_docs*: size of the document's buffer in number of docs, disabled by default (because IndexWriter flushes by RAM usage by default)
- *max_buffered_delete_terms*: disabled by default (because IndexWriter flushes by RAM usage by default).
- *ram_per_thread_MB*: default value is 1945

For a detailed explanation of config parameters and IndexWriter behaviour

- indexWriterConfig : https://lucene.apache.org/core/5_0_0/core/org/apache/lucene/index/IndexWriterConfig.html
- indexWriter: https://lucene.apache.org/core/5_0_0/core/org/apache/lucene/index/IndexWriter.html

Index lifecycle

Starting from 2.2.24, Lucene indexes are lazy. If the index is in idle mode, no reads and no writes, it will be closed. Intervals are fully configurable.

- *flushIndexInterval*: flushing index interval in milliseconds, default to 10000 (10s)
- *closeAfterInterval*: closing index interval in millisecons, default to 20000 (20s)
- *firstFlushAfter*: first flush time in milliseconds, default to 10000 (10s)

To configure the index lifecycle, just pass the parameters in the JSON of metadata:

```
CREATE INDEX City.name ON City(name) FULLTEXT ENGINE LUCENE METADATA
{
  "flushIndexInterval": 20000,
  "closeAfterInterval": 20000,
  "firstFlushAfter": 20000
}
```

Querying Lucene FullText Indexes

You can query the Lucene FullText Index using the custom operator `LUCENE` with the Query Parser Syntax from the Lucene Engine.

```
orientdb> SELECT FROM V WHERE name LUCENE "test*"
```

This query searches for `test`, `tests`, `tester`, and so on from the property `name` of the class `V`. The query can use proximity operator ~, the required (+) and prohibit (-) operators, phrase queries, regexp queries:

```
orientdb> SELECT FROM Article WHERE content LUCENE "(+graph -rdbms) AND +cloud"
```

Working with multiple fields

In addition to the standard Lucene query above, you can also query multiple fields. For example,

```
orientdb> SELECT FROM Class WHERE [prop1, prop2] LUCENE "query"
```

In this case, if the word `query` is a plain string, the engine parses the query using MultiFieldQueryParser on each indexed field.

To execute a more complex query on each field, surround your query with parentheses, which causes the query to address specific fields.

```
orientdb> SELECT FROM Article WHERE [content, author] LUCENE "(content:graph AND author:john)"
```

Here, the engine parses the query using the QueryParser

Numeric and date range queries (from 2.2.14)

If the index is defined over a numeric field (INTEGER, LONG, DOUBLE) or a date field (DATE, DATETIME), the engine supports range queries Suppose to have a `City` class witha multi-field Lucene index defined:

```
orientdb>
CREATE CLASS CITY EXTENDS V
CREATE PROPERTY CITY.name STRING
CREATE PROPERTY CITY.size INTEGER
CREATE INDEX City.name ON City(name,size) FULLTEXT ENGINE LUCENE
```

Then query using ranges:

```
orientdb>
SELECT FROM City WHERE [name,size] LUCENE 'name:cas* AND size:[15000 TO 20000]'
```

Ranges can be applied to DATE/DATETIME field as well. Create a Lucene index over a property:

```
orientdb>
CREATE CLASS Article EXTENDS V
CREATE PROPERTY Article.createdAt DATETIME
CREATE INDEX Article.createdAt  ON Article(createdAt) FULLTEXT ENGINE LUCENE
```

Then query to retrieve articles published only in a given time range:

```
orientdb>
SELECT FROM Article WHERE createdAt LUCENE '[201612221000 TO 201612221100]'
```

Retrieve the Score

When the lucene index is used in a query, the results set carries a context variable for each record representing the score. To display the score add `$score` in projections.

```
SELECT *,$score FROM V WHERE name LUCENE "test*"
```

Creating a Manual Lucene Index

The Lucene Engine supports index creation without the need for a class.

Syntax:

```
CREATE INDEX <name> FULLTEXT ENGINE LUCENE  [<key-type>] [METADATA {<metadata>}]
```

For example, create a manual index using the `CREATE INDEX` command:

```
orientdb> CREATE INDEX Manual FULLTEXT ENGINE LUCENE STRING, STRING
```

Once you have created the index `Manual`, you can insert values in index using the `INSERT INTO INDEX:...` command.

```
orientdb> INSERT INTO INDEX:Manual (key, rid) VALUES(['Enrico', 'Rome'], #5:0)
```

You can then query the index through `SELECT...FROM INDEX:`:

```
orientdb> SELECT FROM INDEX:Manual WHERE key LUCENE "Enrico"
```

Manual indexes could be created programmatically using the Java API

```
ODocument meta = new ODocument().field("analyzer", StandardAnalyzer.class.getName());
OIndex<?> index = databaseDocumentTx.getMetadata().getIndexManager()
        .createIndex("apiManual", OClass.INDEX_TYPE.FULLTEXT.toString(),
            new OSimpleKeyIndexDefinition(1, OType.STRING, OType.STRING), null, null, meta, OLuceneIndexFactory.LUCENE_ALGORIT
HM);
```

Lucene Spatial

(Versions 2.2 and after only, otherwise look at Legacy section)

This module is provided as external plugin. You can find it bundled in the GeoSpatial distribution, or you can add this plugin by yourself into any OrientDB distribution (Community and Enterprise Editions).

Install

Download the plugin jar from maven central:

```
http://central.maven.org/maven2/com/orientechnologies/orientdb-spatial/2.2.37/orientdb-spatial-2.2.37-dist.jar
```

After download, copy the jar to OrientDB lib directory (please make sure that the version of your OrientDB server and the version of the plugin are the same. Upgrade your OrientDB server, if necessary). On *nix system it could be done this way:

```
wget http://central.maven.org/maven2/com/orientechnologies/orientdb-spatial/2.2.37/orientdb-spatial-2.2.37-dist.jar
cp orientdb-spatial-2.2.37-dist.jar /PATH/orientdb-community-2.2.37/lib/
```

OrientDB will load the spatial plugin on startup.

Orient db will load the spatial plugin on startup.

Geometry Data

OrientDB supports the following Geometry objects :

- Point (**OPoint**)
- Line (**OLine**)
- Polygon (**OPolygon)**
- MultiPoint (**OMultiPoint**)
- MultiLine (**OMultiline**)
- MultiPolygon (**OMultiPlygon**)
- Geometry Collections

OrientDB stores those objects like embedded documents with special classes. The module creates abstract classes that represent each Geometry object type, and those classes can be embedded in user defined classes to provide geospatial information.

Each spatial classes (Geometry Collection excluded) comes with field coordinates that will be used to store the geometry structure. The "coordinates" field of a geometry object is composed of one position (Point), an array of positions (LineString or MultiPoint), an array of arrays of positions (Polygons, MultiLineStrings) or a multidimensional array of positions (MultiPolygon).

Geometry data Example

Restaurants Domain

```
CREATE class Restaurant
CREATE PROPERTY Restaurant.name STRING
CREATE PROPERTY Restaurant.location EMBEDDED OPoint
```

To insert restaurants with location

From SQL

```
INSERT INTO  Restaurant SET name = 'Dar Poeta', location = {"@class": "OPoint","coordinates" : [12.4684635,41.8914114]}
```

or as an alternative, if you use WKT format you can use the function `ST_GeomFromText` to create the OrientDB geometry object.

```
INSERT INTO  Restaurant SET name = 'Dar Poeta', location = St_GeomFromText("POINT (12.4684635 41.8914114)")
```

From JAVA

```
ODocument location = new ODocument("OPoint");
location.field("coordinates", Arrays.asList(12.4684635, 41.8914114));

ODocument doc = new ODocument("Restaurant");
doc.field("name","Dar Poeta");
doc.field("location",location);

doc.save();
```

A spatial index on the *location* field s defined by

```
CREATE INDEX Restaurant.location ON Restaurant(location) SPATIAL ENGINE LUCENE"
```

Spatial Reference System

OrientDB supports only EPSG:4326 as Spatial Reference System

Functions

OrientDB follows The Open Geospatial Consortium OGC for extending SQL to support spatial data. OrientDB implements a subset of SQL-MM functions with ST prefix (Spatial Type)

ST_AsText

Syntax : ST_AsText(geom)

Example

```
SELECT ST_AsText({"@class": "OPoint","coordinates" : [12.4684635,41.8914114]})

Result
----------
POINT (12.4684635 41.8914114)
```

ST_GeomFromText

Syntax : ST_GeomFromText(text)

Example

```
select ST_GeomFromText("POINT (12.4684635 41.8914114)")

Result
----------------------------------------------------------------------------------
{"@type":"d","@version":0,"@class":"OPoint","coordinates":[12.4684635,41.8914114]}
```

ST_Equals

Returns true if geom1 is spatially equal to geom2

Syntax : ST_Equals(geom1,geom2)

Example

```
SELECT ST_Equals(ST_GeomFromText('LINESTRING(0 0, 10 10)'), ST_GeomFromText('LINESTRING(0 0, 5 5, 10 10)'))

Result
-----------
true
```

ST_Within

Returns true if geom1 is inside geom2

Syntax : ST_Within(geom1,geom2)

This function will use an index if available.

Example

```
select * from City where  ST_WITHIN(location,'POLYGON ((12.314015 41.8262816, 12.314015 41.963125, 12.6605063 41.963125, 12.66
05063 41.8262816, 12.314015 41.8262816))') = true
```

ST_Distance

Returns the 2D Cartesian distance between two geometries

Syntax : ST_Distance(geom1,geom2)

Example

```
select ST_Distance(ST_GEOMFROMTEXT('POINT(12.4662748 41.8914114)'),ST_GEOMFROMTEXT('POINT(12.4664632 41.8904382)')) as distanc
e

distance
--------
0.0009912682785239528
```

ST_Distance_Sphere (From OrientDB 2.2.4)

Returns the distance between two geometries in meters using a spherical earth. Supports Only Points. This function will use an index if available (Only if the condition is < or <=).

Example

```
SELECT ST_Distance_Sphere(ST_GeomFromText('POINT(12.4686519 41.890438)'), ST_GeomFromText('POINT(12.468933 41.890303)')) as di
stance
```

In where condition

```
SELECT from Place where ST_Distance_Sphere(location, ST_GeomFromText('POINT(12.468933 41.890303)')) < 50
```

ST_DWithin

Returns true if the geometries are within the specified distance of one another

Syntax : ST_DWithin(geom1,geom2,distance)

Example

```
SELECT ST_DWithin(ST_GeomFromText('POLYGON((0 0, 10 0, 10 5, 0 5, 0 0))'), ST_GeomFromText('POLYGON((12 0, 14 0, 14 6, 12 6, 1
2 0))'), 2.0d) as distance
```

```
SELECT from Polygon where ST_DWithin(geometry, ST_GeomFromText('POLYGON((12 0, 14 0, 14 6, 12 6, 12 0))'), 2.0) = true
```

ST_Contains

Returns true if geom1 contains geom2

Syntax : ST_Contains(geom1,geom2)

This function will use an index if available.

Example

```
SELECT ST_Contains(ST_Buffer(ST_GeomFromText('POINT(0 0)'),10),ST_GeomFromText('POINT(0 0)'))

Result
----------
true
```

```
SELECT ST_Contains(ST_Buffer(ST_GeomFromText('POINT(0 0)'),10),ST_Buffer(ST_GeomFromText('POINT(0 0)'),20))

Result
----------
false
```

ST_Disjoint

Returns true if geom1 does not spatially intersects geom2

Syntax: St_Disjoint(geom1,geom2)

This function does not use indexes

Example

```
SELECT ST_Disjoint(ST_GeomFromText('POINT(0 0)'), ST_GeomFromText('LINESTRING ( 2 0, 0 2 )'));

Result
-----------------
true
```

```
SELECT ST_Disjoint(ST_GeomFromText('POINT(0 0)'), ST_GeomFromText('LINESTRING ( 0 0, 0 2 )'));

Result
-----------------
false
```

ST_Intersects

Returns true if geom1 spatially intersects geom2

Syntax: ST_Intersects(geom1,geom2)

Example

```
SELECT ST_Intersects(ST_GeomFromText('POINT(0 0)'), ST_GeomFromText('LINESTRING ( 2 0, 0 2 )'));

Result
-------------
false
```

```
SELECT ST_Intersects(ST_GeomFromText('POINT(0 0)'), ST_GeomFromText('LINESTRING ( 0 0, 0 2 )'));

Result
-------------
true
```

ST_AsBinary

Returns the Well-Known Binary (WKB) representation of the geometry

Syntax : ST_AsBinary(geometry)

Example

```
SELECT ST_AsBinary(ST_GeomFromText('POINT(0 0)'))
```

ST_Envelope

Returns a geometry representing the bounding box of the supplied geometry

Syntax : ST_Envelope(geometry)

Example

```
SELECT ST_AsText(ST_Envelope(ST_GeomFromText('POINT(1 3)')));

Result
----------
POINT (1 3)
```

```
SELECT ST_AsText(ST_Envelope(ST_GeomFromText('LINESTRING(0 0, 1 3)')))

Result
-----------------------------------
POLYGON ((0 0, 0 3, 1 3, 1 0, 0 0))
```

ST_Buffer

Returns a geometry that represents all points whose distance from this Geometry is less than or equal to distance.

Syntax: ST_Buffer(geometry,distance [,config])

where config is an additional parameter (JSON) that can be use to set:

quadSegs: int -> number of segments used to approximate a quarter circle (defaults to 8).

```
{
  quadSegs : 1
}
```

endCap : round|flat|square -> endcap style (defaults to "round").

```
{
  endCap : 'square'
}
```

join : round|mitre|bevel -> join style (defaults to "round")

```
{
  join : 'bevel'
}
```

mitre : double -> mitre ratio limit (only affects mitered join style).

```
{
  join : 'mitre',
  mitre : 5.0
}
```

Example

```
SELECT ST_AsText(ST_Buffer(ST_GeomFromText('POINT(100 90)'),50))
```

```
SELECT ST_AsText(ST_Buffer(ST_GeomFromText('POINT(100 90)'), 50, { quadSegs : 2 }));
```

Operators

A && B

Overlaps operator. Returns true if bounding box of A overlaps bounding box of B. This operator will use an index if available.

Example

```
CREATE CLASS TestLineString
CREATE PROPERTY TestLineString.location EMBEDDED OLineString
INSERT INTO TestLineSTring SET name = 'Test1' , location = St_GeomFromText("LINESTRING(0 0, 3 3)")
INSERT INTO TestLineSTring SET name = 'Test2' , location = St_GeomFromText("LINESTRING(0 1, 0 5)")
SELECT FROM TestLineString WHERE location && "LINESTRING(1 2, 4 6)"
```

Spatial Indexes

To speed up spatial search and match condition, spatial operators and functions can use a spatial index if defined to avoid sequential full scan of every records.

The current spatial index implementation is built upon lucene-spatial.

The syntax for creating a spatial index on a geometry field is :

```
CREATE INDEX <name> ON <class-name> (geometry-field) SPATIAL ENGINE LUCENE
```

Legacy

Before v2.2, OrientDB was able to only index Points. Other Shapes like rectangles and polygons are managed starting from v2.2 (look above). This is the legacy section for databases created before v2.2. **NOTE:** If you are migrating to 2.2.x from 2.1.x and you are going to index null values, remember to add `ignoreNullValues: true` to the index definition:

```
CREATE INDEX Place.l_lon ON Place(latitude,longitude) SPATIAL ENGINE LUCENE METADATA {ignoreNullValues: true}
```

How to create a Spatial Index

The index can be created on a class that has two fields declared as `DOUBLE` (`latitude` , `longitude`) that are the coordinates of the Point.

For example we have a class `Place` with 2 double fields `latitude` and `longitude` . To create the spatial index on `Place` use this syntax.

```
CREATE INDEX Place.l_lon ON Place(latitude,longitude) SPATIAL ENGINE LUCENE
```

The Index can also be created with the Java Api. Example:

```
OSchema schema = databaseDocumentTx.getMetadata().getSchema();
OClass oClass = schema.createClass("Place");
oClass.createProperty("latitude", OType.DOUBLE);
oClass.createProperty("longitude", OType.DOUBLE);
oClass.createProperty("name", OType.STRING);
oClass.createIndex("Place.latitude_longitude", "SPATIAL", null, null, "LUCENE", new String[] { "latitude", "longitude" });
```

How to query the Spatial Index

Two custom operators has been added to query the Spatial Index:

1. `NEAR` : to find all Points near a given location (`latitude` , `longitude`)
2. `WITHIN` : to find all Points that are within a given Shape

NEAR operator

Finds all Points near a given location (`latitude` , `longitude`).

Syntax

```
SELECT FROM Class WHERE [<lat-field>,<long-field>] NEAR [lat,lon]
```

To specify `maxDistance` we have to pass a special variable in the context:

```
SELECT FROM Class WHERE [<lat-field>,<long-field>,$spatial] NEAR [lat,lon,{"maxDistance": distance}]
```

The `maxDistance` field has to be in kilometers, not radians. Results are sorted from nearest to farthest.

To know the exact distance between your Point and the Points matched, use the special variable in the context $distance.

```
SELECT *, $distance FROM Class WHERE [<lat-field>,<long-field>,$spatial] NEAR [lat,lon,{"maxDistance": distance}]
```

Examples

Let's take the example we have written before. We have a Spatial Index on Class `Place` on properties `latitude` and `longitude` .

Example: How to find the nearest Place of a given point:

```
SELECT *,$distance FROM Place WHERE [latitude,longitude,$spatial] NEAR [51.507222,-0.1275,{"maxDistance":1}]
```

WITHIN operator

Finds all Points that are within a given Shape.

❗	The current release supports only **Bounding Box** shape

Syntax

```
SELECT FROM Class WHERE [<lat field>,<long field>] WITHIN [ [ <lat1>, <lon1> ] , [ <lat2>, <lon2> ] ... ]
```

Examples

Example with previous configuration:

```
SELECT * FROM Places WHERE [latitude,longitude] WITHIN [[51.507222,-0.1275],[55.507222,-0.1275]]
```

This query will return all Places within the given Bounding Box.

Distributed Architecture

OrientDB can be distributed across different servers and used in different ways to achieve the maximum of performance, scalability and robustness.

OrientDB uses the Hazelcast Open Source project for auto-discovering of nodes, storing the runtime cluster configuration and synchronize certain operations between nodes. Some of the references in this page are linked to the Hazelcast official documentation to get more information about such topic.

> **NOTE**: When you run in distributed mode, OrientDB needs more RAM. The minimum is 2GB of heap, but we suggest to use at least 4GB of heap memory. To change the heap modify the Java memory settings in the file `bin/server.sh` (or server.bat on Windows).

Main topics

- Distributed Architecture Lifecycle
- Configure the Cluster of servers
- Replication of databases
- Sharding
- Data Centers
- Tutorial to setup a distributed database
- Tuning

Basic concepts

Server roles

OrientDB has a multi-master distributed architecture (called also as "master-less") where each server can read and write. Starting from v2.1, OrientDB support the role of "REPLICA", where the server is in read-only mode, accepting only idempotent commands, like Reads and Query. Furthermore when the server joins the distributed cluster as "REPLICA", own record clusters are not created like does the "MASTER" nodes.

Starting from v2.2, the biggest advantage of having many REPLICA servers is that they don't concur in `writeQuorum`, so if you have 3 MASTER servers and 100 REPLICA servers, every write operation will be replicated across 103 servers, but the majority of the `writeQuorum` would be just 2, because given N/2+1, N is the number of MASTER servers. In this case after the operation is executed locally, the server coordinator of the write operation has to wait only for one more MASTER server.

Cluster Ownership

When new records (documents, vertices and edges) are created in distributed mode, the RID is assigned by following the "cluster locality", where every server defines a "own" record cluster where it is able to create records. If you have the class `Customer` and 3 server nodes (node1, node2, node3), you'll have these clusters (names can be different):

- `customer` with id=#15 (this is the default one, assigned to node1)
- `customer_node2` with id=#16
- `customer_node3` with id=#17

So if you create a new Customer on node1, it will get the RID with cluster-id of "customer" cluster: #15. The same operation on node2 will generate a RID with cluster-id=16 and 17 on node3. In this way RID never collides and each node can be a master on insertion without any conflicts, because each node manages own RIDs. Starting from v2.2, if a node has more than one cluster per class, a round robin strategy is used to balance the assignment between all the available local clusters.

Ownership configuration is stored in the default-distributed-db-config.json file. By default the server owner of the cluster is the first in the list of servers. For example with this configuration:

```
"client_usa": {
  "servers" : [ "usa", "europe", "asia" ]
},
"client_europe": {
  "servers" : [ "europe", "asia", "usa" ]
}
```

The server node "usa" is the owner for cluster `client_usa` , so "usa" is the only server can create records on such cluster. Since every server node has own cluster per class, every node is able to create records, but on different clusters. Since the record clusters are part of a class, when the user executes a `INSERT INTO client SET name = "Jay"` , the local cluster is selected automatically by OrientDB to store the new "client" record. If this INSERT operation is executed on the server "usa", the "client_usa" cluster is selected. If the same operation is executed on the server "europe", then the cluster "client_europe" would be selected. The important thing is that from a logical point of view, both records from clusters "client_usa" and "client_europe" are always instances of "client" class, so if you execute the following query `SELECT * FROM client` , both record would be retrieved.

Static Owner

Starting from v2.2, you can stick a node as owner, no matter the runtime configuration. We call this "static owner". For this purpose use the `"owner" : "<NODE_NAME>"` . Example:

```
"client_usa": {
  "owner": "usa",
  "servers" : [ "usa", "europe", "asia" ]
}
```

With the configuration above, if the "usa" server is unreachable, the ownership of the cluster "client_usa" is not reassigned, so you can't create new records on that cluster until the server "usa" is back online. The static owner comes useful when you want to partition your database to be sure all the inserts come to a particular node.

Distributed transactions

OrientDB supports distributed transactions. When a transaction is committed, all the updated records are sent across all the servers, so each server is responsible to commit the transaction. In case one or more nodes fail on commit, the quorum is checked. If the quorum has been respected, then the failing nodes are aligned to the winner nodes, otherwise all the nodes rollback the transaction.

When running distributed, the transactions use a 2 phase lock like protocol, with the cool thing that everything is optimistic, so no locks between the begin and the commit, but everything is executed at commit time only.

During the commit time, OrientDB acquires locks on the touched records and check the version of records (optimistic MVCC approach). At this point this could happen:

- All the record can be locked and nobody touched the records since the beginning of the tx, so the transaction is committed. Cool.
- If somebody modified any of the records that are part of the transaction, the transaction fails and the client can retry it
- If at commit time, another transaction locked any of the same records, the transaction fails, but the retry in this case is automatic and configurable

If you have 5 servers, and writeQuorum is the majority (N/2+1 = 3), this could happen:

- All the 5 servers commit the TX: cool
- 1 or 2 servers report any error, the TX is still committed (quorum passes) and the 1 or 2 servers will be forced to have the same result as the others
- 3 servers or more have different results/errors, so the tx is rollbacked on all the servers to the initial state

What about the visibility during distributed transaction?

During the distributed transaction, in case of rollback, there could be an amount of time when the records appear changed before they are rollbacked.

Split brain network problem

OrientDB guarantees strong consistency if it's configured to have a `writeQuorum` set to a value as the majority of the number of nodes. If you have 5 nodes, it's 3, but if you have 4 nodes, it's still 3 to have a majority. While `writeQuorum` setting can be configured at database and cluster level too, it's not suggested to set a value minor than the majority of nodes, because in case of re-merge of the 2 split networks, you'd have both network partitions with updated data and OrientDB doesn't support (yet) the merging of 2 non read-only networks. So the suggestion is to always provide a `writeQuorum` with a value to, at least, the majority of the nodes.

Conflict Resolution Policy

In case of an even number of servers or when database are not aligned, OrientDB uses a Conflict Resolution Strategy chain. This default chain is defined as a global setting (`distributed.conflictResolverRepairerChain`):

```
-Ddistributed.conflictResolverRepairerChain=majority,content,version
```

The Conflict Resolution Strategy implementation are called in chain following the declaration order until a winner is selected. In the default configuration (above):

- is first checked if there is a **strict majority** for the record in terms of record versions. If the majority exists, the winner is selected
- if no strict majority was found, the **record content** is analyzed. If the majority is reached by founding a record with different versions but equal content, then that record will be the winner by using the higher version between them
- if no majority has been found with the content, then the **higher version** wins (supposing an higher version means the most update record)

OrientDB Enterprise Edition supports the additional Data Center Conflict Resolution (`dc`).

At the end of the chain, if no winner is found, the records are untouched and only a manual intervention can decide who is the winner. In this case a WARNING message is displayed in the console with text `Auto repair cannot find a winner for record <rid> and the following groups of contents: [<records>]` .

Limitations

OrientDB v2.2.x has some limitations you should notice when you work in Distributed Mode:

- In memory database is not supported.
- Importing a database while running distributed is not supported. Import the database in non-distributed mode and then run the OrientDB in distributed mode.
- With releases < v2.2.6 the creation of a database on multiple nodes could cause synchronization problems when clusters are automatically created. Please create the databases before to run in distributed mode.
- Constraints with distributed databases could cause problems because some operations are executed at 2 steps: create + update. For example in some circumstance edges could be first created, then updated, but constraints like MANDATORY and NOTNULL against fields would fail at the first step making the creation of edges not possible on distributed mode.
- Auto-Sharding is not supported in the common meaning of Distributed Hash Table (DHT). Selecting the right shard (cluster) is up to the application. This will be addressed by next releases.
- Sharded Indexes are not supported yet, so creating a UNIQUE index against a sharded class doesn't guarantee a key to be unique. This will be addressed with Auto-sharding in the further releases.
- Hot change of distributed configuration is available only in Enterprise Edition (commercial licensed).
- Not complete merging of results for all the projections when running on sharder configuration. Some functions like AVG() doesn’t work on map/reduce.

Distributed Architecture Lifecycle

In OrientDB Distributed Architecture all the nodes are masters (Multi-Master), while in most DBMS the replication works in Master-Slave mode where there is only one Master node and N Slaves that are use only for reads or when the Master is down. Starting from OrientDB v2.1, you can also assign the role of REPLICA to some nodes.

When start a OrientDB server in distributed mode (`bin/dserver.sh`) it looks for an existent cluster. If exists the starting node joins the cluster, otherwise creates a new one. You can have multiple clusters in your network, each cluster with a different "group name".

Joining a cluster

Auto discovering

At startup each Server Node sends an IP Multicast message in broadcast to discover if an existent cluster is available to join it. If available the Server Node will connect to it, otherwise creates a new cluster.

This is the default configuration contained in `config/hazelcast.xml` file. Below the multicast configuration fragment:

```
<hazelcast>
  <network>
    <port auto-increment="true">2434</port>
      <join>
        <multicast enabled="true">
          <multicast-group>235.1.1.1</multicast-group>
          <multicast-port>2434</multicast-port>
       </multicast>
     </join>
  </network>
</hazelcast>
```

If multicast is not available (typical on Cloud environments), you can use:

- Direct IPs
- Amazon EC2 Discovering

For more information look at Hazelcast documentation about configuring network.

Security

To join a cluster the Server Node has to configure the cluster group name and password in hazelcast.xml file. By default these information aren't encrypted. If you wan to encrypt all the distributed messages, configure it in hazelcast.xml file:

```
<hazelcast>
    ...
    <network>
        ...
        <!--
            Make sure to set enabled=true
            Make sure this configuration is exactly the same on
            all members
        -->
        <symmetric-encryption enabled="true">
            <!--
               encryption algorithm such as
               DES/ECB/PKCS5Padding,
               PBEWithMD5AndDES,
               Blowfish,
               DESede
            -->
            <algorithm>PBEWithMD5AndDES</algorithm>

            <!-- salt value to use when generating the secret key -->
            <salt>thesalt</salt>

            <!-- pass phrase to use when generating the secret key -->
            <password>thepass</password>

            <!-- iteration count to use when generating the secret key -->
            <iteration-count>19</iteration-count>
        </symmetric-encryption>
    </network>
    ...
</hazelcast>
```

All the nodes in the distributed cluster must have the same settings.

For more information look at: Hazelcast Encryption.

Join to an existent cluster

You can have multiple OrientDB clusters in the same network, what identifies a cluster is it’s name that must be unique in the network. By default OrientDB uses "orientdb", but for security reasons change it to a different name and password. All the nodes in the distributed cluster must have the same settings.

```
<hazelcast>
  <group>
    <name>orientdb</name>
    <password>orientdb</password>
  </group>
</hazelcast>
```

In this case Server #2 joins the existent cluster.

When a node joins an existent cluster, the most updated copy of the database is downloaded to the joining node if the distributed configuration has `autoDeploy:true`. If the node is rejoining the cluster after a disconnection, a delta backup is requested first. If not available a full backup is sent to the joining server. If it has been configured a sharded configuration, the joining node will ask for separate parts of the database to multiple available servers to reconstruct the own database copy. If any copy of database was already present, that is moved under `backup/databases` folder.

Multiple clusters

Multiple clusters can coexist in the same network. Clusters can't see each others because are isolated black boxes.

Distribute the configuration to the clients

Every time a new Server Node joins or leaves the Cluster, the new Cluster configuration is broadcasted to all the connected clients. Everybody knows the cluster layout and who has a database!

Fail over management

When a node is unreachable

When a Server Node becomes unreachable (because it's crashed, network problems, high load, etc.) the Cluster treats this event as if the Server Node left the cluster.

Starting from v2.2.13, the unreachable server is removed from the distributed configuration only if it's dynamic, that means it hasn't registered under the `servers` configuration. Once removed it doesn't concur to the quorum anymore. Instead, if the server has been registered under `servers`, it's kept in configuration waiting for a rejoining.

The `newNodeStrategy` setting specifies if a new joining node is automatically registered as static or is managed as dynamic.

Automatic switch of servers

All the clients connected to the unreachable node will switch to another server transparently without raising errors to the Application User Application doesn't know what is happening!

Re-distribute the updated configuration again

After the Server #2 left the Cluster the updated configuration is sent again to all the connected clients.

Continue with:

- Distributed Architecture
- Replication
- Tutorial to setup a distributed database

Distributed Configuration

The distributed configuration consists of 3 files under the **config/** directory:

- orientdb-server-config.xml
- default-distributed-db-config.json
- hazelcast.xml

Main topics:

- Replication
- Asynchronous replication mode
- Return distributed configuration at run-time
- Load Balancing
- Data Center support
- Cloud support

orientdb-server-config.xml

To enable and configure the clustering between nodes, add and enable the **OHazelcastPlugin** plugin. It is configured as a Server Plugin. The default configuration is reported below.

File **orientdb-server-config.xml**:

```
<handler class="com.orientechnologies.orient.server.hazelcast.OHazelcastPlugin">
  <parameters>
    <!-- NODE-NAME. IF NOT SET IS AUTO GENERATED THE FIRST TIME THE SERVER RUN -->
    <!-- <parameter name="nodeName" value="europe1" /> -->
    <parameter name="enabled" value="true" />
    <parameter name="configuration.db.default"
               value="${ORIENTDB_HOME}/config/default-distributed-db-config.json" />
    <parameter name="configuration.hazelcast"
               value="${ORIENTDB_HOME}/config/hazelcast.xml" />
  </parameters>
</handler>
```

Where:

Parameter	Description
enabled	To enable or disable the plugin: `true` to enable it, `false` to disable it. By default is `true`
nodeName	An optional alias identifying the current node within the cluster. When omitted, a default value is generated as node, example: "node239233932932". By default is commented, so it's automatic generated
configuration.db.default	Path of default distributed database configuration. By default is `${ORIENTDB_HOME}/config/default-distributed-db-config.json`
configuration.hazelcast	Path of Hazelcast configuration file, default is `${ORIENTDB_HOME}/config/hazelcast.xml`

default-distributed-db-config.json

This is the JSON file containing the default configuration for distributed databases. The first time a database run in distributed version this file is copied in the database's folder with name `distributed-config.json`. Every time the cluster shape changes the database specific file is changed. To restore distributed database settings, remove the file `distributed-config.json` from the database folder, and the `default-distributed-db-config.json` file will be used.

Default **default-distributed-db-config.json** file content:

```
{
    "autoDeploy": true,
    "executionMode": "undefined",
    "readQuorum": 1,
    "writeQuorum": "majority",
    "readYourWrites": true,
    "newNodeStrategy": "static",
    "dataCenters": {
      "rome": {
        "writeQuorum": "majority",
        "servers": [ "europe-0", "europe-1", "europe-2" ]
      },
      "denver": {
        "writeQuorum": "majority",
        "servers": [ "usa-0", "usa-1", "usa-2" ]
      }
    },
    "servers": {
        "*": "master"
    },
    "clusters": {
        "internal": {
        },
        "product": {
          "servers": ["usa", "china"]
        },
        "employee_usa": {
          "owner": "usa",
          "servers": ["usa", "<NEW_NODE>"]
        },
        "*": { "servers" : [ "<NEW_NODE>" ] }
    }
}
```

Where:

Parameter	Description	Default value
autoDeploy	Whether to deploy the database to any joining node that does not have it. It can be `true` or `false`	`true`
executionMode	It can be `undefined` to let to the client to decide per call execution between synchronous (default) or asynchronous. `synchronous` forces synchronous mode, and `asynchronous` forces asynchronous mode	`undefined`
readQuorum	On "read" operation (record read, query and traverse) this is the number of responses to be coherent before sending the response to the client. Set to 1 if you don't want this check at read time	`1`
writeQuorum	On "write" operation (any write on database) this is the number of responses to be coherent before sending the response to the client. Set to 1 if you don't want this check at write time. Suggested value is "majority", the default, that means N/2+1 where N is the number of available nodes. In this way the quorum is reached only if the majority of nodes are coherent. "all" means all the available nodes. Starting from v2.2, N represent the MASTER only servers. For more information look at Server Roles.	`"majority"`
readYourWrites	Whether the write quorum is satisfied only when also the local node responded. This assures current the node can read its writes. Disable it to improve replication performance if such consistency is not important. Can be `true` or `false`	`true`
newNodeStrategy	Strategy to use when a new node joins the cluster. Default is `static` that means the server is automatically registered under `servers` list in configuration. If it is `dynamic` , then the node is not registered. This affects the strategy when a node is unreachable. If it is not registered (dynamic), it is removed from the configuration, so it does not concur in the quorum. Available since v2.2.13	`static`
dataCenters	(Since v2.2.4) Optional (and only available in the Enterprise Edition, contains the definition of the data centers. For more information look at Data Centers	-
servers	(Since v2.1) Optional, contains the map of server roles in the format `server-name` : `role` . `*` means any server. Available roles are "MASTER" (default) and "REPLICA". For more information look at Server roles	-
clusters	if the object containing the clusters' configuration as map `cluster-name` : `cluster-configuration` . `*` means all the clusters and is the cluster's default configuration	-

The **cluster** configuration inherits database configuration, so if you declare "writeQuorum" at database level, all the clusters will inherit that setting unless they define your own. Settings can be:

Parameter	Description	Default value
readQuorum	On "read" operation (record read, query and traverse) is the number of responses to be coherent before to send the response to the client. Set to 1 if you don't want this check at read time	`1`
writeQuorum	On "write" operation (any write on database) is the number of responses to be coherent before to send the response to the client. Set to 1 if you don't want this check at write time. Suggested value is "majority", the default, that means N/2+1 where N is the number of available nodes. In this way the quorum is reached only if the majority of nodes are coherent. "all" means all the available nodes. Starting from v2.2, N represent the MASTER only servers. For more information look at Server Roles.	`"majority"`
readYourWrites	The write quorum is satisfied only when also the local node responded. This assure current the node can read its writes. Disable it to improve replication performance if such consistency is not important. Can be `true` or `false`	`true`
owner	By default the cluster owner is assigned at run-time and it's placed as first server in the `servers` list. If `owner` field is defined, the server owner is statically assigned to that server, no matter if the server is available or not.	
servers	Is the array of servers where to store the records of cluster	empty for internal and index clusters and `[ "<NEW_NODE>" ]` for cluster `*` representing any cluster

`"<NEW_NODE>"` is a special tag that put any new joining node name in the array.

Default configuration

In the default configuration all the record clusters are replicated, but `internal`. Every node that joins the cluster shares all the rest of the clusters ("*" settings). Since "readQuorum" is 1 all the reads are executed on the first available node where the local node is preferred if own the requested record. "writeQuorum" to "majority" means that all the changes are in at least N/2+1 nodes, where N is the number of available nodes. Using "majority" instead of a number, allows you to have an elastic configuration where nodes can join and left without the need to rebalance this value manually.

100% Asynchronous Writes

By default writeQuorum is "majority". This means that it waits and checks the answer from at least N/2+1 (where N is the number of available nodes) nodes before to send the ACK to the client. After the quorum is reached, the responses are managed asyncrhonously. For example, with 3 nodes and `writeQuorum: "majority"`, then after the 2nd node's answer, the collecting and check of the 3rd node answer is managed asynchronously. You could also set this to 1 to have all the writes asynchronous.

Full Consistency

To avoid experiencing any dirty reads during the replication, you can setup the writeQuorum to "all", that means all the available nodes. Example: `writeQuorum: "all". Setting writeQuorum to "all" means each replication operation will take the time of the slowest node in the cluster.

hazelcast.xml

A OrientDB cluster is composed by two or more servers that are the **nodes** of the cluster. All the server nodes that want to be part of the same cluster must to define the same Cluster Group. By default "orientdb" is the group name. Look at the default **config/hazelcast.xml** configuration file reported below:

```
<?xml version="1.0" encoding="UTF-8"?>
<hazelcast xsi:schemaLocation="http://www.hazelcast.com/schema/config hazelcast-config-3.0.xsd"
           xmlns="http://www.hazelcast.com/schema/config" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <group>
    <name>orientdb</name>
    <password>orientdb</password>
  </group>
  <network>
    <port auto-increment="true">2434</port>
    <join>
      <multicast enabled="true">
        <multicast-group>235.1.1.1</multicast-group>
        <multicast-port>2434</multicast-port>
      </multicast>
    </join>
  </network>
  <executor-service>
    <pool-size>16</pool-size>
  </executor-service>
</hazelcast>
```

NOTE: Change the name and password of the group to prevent foreign nodes from joining it!

Network configuration

Automatic discovery in LAN using Multicast

OrientDB by default uses TCP Multicast to discover nodes. This is contained in **config/hazelcast.xml** file under the **network** tag. This is the default configuration:

```
<hazelcast>
  ...
  <network>
    <port auto-increment="true">2434</port>
    <join>
      <multicast enabled="true">
        <multicast-group>235.1.1.1</multicast-group>
        <multicast-port>2434</multicast-port>
      </multicast>
     </join>
  </network>
  ...
</hazelcast>
```

Manual IP

When Multicast is disabled or you prefer to assign Hostnames/IP-addresses manually use the TCP/IP tag in configuration. Pay attention to disable the **multicast**:

```
<hazelcast>
  ...
  <network>
    <port auto-increment="true">2434</port>
    <join>
      <multicast enabled="false">
        <multicast-group>235.1.1.1</multicast-group>
        <multicast-port>2434</multicast-port>
      </multicast>
      <tcp-ip enabled="true">
        <member>europe0:2434</member>
        <member>europe1:2434</member>
        <member>usa0:2434</member>
        <member>asia0:2434</member>
        <member>192.168.1.0-7:2434</member>
      </tcp-ip>
     </join>
  </network>
  ...
</hazelcast>
```

For more information look at: Hazelcast Config TCP/IP.

Cloud support

Since multicast is disabled on most of the Cloud stacks, you have to change the **config/hazelcast.xml** configuration file based on the Cloud used.

Amazon EC2

OrientDB supports natively Amazon EC2 through the Hazelcast's Amazon discovery plugin. In order to use it include also the **hazelcast-aws.jar** library under the **lib/** directory.

```
<hazelcast>
  ...
    <join>
      <multicast enabled="false">
        <multicast-group>235.1.1.1</multicast-group>
        <multicast-port>2434</multicast-port>
      </multicast>
      <aws enabled="true">
        <access-key>my-access-key</access-key>
        <secret-key>my-secret-key</secret-key>
        <region>us-west-1</region>                              <!-- optional, default is us-east-1 -->
        <host-header>ec2.amazonaws.com</host-header>            <!-- optional, default is ec2.amazonaws.com. If set region
                                                                     shouldn't be set as it will override this property -->
        <security-group-name>hazelcast-sg</security-group-name>  <!-- optional -->
        <tag-key>type</tag-key>                                 <!-- optional -->
        <tag-value>hz-nodes</tag-value>                         <!-- optional -->
      </aws>
    </join>
  ...
</hazelcast>
```

For more information look at Hazelcast AWS docs.

Other Cloud providers

Uses manual IP like explained in Manual IP.

Asynchronous replication mode

If you are replication OrientDB database across multiple Data Centers, look at Data Centers Configuration available only with the Enterprise Edition.

If you are using the Community Edition or if you don't have multiple data centers, but just a network with high latency, in order to reduce the impact of the latency in the replication, the suggested configuration is to set `executionMode` to "asynchronous". In asynchronous mode any operation is executed on local node and then replicated. In this mode the client doesn't wait for the quorum across all the servers, but receives the response immediately after the local node answer. Example:

```
{
    "autoDeploy": true,
    "executionMode": "asynchronous",
    "readQuorum": 1,
    "writeQuorum": "majority",
    "readYourWrites": true,
    "newNodeStrategy": "static",
    "servers": {
        "*": "master"
    },
    "clusters": {
        "internal": {
        },
        "*": {
            "servers" : [ "<NEW_NODE>" ]
        }
    }
}
```

Starting from v2.1.6 is possible to catch events of command during asynchronous replication, thanks to the following method of OCommandSQL:

- `onAsyncReplicationOk()` , to catch the event when the asynchronous replication succeed
- `onAsyncReplicationError()` , to catch the event when the asynchronous replication returns error

Example retrying up to 3 times in case of concurrent modification exception on creation of edges:

```
g.command( new OCommandSQL("create edge Own from (select from User) to (select from Post)")
 .onAsyncReplicationError(new OAsyncReplicationError() {
  @Override
  public ACTION onAsyncReplicationError(Throwable iException, int iRetry) {
    System.err.println("Error, retrying...");
    return iException instanceof ONeedRetryException && iRetry<=3 ? ACTION.RETRY : ACTION.IGNORE;
  }
})
 .onAsyncReplicationOk(new OAsyncReplicationOk() {
   System.out.println("OK");
 }
).execute();
```

Load Balancing

(Since v2.2) OrientDB allows to do load balancing when you have multiple servers connected in cluster. Below are the available connection strategies:

- `STICKY` , the default, where the client remains connected to a server until the close of database
- `ROUND_ROBIN_CONNECT` , at each connect, the client connects to a different server between the available ones
- `ROUND_ROBIN_REQUEST` , at each request, the client connects to a different server between the available ones. Pay attention on using this strategy if you're looking for strong consistency. In facts, in case the writeQuorum is minor of the total nodes available, a client could have executed an operation against another server and current operation cannot see updates because wasn't propagated yet.

Once a client is connected to any server node, it retrieves the list of available server nodes. In case the connected server becomes unreachable (crash, network problem, etc.), the client automatically connects to the next available one.

To setup the strategy using the Java Document API:

```
final ODatabaseDocumentTx db = new ODatabaseDocumentTx("remote:localhost/demo");
db.setProperty(OStorageRemote.PARAM_CONNECTION_STRATEGY,
      OStorageRemote.CONNECTION_STRATEGY.ROUND_ROBIN_CONNECT.toString());
db.open(user, password);
```

To setup the strategy using the Java Graph API:

```
final OrientGraphFactory factory = new OrientGraphFactory("remote:localhost/demo");
factory.setConnectionStrategy(OStorageRemote.CONNECTION_STRATEGY.ROUND_ROBIN_CONNECT.toString());
OrientGraphNoTx graph = factory.getNoTx();
```

Use multiple addresses

If the server addresses are known, it is good practice to connect the clients to a set of URLs, instead of just one. You can separate hosts/addresses by using a semicolon (;). OrientDB client will try to connect to the addresses in order. Example: `remote:server1:2424;server2:8888;server3/mydb` .

Use the DNS

Before v2.2, the simplest and most powerful way to achieve load balancing seems to use some hidden (to some) properties of DNS. The trick is to create a TXT record listing the servers.

The format is:

```
v=opf<version> (s=<hostname[:<port>]> )*
```

Example of TXT record for domain **dbservers.mydomain.com**:

```
v=opf1 s=192.168.0.101:2424 s=192.168.0.133:2424
```

In this way if you open a database against the URL `remote:dbservers.mydomain.com/demo` the OrientDB client library will try to connect to the address **192.168.0.101** port 2424. If the connection fails, then the next address **192.168.0.133:** port 2424 is tried.

To enable this feature in Java Client driver set `network.binary.loadBalancing.enabled=true` :

```
java ... -Dnetwork.binary.loadBalancing.enabled=true
```

or via Java code:

```
OGlobalConfiguration.NETWORK_BINARY_DNS_LOADBALANCING_ENABLED.setValue(true);
```

Troubleshooting

Hazelcast Monitor

Users reported that Hazelcast Health Monitoring could cause problem with a JVM kill (OrientDB uses Hazelcast to manage replication between nodes). By default this setting is OFF, so if you are experiencing this kind of problem assure this is set: `hazelcast.health.monitoring.level=OFF`

Extraction Directory

OrientDB extract the database form the network to the temporary directory set in the JVM by `java.io.tmpdir` setting. To change the temporary directory used to store the temporary database, overwrite the default setting at JVM startup (or even at run-time from Java). Example:

```
java ... -Djava.io.tmpdir=/myfolder/server1
```

OrientDB will create the temporary database under the folder `/myfolder/server1/orientdb` .

History

v2.2.13

New setting `newNodeStrategy` to specify what happens when a new node joins. This affects the behaviour when a node is unreachable. If it is not present under the `servers` list, then it's removed from the configuration and it doesn't concur in the quorum anymore.

v2.2

The intra-node communication is not managed with Hazelcast Queues anymore, but rather through the OrientDB binary protocol. This assure better performance and avoid the problem of locality of the Hazelcast queues. Release v2.2 also supported the wildcard "majority" and "all" for the read and write quorums. Introduced also the concept of static cluster owner. Furthermore nodes are not removed by configuration when they are offline.

Introduced Load balancing at client level. For more information look at load balancing.

v1.7

Simplified configuration by moving. Removed some flags (replication:boolean, now it’s deducted by the presence of “servers” field) and settings now are global (autoDeploy, hotAlignment, offlineMsgQueueSize, readQuorum, writeQuorum, failureAvailableNodesLessQuorum, readYourWrites), but you can overwrite them per-cluster.

For more information look at News in 1.7.

Distributed Architecture Plugin

Java class: `com.orientechnologies.orient.server.hazelcast.OHazelcastPlugin`

Introduction

This is part of Distributed Architecture. Configure a distributed clustered architecture. This task is configured as a Server handler. The task can be configured easily by changing these parameters:

- **enabled**: Enable the plugin: `true` to enable, `false` to disable it.
- **configuration.hazelcast**: The location of the Hazelcast configuration file (`hazelcast.xml`).
- **alias**: An alias for the current node within the cluster name. Default value is the IP address and port for OrientDB on this node.
- **configuration.db.default**: The location of a file that describes, using JSON syntax, the synchronization configuration of the various clusters in the database.

Default configuration in orientdb-server-config.xml:

```
<handler class="com.orientechnologies.orient.server.hazelcast.OHazelcastPlugin">
   <parameters>
      <!-- <parameter name="alias" value="europe1" /> -->
      <parameter name="enabled" value="true" />
      <parameter name="configuration.db.default" value="${ORIENTDB_HOME}/config/default-distributed-db-config.json" />
      <parameter name="configuration.hazelcast" value="${ORIENTDB_HOME}/config/hazelcast.xml" />
   </parameters>
</handler>
```

Distributed runtime

NOTE: available only in Enteprise Edition

Node status

To retrieve the distributed configuration of a OrientDB server, execute a HTTP GET operation against the URL `http://<server>:<port>/distributed/node`. Example:

```
curl -u root:root "http://localhost:2480/distributed/node"
```

Result:

```
{
    "localId": "9e20f766-5f8c-4a5c-a6a2-7308019db702",
    "localName": "_hzInstance_1_orientdb",
    "members": [
        {
            "databases": [],
            "id": "b7888b58-2b26-4098-bb4d-8e23a5050b68",
            "listeners": [
                {
                    "listen": "10.0.1.8:2425",
                    "protocol": "ONetworkProtocolBinary"
                },
                {
                    "listen": "10.0.1.8:2481",
                    "protocol": "ONetworkProtocolHttpDb"
                }
            ],
            "name": "node2",
            "startedOn": "2015-09-28 13:19:09:267"
        },
        {
            "databases": [],
            "id": "9e20f766-5f8c-4a5c-a6a2-7308019db702",
            "listeners": [
                {
                    "listen": "10.0.1.8:2424",
                    "protocol": "ONetworkProtocolBinary"
                },
                {
                    "listen": "10.0.1.8:2480",
                    "protocol": "ONetworkProtocolHttpDb"
                }
            ],
            "name": "node1",
            "startedOn": "2015-09-28 12:58:11:819"
        }
    ]
}
```

Database configuration

To retrieve the distributed configuration for a database, execute a HTTP GET operation against the URL `http://<server>:<port>/distributed/database/<database-name>`. Example:

```
curl -u root:root "http://localhost:2480/distributed/database/GratefulDeadConcerts"
```

Result:

```
{
    "autoDeploy": true,
    "clusters": {
        "*": {
            "servers": [
                "node1",
                "node2",
                "<NEW_NODE>"
            ]
        },
        "v": {
            "servers": [
                "node2",
                "node1",
                "<NEW_NODE>"
            ]
        }
    },
    "executionMode": "undefined",
    "failureAvailableNodesLessQuorum": false,
    "hotAlignment": false,
    "readQuorum": 1,
    "readYourWrites": true,
    "servers": {
        "*": "master"
    },
    "version": 21,
    "writeQuorum": 2
}
```

Queues

OrientDB uses distributed queues to exchange messages between OrientDB servers. To have metrics about queues, execute a HTTP GET operation against the URL `http://<server>:<port>/distributed/queue/<queue-name>` . Use `*` as queue name to return stats for all he queues. Example:

```
curl -u root:root "http://localhost:2480/distributed/queue/*"
```

Result:

```
{
    "queues": [
        {
            "avgAge": 0,
            "backupItemCount": 0,
            "emptyPollOperationCount": 0,
            "eventOperationCount": 0,
            "maxAge": 0,
            "minAge": 0,
            "name": "orientdb.node.node1.benchmark.insert.request",
            "nextMessages": [],
            "offerOperationCount": 0,
            "otherOperationsCount": 0,
            "ownedItemCount": 0,
            "partitionKey": "orientdb.node.node1.benchmark.insert.request",
            "pollOperationCount": 0,
            "rejectedOfferOperationCount": 0,
            "serviceName": "hz:impl:queueService",
            "size": 0
        },
        {
            "avgAge": 1,
            "backupItemCount": 0,
            "emptyPollOperationCount": 0,
            "eventOperationCount": 0,
            "maxAge": 1,
            "minAge": 1,
            "name": "orientdb.node.node2.response",
            "nextMessages": [],
            "offerOperationCount": 60,
            "otherOperationsCount": 12,
            "ownedItemCount": 0,
            "partitionKey": "orientdb.node.node2.response",
            "pollOperationCount": 60,
            "rejectedOfferOperationCount": 0,
            "serviceName": "hz:impl:queueService",
            "size": 0
        },
        {
            "avgAge": 0,
            "backupItemCount": 0,
            "emptyPollOperationCount": 0,
            "eventOperationCount": 0,
            "maxAge": 0,
            "minAge": 0,
            "name": "orientdb.node.node2.benchmark.request",
            "nextMessages": [],
            "offerOperationCount": 0,
            "otherOperationsCount": 0,
            "ownedItemCount": 0,
            "partitionKey": "orientdb.node.node2.benchmark.request",
            "pollOperationCount": 0,
            "rejectedOfferOperationCount": 0,
            "serviceName": "hz:impl:queueService",
            "size": 0
        },
        {
            "avgAge": 1,
            "backupItemCount": 0,
            "emptyPollOperationCount": 0,
            "eventOperationCount": 0,
            "maxAge": 1,
            "minAge": 1,
            "name": "orientdb.node.node1.GratefulDeadConcerts.request",
            "nextMessages": [],
            "offerOperationCount": 44,
            "otherOperationsCount": 53,
            "ownedItemCount": 0,
            "partitionKey": "orientdb.node.node1.GratefulDeadConcerts.request",
            "pollOperationCount": 44,
            "rejectedOfferOperationCount": 0,
            "serviceName": "hz:impl:queueService",
            "size": 0
        }
    ]
}
```

Replication

OrientDB supports the Multi Master replication. This means that all the nodes in the cluster are Master and are able to read and write to the database. This allows to scale up horizontally without bottlenecks like most of any other RDBMS and NoSQL solution do.

Replication works only in the Distributed-Architecture.

Sharing of database

In Distributed Architecture the replicated database must have the same name. When an OrientDB Server is starting, it sends the list of current databases (all the databases located under `$ORIENTDB_HOME/databases` directory) to all the nodes in the cluster. If other nodes have databases with the same name, a replication is automatically set.

NOTE: In Distributed Architecture assure to avoid conflict with database names, otherwise 2 different databases could start replication with the chance to get corrupted.

If the database configuration has the setting `"autoDeploy" : true`, then the databases are automatically deployed across the network to the other nodes as soon as they join the cluster.

Server unreachable

In case a server becomes unreachable, the node is removed by database configuration unless the setting `"hotAlignment" : true`. In this case all the new synchronization messages are kept in a distributed queue.

As soon as the Server becomes online again, it starts the synchronization phase (status=SYNCHRONIZING) by polling all the synchronization messages in the queue.

Once the alignment is finished, the node becomes online (status=ONLINE) and the replication continues like at the beginning.

Further readings

Continue with:

- Distributed Architecture
- Distributed Sharding
- Distributed database configuration

Sharding

> **NOTE**: Sharding is a new feature with some limitations. Please read them before using it.

OrientDB supports sharding of data at class level, by using multiple clusters per class, where each cluster has own list of server where data is replicated. From a logical point of view all the records stored in clusters that are part of the same class, are records of that class.

Follows an example that split the class “Client” in 3 clusters:

Class **Client** -> Clusters [`client_usa` , `client_europe` , `client_china`]

This means that OrientDB will consider any record/document/graph element in any of such clusters as “Clients” (Client class relies on such clusters). In Distributed-Architecture each cluster can be assigned to one or multiple server nodes.

Shards, based on clusters, work against indexed and non-indexed class/clusters.

Multiple servers per cluster

You can assign each cluster to one or more servers. If more servers are enlisted then records will be copied across all of the servers. This is similar to what RAID does for Disks. The first server in the list will be the **master server** for that cluster.

For example, consider a configuration where the Client class has been split in the 3 clusters client_usa, client_europe and client_china. Each cluster might have a different configuration:

- `client_usa` , will be managed by the ”usa" and "europe" nodes
- `client_europe` , will be managed only by the node, ”europe"
- `client_china` , will be managed by all of the nodes (it would be equivalent as writing `“<NEW_NODE>”` , see cluster "*", the default one)

Configuration

In order to keep things simple, the entire OrientDB Distributed Configuration is stored on a single JSON file. Example of distributed database configuration for (Multiple servers per cluster)[Distributed-Sharding.md#Multiple-servers-per-cluster] use case:

```
{
  "autoDeploy": true,
  "readQuorum": 1,
  "writeQuorum": "majority",
  "executionMode": "undefined",
  "readYourWrites": true,
  "newNodeStrategy": "dynamic",
  "servers": {
    "*": "master"
  },
  "clusters": {
    "internal": {
    },
    "client_usa": {
      "servers" : [ "usa", "europe" ]
    },
    "client_europe": {
      "servers" : [ "europe" ]
    },
    "client_china": {
      "servers" : [ "china", "usa", "europe" ]
    },
    "*": {
      "servers" : [ "<NEW_NODE>" ]
    }
  }
}
```

Cluster Locality

OrientDB automatically creates a new cluster per each class as soon a new insert operation is performed on a class where the local server has no ownership. When a node goes down, the clusters where the node was master are reassigned to other servers. As soon as that node returns up and running, OrientDB will reassign the ownership of a cluster to the server specified in the `owner` property if specified, otherwise to the server name indicated as suffix of the cluster (`<class>_<node>`). For example the cluster `customer_usa` will be assigned automatically to the server `usa` if exists in the cluster.

This is defined as "Cluster Locality". The local node is always selected when a new record is created. This avoids conflicts and allows to insert record in parallel on multiple nodes. This means also that in distributed mode you can't select the cluster selection strategy, because "local" strategy is always injected to all the cluster automatically.

If you want to change permanently the mastership of clusters, rename the cluster with the suffix of the node you want assign as master.

CRUD Operations

Create new records

In the configuration above, if a new Client record is created on node USA, then the selected cluster will be `client_usa` , because it's the local cluster for class Client. Now, `client_usa` is managed by both USA and EUROPE nodes, so the "create record" operation is sent to both "usa" (locally) and "europe" nodes.

Update and Delete of records

Updating and Deleting of single records always involves all the nodes where the record is stored. No matter the node that receives the update operation. If we update record `#13:22` that is stored on cluster `13` , namely `client_china` in the example above, then the update is sent to nodes: "china", "usa", "europe".

Instead, SQL UPDATE and DELETE commands cannot run distributed in case of sharding. For example, the SQL command `delete from vertex Client` works if the `Client` class is replicated across all the sevrers, but it won't work if the class is sharded by throwing a `ODistributedOperationException` exception with the following reason: "because it is not idempotent and a map-reduce has been requested".

If you want to execute a distributed update or delete, you can still execute a distributed SQL SELECT against all the shards, and then update or delete every single records in your code. Example to delete all the clients, no matter where they are located:

```
Iterable<OrientVertex> res = g.command(new OCommandSQL("select from Client")).execute();
for (OrientVertex v : res) {
  v.remove();
}
```

NOTE: if something happens during the iteration of the resultset, you could end up with only a few records deleted.

Read records

If the local node has the requested record, the record is read directly from the storage. If it's not present on local server, a forward is executed to any of the nodes that have the requested record. This means a network call to between nodes.

In case of queries, OrientDB checks where the query target are located and send the query to all the involved servers. This operation is equivalent to a Map-Reduce. If the query target is 100% managed on local node, the query is simply executed on local node without paying the cost of network call.

All the query works by aggregating the result sets from all the involved nodes.

Example of executing this query on node "usa":

```
SELECT FROM Client
```

Since local node (USA) already owns `client_usa` and `client_china`, 2/3 of data are local. The missing 1/3 of data is in `client_europe` that is managed only by node "Europe". So the query will be executed on local node "usa" and "Europe" providing the aggregated result back to the client.

You can query also a particular cluster:

```
SELECT FROM CLUSTER:client_china
```

In this case the local node (USA) is used, because `client_china` is hosted on local node.

MapReduce

OrientDB supports MapReduce without Hadoop or external framework like Spark (even if there is a connector), but rather by using the OrientDB SQL. The MapReduce operation is totally transparent to the developer. When a query involve multiple shards (clusters), OrientDB executes the query against all the involved server nodes (Map operation) and then merge the results (Reduce operation). Example:

```
SELECT MAX(amount), COUNT(*), SUM(amount) FROM Client
```

In this case the query is executed across all the 3 nodes and then filtered again on starting node.

Define the target cluster/shard

The application can decide where to insert a new Client by passing the cluster number or name. Example:

```
INSERT INTO CLUSTER:client_usa SET @class = 'Client', name = 'Jay'
```

If the node that executes this command is not the master of cluster `client_usa`, an exception is thrown.

Java Graph API

```
OrientVertex v = graph.addVertex("class:Client,cluster:client_usa");
v.setProperty("name", "Jay");
```

Java Document API

```
ODocument doc = new ODocument("Client");
doc.field("name", "Jay");
doc.save( "client_usa" );
```

Sharding and Split brain network problem

OrientDB guarantees strong consistency if it's configured to have a `writeQuorum` to the majority of the nodes. For more information look at Split Brain network problem. In case of Sharding you could have a situation where you'd need a relative `writeQuorum` to a certain partition of your data. While `writeQuorum` setting can be configured at database and cluster level too, it's not suggested to set a value minor than the majority, because in case of re-merge of the 2 split networks, you'd have both network partitions with updated data and OrientDB doesn't support (yet) the merging of 2 non read-only networks. So the suggestion is to always provide a `writeQuorum` at least at the majority of nodes, even with sharded configuration.

Limitation

1. *Auto-Sharding* is not supported in the common meaning of Distributed Hash Table (DHT). Selecting the right shard (cluster) is up to the application. This will be addressed by next releases
2. Sharded Indexes are not supported.
3. If `hotAlignment=false` is set, when a node re-joins the cluster (after a failure or simply unreachability) the full copy of database from a node could have no all information about the shards.
4. Hot change of distributed configuration not available. This will be introduced at release 2.0 via command line and in visual way in the Workbench of the Enterprise Edition (commercial licensed)
5. Not complete merging of results for all the projections. Some functions like AVG() doesn’t work on map/reduce
6. Backup doesn't work on distributed nodes yet, so doing a backup of all the nodes to get all the shards is a manual operation in charge to the user

Indexes

All the indexes are managed locally to a server. This means that if a class is spanned across 3 clusters on 3 different servers, each server will have own local indexes. By executing a distributed query (Map/Reduce like) each server will use own indexes.

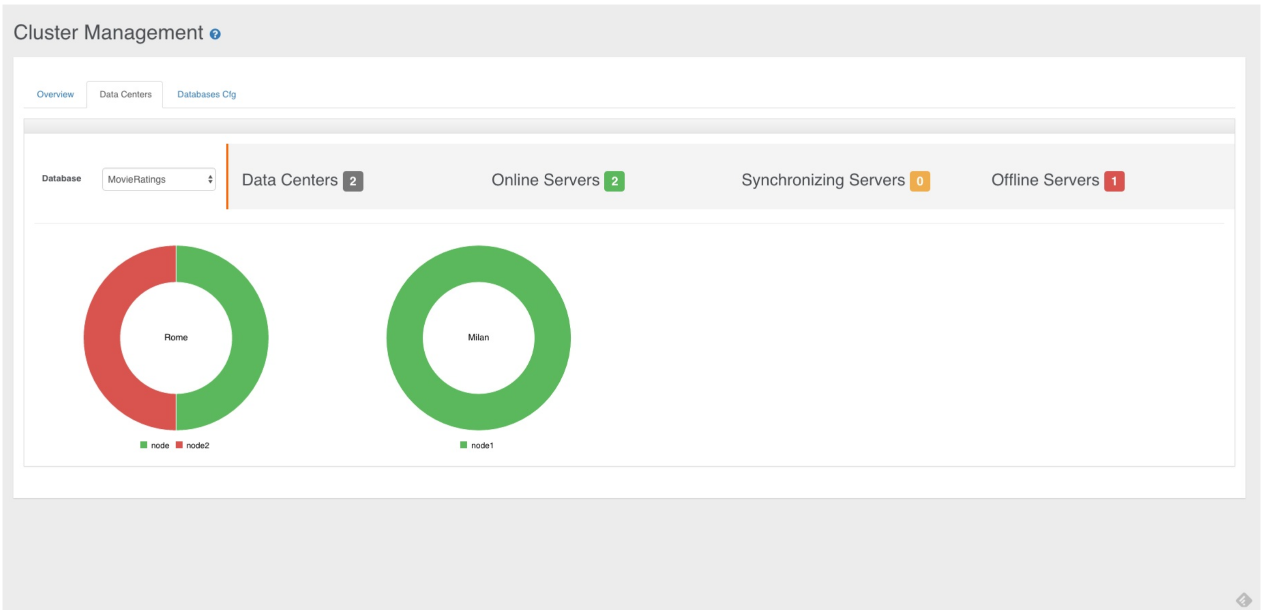
Hot management of distributed configuration

With Community Edition the distributed configuration cannot be changed at run-time but you have to stop and restart all the nodes. Enterprise Edition allows to create and drop new shards without stopping the distributed cluster.

By using Enterprise Edition and the Workbench, you can deploy the database to the new server and defining the cluster to assign to it. In this example a new server "usa2" is created where only the cluster `client_usa` will be copied. After the deployment, cluster `client_usa` will be replicated against nodes "usa" and "usa2".

Data Centers

Starting from OrientDB Enterprise Edition v2.2.4, you can define how your servers are deployed in multiple **Data Centers**.



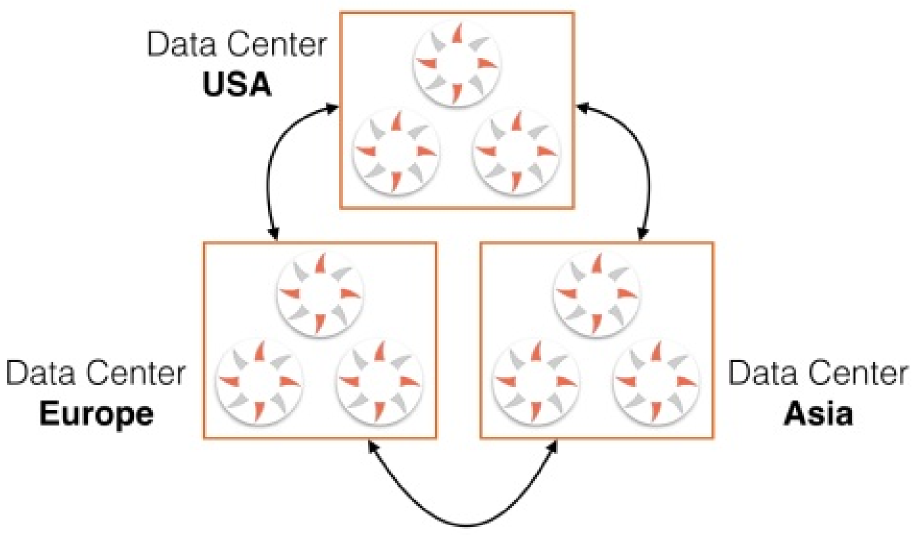
All you need is to use the tag `"dataCenters"` in your `default-distributed-config.json` configuration file. This is the format:

```
"dataCenters": {
  "<data-center1-name>": {
    "writeQuorum": "<data-center1-quorum>",
    "servers": [ "<data-center1-server1>", "<data-center1-server2>", "<data-center1-serverN>" ]
  },
  "<data-center2-name>": {
    "writeQuorum": "<data-center2-quorum>",
    "servers": [ "<data-center2-server1>", "<data-center2-server2>", "<data-center2-serverN>" ]
  },
},
```

> **NOTE:** This feature is available only in the OrientDB Enterprise Edition. If you are interested in a commercial license look at OrientDB Subscription Packages.

Example:

```
"dataCenters": {
  "usa": {
    "writeQuorum": "majority",
    "servers": [ "<austin>", "<paloalto>", "<newyork>" ]
  },
  "europe": {
    "writeQuorum": "majority",
    "servers": [ "<rome>", "<dublin>", "<london>" ]
  },
  "asia": {
    "writeQuorum": "majority",
    "servers": [ "<tokyo>", "<singapore>", "<hongkong>" ]
  }
},
```



Write Quorum

The most common reason for defining data centers is to be able to set the consistency level per data center. A typical scenario is to have synchronous replication between the servers in the same data center where the coordinator server is located and then to propagate changes to the other data centers asynchronously. In this way you can avoid the cost of the replication latency of the servers located at different data centers. In order to activate this mode, set the global `"writeQuorum": "localDataCenter"` and then specify a writeQuorum per data center.

For example, if a write operation is executed by a server where its data center's write quorum setting is `majority`, then the used quorum will be `majority` between only the servers located in the same data center.

Example about the configuration of 2 data centers, "rome" and "austin", with respectively 3 and 2 servers.

```
{
  "autoDeploy": true,
  "readQuorum": 1,
  "writeQuorum": "localDataCenter",
  "readYourWrites": true,
  "dataCenters": {
    "rome": {
      "writeQuorum": "majority",
      "servers": [ "europe-0", "europe-1", "europe-2" ]
    },
    "austin": {
      "writeQuorum": "majority",
      "servers": [ "usa-0", "usa-1" ]
    }
  },
  "servers": { "*": "master" },
  "clusters": {
    "internal": {},
    "*": { "servers": [ "<NEW_NODE>" ] }
  }
}
```

If a write operation is executed on the server "europe-0", the quorum used will be `majority`, but only between the servers located in the same data center: namely "europe-0" (the coordinator), "europe-1", and "europe-2". Since the coordinator writes in the database before distributing the operation, the write operation succeeds as soon as at least one other local server ("europe-1" or "europe-2") responds with the same result. The rest of the replication to the other data centers will be executed asynchronously.

Consistency

Since multiple data centers can have a local quorum, it is possible to have **Eventual Consistency** between them. It's always suggested to keep the number of servers odd, so you can, eventually, always be consistent. Example of 2 data centers with an equal number of servers:

```
  "dataCenters": {
    "rome": {
      "writeQuorum": "all",
      "servers": [ "europe-0", "europe-1", "europe-2" ]
    },
    "austin": {
      "writeQuorum": "all",
      "servers": [ "usa-0", "usa-1", "usa-2" ]
    }
```

In this case if an UPDATE operation is executed by the server "usa-0" (the coordinator), it will reach the quorum only if `all` the servers in the "austin" data center provide the same result. Let's say the result for all these 2 servers was `5` (in the UPDATE operation the result is the updated version of the record). But what happens if all the other 3 servers in "rome" return the version `6` ? You have no majority in this case (3 against 3), so the coordinator cannot establish who is the winner. The database become inconsistent.

In order to automatically manage conflicts, the suggested configuration is always to keep an **odd number** of servers whether you use multiple data centers or not.

Conflict Resolution Policy

In OrientDB Enterprise Edition the additional `dc` Conflict Resolution Strategy is supported to let to a configured data center to always win in case of conflict. To use this strategy in the conflict resolution chain, append `dc` at the chain by overwriting the global setting `distributed.conflictResolverRepairerChain` . Example:

`-Ddistributed.conflictResolverRepairerChain=majority,content,version,dc{winner:asia}`

Note the configuration passed in curly brackets `{winner:asia}` containing the name of the data center that will be the winner in case no winner has been found in the chain.

Client Reconnection

Starting from OrientDB v2.2.25, when a client has lost its connection to the server, it's able to transparently reconnect to another server in the same Data Center of the server where it was connected. If no other server is available for the same Data Center, then it will try to reconnect to a server in another Data Center and, one by one, will try until one server is reachable. If no servers are reachable, then a connection error is thrown. You can change this strategy by forcing the reconnection to be only against the same Data Center. In this case, if no servers are available in the same Data Center, a connection error is thrown. This is the setting:

```
-Dnetwork.retry.strategy=same-dc
```

HA SQL Commands

Some HA-related SQL commands are available.

Internals

This section contains internal technical information. Users usually are not interested to such technical details, but if you want to hack OrientDB or become a contributor this information could be useful.

Storages

Any OrientDB database relies on a Storage. OrientDB supports 4 storage types:

- **plocal**, persistent disk-based, where the access is made in the same JVM process
- **remote**, by using the network to access a remote storage
- **memory**, all data remains in memory
- **local**, deprecated, it's the first version of disk based storage, but has been replaced by **plocal**

A Storage is composed of multiple Clusters.

Memory Storage

Memory Storage is local and endures **only as long as the JVM is running**. For the application programmer, it is identical in use to Paginated Local Storage. Transactions remain atomic, consistent and isolated (but are not durable).

Memory is the fastest choice for tasks where data are read from an external source, and where the results are provided without needing to persist the graph itself. It is also a good choice for certain test scenarios, since an abrupt exit leaves no debris on the disk.

A database using memory storage is designated by a URL of the form `memory:<path>`, for example `memory:test`. A hierarchical path is allowed, for example `memory:subdir/test`.

The memory available for the database is *direct memory* allocated by the JVM as needed. In the case of a `plocal:` database, memory like this provides a cache whose pages are loaded from or flushed to pages files on disk. In the case of a `memory:` database, these direct memory pages are the ultimate storage. The database is allowed to take up more memory than there is physical RAM, as the JVM will allocate more from swap. Total size is limited by the `-XX:MaxDirectMemorySize` JVM option.

PLocal Storage

The Paginated Local Storage, "**plocal**" from now, is a disk based storage which works with data using page model.

plocal storage consists of several components each of those components use disk data through **disk cache**.

Below is list of plocal storage components and short description of each of them:

1. **Clusters** are managed by 2 kinds of files:
    - **.pcl** files contain the cluster data
    - **.cpm** files contain the mapping between record's cluster position and real physical position
2. **Write Ahead (operation) Log (WAL)** are managed by 2 kinds of files:
    - **.wal** to store the log content
    - **.wmr** contains timing about synchronization operations between storage cache and disk system
3. **SBTree Index**, it uses files with extensions **.sbt**.
4. **Hash Index**, it uses files with extensions **.hit**, **.him** and **.hib**.
5. **Index Containers** to store values of single entries of not unique index (Index RID Set). It uses files with extension **.irs**.
6. **File mapping**, maps between file names and file ids (used internally). It's a single file with name: **name_id_map.cm**.

File System

Since PLOCAL is disk-based, all pages are flushed to physical files. You can specify any mounted partitions on your machine, backed by Disks, SSD, Flash Disks or DRAM.

Cluster

Cluster is logical piece of disk space where storage stores records data. Each cluster is split in pages. **Page** is a single atomic unit, which is used by cluster.

Each page contains system information and records data. System information includes "magic number" and a crc32 check sum of the page content. This information is used to check storage integrity after a DB crash. To start an integrity check run command "check database" from console.

Each cluster has 2 sub components:

- data file, with extension .pcl
- mapping between physical position of record in the data file and cluster position, with extension .cpm

File System

To speed up access to the most requested clusters it's recommended to use the cluster files to a SSD or any faster support than disk. To do that, move the files to the mounted partitions and create symbolic links to them on original path. OrientDB will follow symbolic links and will open cluster files everywhere are reachable.

Cluster pointers

The mapping between data file and physical position is managed with a list. Each entry in this list is a fixed size element, which is the pointer to the physical position of the record in the data file.

Because the data file is paginated, this pointer will consist of 2 items: a page index (long value) and the position of the record inside the page (int value). Each record pointer consumes 12 bytes.

Creation of new records in cluster

When a new record is inserted, a pointer is added to the list. The index of this pointer is the cluster position. The list is an append only data structure, so if you add a new record its cluster position will be unique and will not be reused.

Deletion of records in cluster

When you delete a record, the page index and record position are set to -1. So the record pointer is transformed into a "tombstone". You can think of a record id like a **uuid**. It is unique and never reused.

Usually when you delete records you lose very small amount of disk space. This can be mitigated with a periodic "offline compaction", by performing a database export/import. During this process, cluster positions will be changed (tombstones will be ignored during export) and the lost space will be recovered. So during the import process, the cluster positions can change.

Migration of RID

The OrientDB import tool uses a manual hash index (by default the name is '___exportImportRIDMap') to map the old record ids to new record ids.

Write Ahead (operation) Log (WAL)

The Write Ahead Log (or WAL) is used to restore storage data after a non-soft shutdown:

- Hard kill of the OrientDB process
- Crash/Failure of the Java Virtual Machine that runs OrientDB
- Crash/Failure of the Operating System that is hosting OrientDB

All the operations on **plocal** components are logged in WAL before they are performed. WAL is an append only data structure. You can think of it as a list of records which contain information about operations performed on storage components.

WAL flush

WAL content is flushed to the disk on these events:

- every 1 second in background thread (flush interval can be changed in **storage.wal.commitTimeout** configuration property)
- synchronously if the amount of RAM used by WAL exceeds 65Mb (can be changed in **storage.wal.cacheSize** configuration property).

As result if OrientDB crashes, all data changes done during <=1 second interval before crash will be lost. This is a trade off between performance and durability.

Put the WAL on a separate disk

It's strongly recommended that WAL records are stored on a different disk than the disk used to store the DB content. In this way data I/O operations will not be interrupted by WAL I/O operations. This can be done by setting the **storage.wal.path** property to the folder where storage WAL files will be placed.

How Indexes use WAL?

Indexes can work with WAL in 2 modes:

- **ROLLBACK_ONLY** (default mode) and
- **FULL**

In ROLLBACK_ONLY mode only the data needed to rollback transactions is stored. This means that WAL records can not be used to restore index content after a crash. In the case of a crash, the indexes will be rebuilt automatically.

In FULL mode, indexes can be restored after DB crash without a rebuild. You can change index durability mode by setting the property **index.txMode**.

You can find more details about WAL **here**.

File types

PLocal stores data on the file system using different files, using the following extensions:

- **.cpm**, contains the mapping between real physical positions and cluster positions. If you delete a record, the tombstone is placed here. Each tombstone consumes about 12 bytes.
- **.pcl**, data file
- **.sbt**, is index file
- **.wal** and **.wmr**, are Journal Write Ahead (operation) Log files
- **.cm**, is the mapping between file id and real file name (used internally)
- **.irs**, is the RID set file for non-unique indexes

How it works (Internal)

Paginated storage is a 2-level disk cache that works together with the write ahead log.

Every file is spit into pages, and each file operation is atomic at a page level. The 2-level disk cache allows:

1. Cache frequently accessed pages in memory.
2. Automatically separate pages which are rarely accessed from frequently accessed and rid off the first from cache memory.
3. Minimize amount of disk head seeks during data writes.
4. In case of low or middle write data load allows to mitigate pauses are needed to write data to the disk by flushing all changed or newly added pages to the disk in background thread.
5. Works together with WAL to make any set changes on single page look like atomic operation.

2-level cache itself consist of a **Read Cache** (implementation is based on 2Q cache algorithm) and a **Write cache* (implementation is based on WOW cache algorithm).

Typical set of operations are needed to work with any file looks like following:

1. Open file using OReadWriteDiskCache#openFile operation and get id of open file. If the file does not exist it will be automatically created. The id of file is stored in a special meta data file and will always belong to the given file till it will be deleted.
2. Allocate new page OReadWriteDiskCache#allocateNewPage or load existing one ORreadWriteDiskCache#load into off-heap memory.
3. Retrieve pointer to the allocated area of off-heap memory OCacheEntry#getCachePointer().
4. If you plan to change page data acquire a write lock, or a read lock if you read data and your single file page is shared across several data structures. Write lock must be acquired whether a single page is used between several data structures or not. The write lock is needed to prevent flushing inconsistent pages to the disk by the “data flush” thread of the write cache. OCachePointer#acquireExclusiveLock.
5. Update/read data in off heap memory.
6. Release write lock if needed. OCachePointer#releaseExclusiveLock.
7. Mark page as dirty if you changed page data. It will allow write cache to flush pages which are really changed OCacheEntry#markDirty.
8. Push record back to the disk cache: indicate to the cache that you will not use this page any more so it can be safely evicted from the memory to make room to other pages OReadWriteDiskCache#release.

So what is going on underneath when we load and release pages?

When we load page the Read Cache looks it in one of its two LRU lists. One list is for data that was accessed several times and then not accessed for very long period of time. It consumes 25% of memory. The second is for data that is accessed frequently for a long period of time. It consumes 75% of memory.

If the page is not in either LRU queue, the Read Cache asks the Write Cache to load page from the disk.

If we are lucky and the page is queued to flush but is still in the Write Queue of Write Cache it will be retrieved from there. Otherwise, the Write Cache will load the page from disk.

When data will be read from file by Write Cache, it will be put in LRU queue which contains “short living” pages. Eventually, if this pages will be accessed frequently during long time interval, loaded page will be moved to the LRU of “long living” pages.

When we release a page and the page is marked as dirty, it is put into the Write Cache which adds it to the Write Queue. The Write Queue can be considered as ring buffer where all the pages are sorted by their position on the disk. This trick allows to minimize disk head movements during pages flush. What is more interesting is that pages are always flushed in the background in the “background flush” thread. This approach allows to mitigate I/O bottleneck if we have enough RAM to work in memory only and flush data in background.

So it was about how disk cache works. But how we achieve durability of changes on page level and what is more interesting on the level when we work with complex data structures like Trees or Hash Maps (these data structures are used in indexes).

If we look back on set of operations which we perform to manipulate file data you see that step 5 does not contains any references to OrientDB API. That is because there are two ways to work with off heap pages: durable and not durable.

The simple (not durable way) is to work with methods of direct memory pointer com.orientechnologies.common.directmemory.ODirectMemoryPointer(setLong/getLong, setInt/getInt and so on). If you would like to make all changes in your data structures durable you should not work with direct memory pointer but should create a component that will present part of your data structure by extending com.orientechnologies.orient.core.storage.impl.local.paginated.ODurablePage class. This class has similar methods for manipulation of data in off heap pages, but it also tracks all changes made to the page. It can return the diff between the old/new states of page using the com.orientechnologies.orient.core.storage.impl.local.paginated.ODurablePage#getPageChanges method. Also this class allows to apply given diff to the old/new snapshot of given pages to repeat/revert (restoreChanges()/revertChanges()) changes are done for this page.

PLocal Engine

Paginated Local storage engine, also called as **"plocal"**, is intended to be used as durable replacement of the previous local storage.

plocal storage is based on principle that using disk cache which contains disk data that are split by fixed size portions (pages) and write ahead logging approach (when changes in page are logged first in so called durable storage) we can achieve following characteristics:

1. Operations on single page are atomic.
2. Changes applied to the page can be restored after server crash even if they were not flushed to the disk.

Using write ahead log and page based cache we can achieve durability/performance trade off. We do not need to flush every page to the disk so we will avoid costly random I/O operations as much as possible and still can achieve durability using much cheaper append only I/O operations.

From all given above we can conclude one more advantage of plocal against local - it has much faster transactions implementation. In order achieve durability on local storage we should set tx.commit.synch property to true (perform synchronization of disk cache on each transaction commit) which of course makes create/update/delete operations inside transaction pretty slow.

Lets go deeper in implementation of both storages.

Local storage uses MMAP implementation and it means that caching of read and write operations can not be controlled, plocal from other side uses two types of caches read cache and write cache (the last is under implementation yet and not included in current implementation).

The decision to split responsibilities between 2 caches is based on the fact that characters of distribution of "read" and "write" data are different and they should be processed separately.

We replaced MMAP by our own cache solution because we needed low level integration with cache life cycle to provide fast and durable integration between WAL and disk cache. Also we expect that when cache implementation will be finished issues like https://github.com/orientechnologies/orientdb/issues/1202 and https://github.com/orientechnologies/orientdb/issues/1339 will be fixed automatically.

Despite of the fact that write cache is still not finished it does not mean that plocal storage is not fully functional. You can use plocal storage and can notice that after server crash it will restore itself.

But it has some limitations right now, mostly related to WAL implementation. When storage is crashed it finds last data check point and restores data from this checkpoint by reading operations log from WAL.

There are two kind of check points full check point and fuzzy check point. The full check point is simple disk cache flush it is performed when cluster is added to storage or cluster attributes are changed, also this check point is performed during storage close.

Fuzzy checkpoint is completely different (it is under implementation yet). During this checkpoint we do not flush disk cache we just store the position of last operation in write ahead log which is for sure flushed to the disk. When we restore data after crash we find this position in WAL and restore all operations from it. Fuzzy check points are much faster and will be performed each hour.

To achieve this trick we should have special write cache which will guarantee that we will not restore data from the begging of database creation during restore from fuzzy checkpoint and will not have performance degradation during write operations. This cache is under implementation.

So right now when we restore data we need to restore data since last DB open operation. It is quite long procedure and require quite space for WAL.

When fuzzy check points will be implemented we will cut unneeded part of WAL during fuzzy check point which will allow us to keep WAL quite small.

We plan to finish fuzzy checkpoints during a month.

But whether we use fuzzy checkpoints or not we can not append to the WAL forever. WAL is split by segments, when WAL size is exceed maximum allowed size the oldest WAL segment will be deleted and new empty one will be created.

The segments size are controlled by storage.wal.maxSegmentSize parameter in megabytes. The maximum WAL size is set by property storage.wal.maxSize parameter in megabytes.

Maximum amount of size which is consumed by disk cache currently is set using two parameters: storage.diskCache.bufferSize - Maximum amount of memory consumed by disk cache in megabytes. storage.diskCache.writeQueueLength - Currently pages are nor flushed on the disk at the same time when disk cache size exceeds, they placed to write queue and when write queue will be full it is flushed. This approach minimize disk head movements but it is temporary solution and will be removed at final version of plocal storage. This parameter is measured in megabytes.

During update the previous record deleted and content of new record is placed instead of old record at the same place. If content of new record does not fit in place occupied by old record, record is split on two parts first is written on old record's place and the second is placed on new or existing page. Placing of part of the record on new page requires to log in WAL not only new but previous data are hold in both pages which requires much more space. To prevent such situation cluster in plocal storage has following attributes:

1. RECORD_GROW_FACTOR the factor which shows how many space will be consumed by record during initial creation. If record size is 100 bytes and RECORD_GROW_FACTOR is 2 record will consume 200 bytes. Additional 100 bytes will be reused when record will grow.
2. RECORD_OVERFLOW_GROW_FACTOR the factor shows how many additional space will be added to the record when record size will exceed initial record size. If record consumed 200 bytes and additional 20 bytes will be needed and RECORD_OVERFLOW_GROW_FACTOR is 1.5 then record will consume 300 bytes after update. Additional 80 bytes will be used during next record updates.

Default value for both parameters are 1.2.

1. USE_WAL if you prefer that some clusters will be faster but not durable you can set this parameter to false.

PLocal Disk-Cache

OrientDB Disk cache consists of two separate cache components that work together:

- **Read Cache**, based on 2Q cache algorithm
- **Write Cache**, based on WOW cache algorithm

Starting from v2.1, OrientDB exposes internal metrics through JMX Beans. Use this information to track and profile OrientDB.

Read Cache

It contains the following queues:

- **a1**, as FIFO queue for pages which were not in the read cache and accessed for the first time
- **am**, as FIFO queue for the hot pages (pages which are accessed frequently during db lifetime). The most used pages stored in **a1** becomes "hot pages" and are moved into the **am** queue.

a1 Queue

a1 queue is split in two queues:

- **a1in** that contains pointers to the pages are cached in memory
- **a1out** that contains pointers to the pages which were in **a1in**, but was not accessed for some time and were removed from RAM. **a1out** contains pointers to the pages located on the disk, not in RAM.

Loading a page

When a page is read for the first time, it's loaded from the disk and put in the **a1in** queue. If there isn't enough space in RAM, the page is moved to **a1out** queue.

If the same page is accessed again, then:

1. if it is in **a1in** queue, nothing
2. if it is in **a1out** queue, the page is supposed to be a "hot page" (that is page which is accessed several times, but doesn't follow the pattern when the page is accessed several times for short interval, and then not accessed at all) we put it in **am** queue
3. if it is in **am** queue, we put the page at the top of am queue

Queue sizes

By default this is the configuration of queues:

- **a1in** queue is 25% of Read Cache size
- **a1out** queue is 50% of Read Cache size
- **am** is 75% of Read Cache size.

When OrientDB starts, both caches are empty, so all the accessed pages are put in **a1in** queue, and the size of this queue is 100% of the size of the Read Cache.

But then, when there is no more room for new pages in **a1in**, the old pages are moved from **a1in** to **a1out**. Eventually when **a1out** contains requested pages we need room for **am** queue pages, so once again we move pages from **a1in** queue to **a1out** queue, **a1in** queue is truncated till it is reached 25% size of read cache.

To make more clear how RAM and pages are distributed through queues lets look at example. Lets suppose we have cache which should cache in RAM 4 pages, and we have 8 pages stored on disk (which have indexes from 0 till 7 accordingly).

When we start database server all queues contain 0 pages:

- am - []
- a1in - []
- a1out - []

Then we read first 4 pages from the disk. So we have:

- am - []
- a1in - [3, 2, 1, 0]
- a1out - []

Then we read 5-th page from the disk and then 6-th , because only 4 pages can be fit into RAM we remove the last pages with indexes 0 and 1, free memory which is consumed by those pages and put them in a1out. So we have:

- am - []
- a1in - [5, 4, 3, 2]
- a1out - [1, 0]

lets read pages with indexes from 6 till 7 (last 2 pages) but a1out can contain only 2 pages (50% of cache size) so the first pages will be removed from o1out. We have here:

- am - []
- a1in - [7, 6, 5, 4]
- a1out - [3, 2]

Then if we will read pages 2, 3 then we mark them (obviously) as hot pages and we put them in am queue but we do not have enough memory for these pages, so we remove pages 5 and 4 from a1in queue and free memory which they consumed. Here we have:

- am - [3, 2]
- a1in - [7, 6]
- a1out - [5, 4]

Then we read page 4 because we read it several times during long time interval it is hot page and we put it in am queue. So we have:

- am - [4, 3, 5]
- a1in - [7]
- a1out - [6, 5]

We reached state when queues can not grow any more so we reached stable, from point of view of memory distribution, state.

This is the used algorithm in pseudo code:

```
On accessing a page X
begin:
 if X is in Am then
   move X to the head of Am
else if (X is in A1out) then
 removeColdestPageIfNeeded
 add X to the head of Am
else if (X is in A1in)
 // do nothing
else
 removeColdestPageIfNeeded
 add X to the head of A1in
end if
end

removeColdestPageIfNeeded
begin
 if there is enough RAM do nothing
 else if( A1in.size > A1inMaxSize)
  free page out the tail of A1in, call it Y
  add identifier of Y to the head of A1out
 if(A1out.size > A1OutMaxSize)
  remove page from the tail of Alout
 end if
 else
  remove page out the tail of Am
  // do not put it on A1out; it hasn’t been
  // accessed for a while
 end if
end
```

Write cache

The main target of the write cache is to eliminate disk I/O overhead, by using the following approaches:

1. All the pages are grouped by 4 adjacent pages (group 0 contains pages from 0 to 3, group 1 contains pages from 4 to 7, etc.). Groups are sorted by position on the disk. Groups are flushed in sorted order, in such way we reduce the random I/O disk head seek overhead. Group's container is implemented as SortedMap: when we reach the end of the map we start again from the beginning. You can think about this data structure as a "ring buffer"
2. All the groups have "recency bit", this bit is set when group is changed. It is needed to avoid to flush pages that are updated too often, it will be wasting of I/O time
3. Groups are continuously flushed by background thread, so until there is enough free memory, all data operations do not suffer of I/O overhead because all operations are performed in memory

Below the pseudo code for write cache algorithms:

Add changed page in cache:

```
begin
 try to find page in page group.
 if such page exist
  replace page in page group
  set group's "recency bit" to true
 end if
 else
  add page group
  set group's "recency bit" to true
 end if
end
```

On periodical background flush

```
begin
 calculate amount of groups to flush
 start from group next to flushed in previous flush iteration
 set "force sync" flag to false

 for each group
  if "recency bit" set to true and "force sync" set to false
   set "recency bit" to false
  else
   flush pages in group
   remove group from ring buffer
  end if
 end for

  if we need to flush more than one group and not all of them are flushed repeat "flush
loop" with "force sync" flag set to true.
end
```

The collection of groups to flush is calculated in following way:

1. if amount of RAM consumed by pages is less than 80%, then 1 group is flushed.
2. if amount of RAM consumed by pages is more than 80%, then 20% of groups is flushed.
3. if amount of RAM consumed by pages is more than 90%, then 40% of groups is flushed.

Interaction between Read and Write Caches

By default the maximum size of Read Cache is 70% of cache RAM and 30% for Write Cache.

When a page is requested, the Read Cache looks into the cached pages. If it's not present, the Read Cache requests page from the Write Cache. Write Cache looks for the page inside the Ring Buffer: if it is absent, it reads the page from the disk and returns it directly to the Read Cache without caching it inside of Write Cache Ring Buffer.

Implementation details

Page which is used by storage data structure (such as cluster or index) can not be evicted (removed from memory) so each page pointer also has "usage counter" when page is requested by cache user, "usage counter" is incremented and decremented when page is released. So removeColdestPageIfNeeded() method does not remove tail page, but removes page closest to tail which usage counter is 0, if such pages do not exit either exception is thrown or cache size is automatically increased and warning message is added to server log (default) (it is controlled by properties **server.cache.2q.increaseOnDemand** and **server.cache.2q.increaseStep**, the last one is amount of percent of RAM from original size on which cache size will be increased).

When a page is changed, the cache page pointer (data structure which is called OCacheEntry) is marked as dirty by cache user before release. If cache page is dirty it is put in write cache by read cache during call of OReadWriteDiskCache#release() method. Strictly speaking memory content of page is not copied, it will be too slow, but pointer to the page is passed. This pointer (OCachePointer) tracks amount of referents if no one references this pointer, it frees referenced page.

Obviously caches work in multithreaded environment, so to prevent data inconsistencies each page is not accessed directly. Read cache returns data structure which is called cache pointer. This pointer contains pointer to the page and lock object. Cache user should acquire read or write lock before it will use this page. The same read lock is acquired by write cache for each page in group before flush, so inconsistent data will not be flushed to the disk. There is interesting nuance here, write cache tries to acquire read lock and if it is used by cache user it will not wait but will try to flush other group.

PLocal WAL (Journal)

Write Ahead Log, **WAL** form now, is operation log which is used to store data about operations which were performed on disk cache page. WAL is enabled by default.

You could disable the journal (WAL) for some operations where reliability is not necessary:

```
-Dstorage.useWAL=false
```

By default, the WAL files are written in the database folder. Since these files can growth very fast, it's a best practice to store in a dedicated partition. WAL are written in append-only mode, so there is not much difference on using a SSD or a normal HDD. If you have a SSD we suggest to use for database files only, not WAL.

To setup a different location than database folder, set the `WAL_LOCATION` variable.

```
OGlobalConfiguration.WAL_LOCATION.setValue("/temp/wal")
```

or at JVM level:

```
java ... -Dstorage.wal.path=/temp/wal ...
```

This log is not an high level log, which is used to log operations on record level. During each page change following values are stored:

1. offset and length of chunk of bytes which was changed.
2. previous value of chunk of bytes.
3. replaced (new) value of chunk of bytes.

As you can see WAL contains not logical but raw (in form of chunk of bytes) presentation of data which was/is contained inside of page. Such format of record of write ahead log allows to apply the same changes to the page several times and as result allows do not flush cache content after each TX operation but do such flush on demand and flush only chosen pages instead of whole cache. The second advantage is following if storage is crashed during data restore operation it can be restored again , again and again.

Lets say we have page where following changes are done.

1. 10 bytes at the beginning were changed.
2. 10 bytes at the end were changed.

Storage is crashed during the middle of page flush, which does not mean that first 10 bytes are written, so lets suppose that the last 10 changed byte were written, but first 10 bytes were not.

During data restore we apply all operations stored in WAL one by one, which means that we set first 10 bytes of changed page and then last 10 bytes of this page. So the changed page will have correct state does not matter whether it's state was flushed to the disk or not.

WAL file is split on pages and segments, each page contains in header CRC32 code of page content and "magic number". When operation records are logged to WAL they are serialized and binary content appended to the current page, if it is not enough space left in page to accommodate binary presentation of whole record, the part of binary content (which does not fit inside of current page) will be put inside of next record. It is important to avoid gaps (free space) inside of pages. As any other files WAL can be corrupted because of power failure and detection of gaps inside WAL pages is one of the approaches how database separates broken and "healthy" WAL pages. More about this later.

Any operation may include not single but several pages, to avoid data inconsistency all operations on several records inside of one logical operation are considered as single atomic operation. To achieve this functionality following types of WAL records were introduced:

1. atomic operation start.
2. atomic operation end.
3. record which contains changes are done in single page inside of atomic operation.

These records contain following fields:

1. Atomic operation start record contains following fields:
    i. Atomic operation id (uuid).
    ii. LSN (log sequence number) - physical position of log record inside WAL.
2. Atomic operation end record contains following fields:
    i. Atomic operation id (uuid).
    ii. LSN (log sequence number) - physical position of log record inside WAL.
    iii. rollback flag - indicates whether given atomic operation should be rolled back.
3. Record which contains page changes contains following fields:
    i. LSN (log sequence number) - physical position of log record inside WAL.
    ii. page index and file id of changed page.
    iii. Page changes itself.
    iv. LSN of change which was applied to the current page before given one - prevLSN.

The last record's type (page changes container) contains field (d. item) which deserves additional explanation. Each cache page contains following "system" fields:

1. CRC32 code of the rest of content.
2. magic number
3. LSN of last change applied to the page - page LSN.

Every time we perform changes on the page before we release it back to the cache we log page changes to the WAL, assign LSN of WAL record as the "page LSN" and only after that release page back to the cache.

When WAL flushes it's pages it does not do it at once when current page is filled it is put in cache and is flushed in background along with other cached pages. Flush is performed every second in background thread (it is trade off between performance and durability). But there are two exceptions when flush is performed in thread which put record in WAL:

1. If WAL page's cache is exhausted.
2. If cache page is flushed, page LSN is compared with LSN of last flushed WAL record and if page LSN is more than LSN of flushed WAL record then flush of WAL pages is triggered. LSN is physical position of WAL record, because of WAL is append only log so if "page LSN" is more than LSN of flushed record it means that changes for given page were logged but not flushed, but we can restore state of page only and only if all page changes will be contained in WAL too.

Given all of this data restore process looks like following:

```
begin
go trough all WAL records one by one
gather together all atomic operation records in one batch
when "atomic operation end" record was found
  if commit should be performed
    go through all atomic operation records from first to last, apply all page changes,
set page LSN to the LSN of applied WAL record.
  else
    go through all atomic operation records from last to first, set old page's content,
set page LSN to the WALRecord.prevLSN value.
  endif
end
```

As it is written before WAL files are usual files and they can be flushed only partially if power is switched off during WAL cache flush. There are two cases how WAL pages can be broken:

1. Pages are flushed partially.
2. Some of pages are completely flushed, some are not flushed.

First case is very easy to detect and resolve:

1. When we open WAL during DB start we verify that size of WAL multiplies of WAL page size if it is not WAL size is truncated to page size.
2. When we read pages one by one we verify CR32 and magic number of each page. If page is broken we stop data restore procedure here.

Second case a bit more tricky. Because WAL is append only log, there is two possible sub-cases, lets suppose we have 3 pages after 2-nd (broken) flush. First and first half of second page were flushed during first flush and second half of second page and third page were flushed during second flush. Because second flush was interrupted by power failure we can have two possible states:

1. Second half of page was flushed but third was not. It is easy to detect by checking CRC and magic number values.
2. Second half of page is not flushed but third page is flushed. In such case CRC and magic number values will be correct and we can not use them instead of this when we read WAL page we check if this page has free space if it has then we check if this is last page if it is not we mark this WAL page as broken.

Local Storage

> **NOTE**: Local storage is no longer available as of Version 2.0.

Local storage is the first version of disk-based storage engine, but has been replaced by plocal. Don't create new databases using **local**, but rather plocal. Local storage has been kept only for compatibility purpose.

A **local** storage is composed of multiple Clusters and Data Segments.

Local Physical Cluster

The cluster is mapped 1-by-2 to files in the underlying File System. The local physical cluster uses two or more files: One or more files with extension "ocl" (OrientDB Cluster) and only one file with the extension "och" (OrientDB Cluster Holes).

For example, if you create the "Person" cluster, the following files will be created in the folder that contains your database:

- person.0.ocl
- person.och

The first file contains the pointers to the record content in ODA (OrientDB Data Segment). The '0' in the name indicates that more successive data files can be created for this cluster. You can split a physical cluster into multiple real files. This behavior depends on your configuration. When a cluster file is full, a new file will be used.

The second file is the "Hole" file that stores the holes in the cluster caused by deleted data.

NOTE (again, but very important): You can move real files in your file system only by using the OrientDB APIs.

Data Segment

OrientDB uses **data segments** to store the record content. The data segment behaves similar to the physical cluster files: it uses two or more files. One or multiple files with the extension "oda" (OrientDB Data) and only one file with the extension "odh" (OrientDB Data Holes).

By default OrientDB creates the first data segment named "default". In the folder that contains your database you will find the following files:

- default.0.oda
- default.odh

The first file is the one that contains the real data. The '0' in the name indicates that more successive data files can be created for this cluster. You can split a data segment into multiple real files. This behavior depends on your configuration. When a data segment file is full, a new file will be used.

NOTE (again, but it can't be said too many times): You can move real files in your file system only by using the OrientDB APIs.

Interaction between components: load record use case:

Clusters

OrientDB uses **clusters** to store links to the data. A cluster is a generic way to group records. It is a concept that does not exist in the Relational world, so it is something that readers from the relational world should pay particular attention to.

You can use a cluster to group all the records of a certain type, or by a specific value. Here are some examples of how clusters may be used:

- Use the cluster "Person" to group all the records of type "Person". This may at first look very similar to the RDBMS tables, but be aware that the concept is quite different.
- Use the cluster "Cache" to group all the records most accessed.
- Use the cluster "Today" to group all the records created today.
- Use the cluster "CityCar" to group all the city cars.

If you have a background from the RDBMS world, you may benefit to think of a cluster as a table (at least in the beginning). OrientDB uses a cluster per "class" by default, so the similarities may be striking at first. However, as you get more advanced, we strongly recommend that you spend some time understanding clustering and how it differs from RDBMS tables.

A cluster can be local (physical) or in-memory.

Note: If you used an earlier version of OrientDB. The concept of "Logical Clusters" are not supported after the introduction of version 1.0.

Persistent Cluster

Also called Physical cluster, it stores data on disk.

In-Memory Cluster

The information stored in "In-Memory clusters" is volatile (that is, it is never stored to disk). Use this cluster only to work with temporary data. If you need an In-Memory database, create it as an In-memory Database. In-memory databases have only In-memory clusters.

Limits

Below are the limitations of the OrientDB engine:

- **Databases**: There is no limit to the number of databases per server or embedded. Users reported no problem with 1000 databases open
- **Clusters**: each database can have a maximum of 32,767 clusters (2^15-1)
- **Records** per cluster (**Documents**, **Vertices** and **Edges** are stored as records): can be up to 9,223,372,036,854,780,000 (2^63-1), namely 9,223,372 Trillion records
- **Records** per database (**Documents**, **Vertices** and **Edges** are stored as records): can be up to 302,231,454,903,000,000,000,000 (2^78-1), namely 302,231,454,903 Trillion records
- **Record size**: up to 2GB each, even if we suggest avoiding the creation of records larger than 10MB. They can be split into smaller records, take a look at Binary Data
- **Document Properties** can be:
  - up to 2 Billion per database for schema-full properties
  - there is no limitation regarding the number of properties in schema-less mode. The only concrete limit is the size of the Document where they can be stored. Users have reported no problems working with documents made of 15,000 properties
- **Indexes** can be up to 2 Billion per database. There are no limitations regarding the number of indexes per class
- **Queries** can return a maximum of 2 Billion rows, no matter the number of the properties per record
- **Concurrency level**: in order to guarantee atomicity and consistency, OrientDB acquire an exclusive lock on the storage during transaction commit. This means transactions are serialized. Giving this limitation, *the OrientDB team is already working on improving parallelism to achieve better scalability on multi-core machines by optimizing internal structure to avoid exclusive locking.*

Look also at the limitations with distributed setup.

RidBag

RidBag is a data structure that manages multiple RIDs. It is a collection without an order that could contain duplication. Actually the bag (or multi-set) is similar to set, but could hold several instances of the same object.

RidBag is designed to efficiently manage edges in graph database, however it could be used directly in document level.

Why it doesn't implement java java.util.Collection

The first goal of RidBag is to be able efficiently manage billions of entries. In the same time it should be possible to use such collection in the remote. The main restriction of such case is amount of data that should be sent over the network.

Some of the methods of `java.util.Collection` is really hard to efficiently implement for such case, when most of them are not required for relationship management.

How it works

RidBag has 2 modes:

- **Embedded** - has list-like representation and serialize its content right in document
- **Tree-based** - uses external tree-based data structure to manages its content. Has some overhead over embedded one, but much more efficient for many records.

By default newly created RidBags are embedded and they are automatically converted to tree-based after reaching a threshold. The automatic conversion in opposite direction is disabled by default due to an issues in remote mode. However you can use it if you are using OrientDB embedded and don't use remote connections.

The conversion is **always** done on server and never on client. Firstly it allows to avoid a lot of issues related to simultaneous conversions. Secondly it allows to simplify the clients.

Configuration

RidBag could be configured with OGlobalConfiguration.

- `RID_BAG_EMBEDDED_TO_SBTREEBONSAI_THRESHOLD` (`ridBag.embeddedToSbtreeBonsaiThreshold`) - The threshold of LINKBAG conversion to sbtree-based implementation. *Default value: 40.*
- `RID_BAG_SBTREEBONSAI_TO_EMBEDDED_THRESHOLD` (`ridBag.sbtreeBonsaiToEmbeddedToThreshold`) - The threshold of LINKBAG conversion to embedded implementation. *Disabled by default.*

Setting `RID_BAG_EMBEDDED_TO_SBTREEBONSAI_THRESHOLD` to `-1` forces using of sbtree-based RidBag. Look at Concurrency on adding edges to know more about impact on graphs of this setting.

❗	*NOTE: While running in distributed mode SBTrees are not supported. If using a distributed database then you must set* `ridBag.embeddedToSbtreeBonsaiThreshold=Integer.MAX\_VALUE` *to avoid replication errors.*

Interaction with remote clients

> NOTE: This topic is rather for contributors or driver developers. OrientDB users don't have to care about bag internals.

As been said rid bag could be represented in two ways: *embedded* and *tree-based*. The first implementation serializes its entries right into stream of its owner. The second one serializes only a special pointer to an external data structure.

In the same time the server could automatically convert the bag from embedded to tree-based during save/commit. So client should be aware of such conversion because it can hold an instance of rid bag.

To "listen" for such changes client should assign a temporary collection id to bag.

The flow of save/commit commands:

```
Client                                                         Server
  |                                                              |
  V                                                              |
 /---------\      Record content [that contain bag with uuid]        |
|           |------------------------------------------------------->|
|   Send    |                                                        | Convert to tree
|  command  |                                                        | and save to disk
| to server |   Response with changes (A new collection pointer)     |
|           |<-------------------------------------------------------/
 \---------/        the target of new identity assignment
  |                  identified by temporary id
  |
  V
/-----------------------------\
| Update a collection pointer |
| to be able perform actions  |
| with remote tree            |
\-----------------------------/
```

Serialization.

> NOTE: This topic is rather for contributors or driver developers. OrietnDB users don't have to care about bag serialization

Save and load operations are performed during save/load of owner of RidBag. Other operations are performed separately and have its own commands in binary protocol.

To get definitive syntax of each network command see Network Binary Protocol

Serialization during save and load

The bag is serialized in a binary format. If it is serialized into document by CSV serializer it's encoded with base64.

The format is following:

```
(config:byte)[(temp_id:uuid:optional)](content.md)
```

The first byte is reserved for configuration. The bits of config byte define the further structure of binary stream:

1. 1st: 1 if bag is embedded. 0 if tree-based.
2. 2nd: 1 if uuid is assigned, 0 otherwise. Used to prevent storing of UUID to disk.

If bag is embedded content has following

```
(size:int)(link:rid)*
```

If bag is tree based it doesn't serialize the content it serialize just a **collection pointer** that points where the tree structure is saved:

```
(collectionPointer)(size:int)(changes)
```

See also serialization of collection pointer and rid bag changes

The cached size value is also saved to stream. It don't have to be recalculated in most cases.

The *changes* part is used by client to send changes to server. In all other cases *size* of cahnges is 0

Size of rid bag

Calculation of size for embedded rid bag is straight forward. But what about tree-based bag.

The issue there that we probably have some changes on client that have not been send to the server. On the other hand we probably have billions of records in bag on server. So we can't just calculate size on server because we don't know how to apply changes readjust that size regarding to changes on client. And in the same time calculation of size on client is inefficient because we had to iterate over big amount of records over the network.

That's why following approach is used:

- Client ask server for RidBag size and provide client changes
- Server apply changes in memory to calculate size, but doesn't save them to bag.
- New entries (documents that have never been saved) are not sent to server for recalculation, and the size is adjusted on client. New entries doesn't have an identity yet, but rid bag works only with identities. So to prevent miscalculation it is easier to add the count of not saved entries to calculated bag size on client.

REQUEST_RIDBAG_GET_SIZE network command

Request:

```
(treePointer:collectionPointer)(changes)
```

See also serialization of collection pointer and rid bag changes

Response:

```
(size:int)
```

Iteration over tree-based RidBag

Iteration over tree-based RidBag could be implemented with **REQUEST_SBTREE_BONSAI_GET_ENTRIES_MAJOR** and **REQUEST_SBTREE_BONSAI_FIRST_KEY**.

Server doesn't know anything about client changes. So iterator implementation should apply changes to the result before returning result to the user.

The algorithm of fetching records from server is following:

1. Get the *first key* from SB-tree.
2. Fetch portion of data with *getEtriesMajor* operation.
3. Repeat **step 2** while *getEtriesMajor* returns any result.

Serialization of rid bag changes

```
(changesSize:int)[(link:rid)(changeType:byte)(value:int)]*
```

changes could be 2 types:

- **Diff** - value defines how the number of entries is changed for specific link.
- **Absolute** - sets the number of entries of specified link. The number defined by value field.

Serialization of collection pointer

```
(fileId:long)(pageIndex:long)(pageOffset:int)
```

SQL parser syntax

BNF token specification

```
DOCUMENT START
TOKENS
<DEFAULT> SKIP : {
" "
| "\t"
| "\n"
| "\r"
}

/** reserved words **/<DEFAULT> TOKEN : {
<SELECT: ("s" | "S") ("e" | "E") ("l" | "L") ("e" | "E") ("c" | "C") ("t" | "T")>
| <INSERT: ("i" | "I") ("n" | "N") ("s" | "S") ("e" | "E") ("r" | "R") ("t" | "T")>
| <UPDATE: ("u" | "U") ("p" | "P") ("d" | "D") ("a" | "A") ("t" | "T") ("e" | "E")>
| <DELETE: ("d" | "D") ("e" | "E") ("l" | "L") ("e" | "E") ("t" | "T") ("e" | "E")>
| <FROM: ("f" | "F") ("r" | "R") ("o" | "O") ("m" | "M")>
| <WHERE: ("w" | "W") ("h" | "H") ("e" | "E") ("r" | "R") ("e" | "E")>
| <INTO: ("i" | "I") ("n" | "N") ("t" | "T") ("o" | "O")>
| <VALUES: ("v" | "V") ("a" | "A") ("l" | "L") ("u" | "U") ("e" | "E") ("s" | "S")>
| <SET: ("s" | "S") ("e" | "E") ("t" | "T")>
| <ADD: ("a" | "A") ("d" | "D") ("d" | "D")>
| <REMOVE: ("r" | "R") ("e" | "E") ("m" | "M") ("o" | "O") ("v" | "V") ("e" | "E")>
| <AND: ("a" | "A") ("n" | "N") ("d" | "D")>
| <OR: ("o" | "O") ("r" | "R")>
| <NULL: ("N" | "n") ("U" | "u") ("L" | "l") ("L" | "l")>
| <ORDER: ("o" | "O") ("r" | "R") ("d" | "D") ("e" | "E") ("r" | "R")>
| <BY: ("b" | "B") ("y" | "Y")>
| <LIMIT: ("l" | "L") ("i" | "I") ("m" | "M") ("i" | "I") ("t" | "T")>
| <RANGE: ("r" | "R") ("a" | "A") ("n" | "N") ("g" | "G") ("e" | "E")>
| <ASC: ("a" | "A") ("s" | "S") ("c" | "C")>
| <AS: ("a" | "A") ("s" | "S")>
| <DESC: ("d" | "D") ("e" | "E") ("s" | "S") ("c" | "C")>
| <UNSAFE: ("u" | "U") ("n" | "N") ("s" | "S") ("a" | "A") ("f" | "F") ("e" | "E")>
| <BATCH: ("b" | "B") ("a" | "A") ("t" | "T") ("c" | "C") ("h" | "H") >
| <THIS: "@this">
| <RECORD_ATTRIBUTE: <RID_ATTR> | <CLASS_ATTR> | <VERSION_ATTR> | <SIZE_ATTR> | <TYPE_ATTR>>
| <#RID_ATTR: "@rid">
| <#CLASS_ATTR: "@class">
| <#VERSION_ATTR: "@version">
| <#SIZE_ATTR: "@size">
| <#TYPE_ATTR: "@type">
}

/** LITERALS **/<DEFAULT> TOKEN : {
<INTEGER_LITERAL: <DECIMAL_LITERAL> ([| <HEX_LITERAL> (["l","L"]("l","L"].md)?))? | <OCTAL_LITERAL> ([| <#DECIMAL_LITERAL: ["1
"-"9"]("l","L"].md)?>) ([| <#HEX_LITERAL: "0" ["x","X"]("0"-"9"].md)*>) ([| <#OCTAL_LITERAL: "0" (["0"-"7"]("0"-"9","a"-"f","A
"-"F"].md)+>))**>
| <FLOATING_POINT_LITERAL: <DECIMAL_FLOATING_POINT_LITERAL> | <HEXADECIMAL_FLOATING_POINT_LITERAL>>
| <#DECIMAL_FLOATING_POINT_LITERAL: (["." (["0"-"9"]("0"-"9"].md)+))** (<DECIMAL_EXPONENT>)? ([| "." (["0"-"9"]("f","F","d","D
"].md)?))+ (<DECIMAL_EXPONENT>)? ([| (["0"-"9"]("f","F","d","D"].md)?))+ <DECIMAL_EXPONENT> ([| (["0"-"9"]("f","F","d","D"].md
)?))+ (<DECIMAL_EXPONENT>)? [| <#DECIMAL_EXPONENT: ["e","E"]("f","F","d","D"]>.md) ([(["0"-"9"]("+","-"].md)?))+>
| <#HEXADECIMAL_FLOATING_POINT_LITERAL: "0" [(["0"-"9","a"-"f","A"-"F"]("x","X"].md))+ (".")? <HEXADECIMAL_EXPONENT> ([| "0" [
"x","X"]("f","F","d","D"].md)?) (["." (["0"-"9","a"-"f","A"-"F"]("0"-"9","a"-"f","A"-"F"].md)*))+ <HEXADECIMAL_EXPONENT> ([| <
#HEXADECIMAL_EXPONENT: ["p","P"]("f","F","d","D"].md)?>) ([(["0"-"9"]("+","-"].md)?))+>
| <CHARACTER_LITERAL: "\'" (~[| "\\" (["n","t","b","r","f","\\","\'","\""]("\'","\\","\n","\r"].md) | [(["0"-"7"]("0"-"7"].md)
)? | ["0"-"7"]("0"-"7".md)"0"-"3"]) ["\'">
| <STRING_LITERAL: "\"" (~["\"","\\","\n","\r"]("0"-"7"].md))) | "\\" ([| ["0"-"7"]("n","t","b","r","f","\\","\'","\""].md) ([
| ["0"-"3"]("0"-"7"].md)?) ["0"-"7"]("0"-"7".md)"0"-"7"])))** "\"" | "\'" (~[| "\\" (["n","t","b","r","f","\\","\'","\""]("\'"
,"\\","\n","\r"].md) | [(["0"-"7"]("0"-"7"].md))? | ["0"-"7"]("0"-"7".md)"0"-"3"]) ["\'">
}

/* SEPARATORS */<DEFAULT> TOKEN : {
<LPAREN: "(">
| <RPAREN: ")">
| <LBRACE: "{">
| <RBRACE: "}">
| <LBRACKET: "[">
| <RBRACKET: "]("0"-"7"].md))*)">
| <SEMICOLON: ";">
| <COMMA: ",">
| <DOT: ".">
| <AT: "@">
}

/** OPERATORS **/<DEFAULT> TOKEN : {
<EQ: "=">
| <LT: "<">
| <GT: ">">
| <BANG: "!">
| <TILDE: "~">
| <HOOK: "?">
| <COLON: ":">
| <LE: "<=">
| <GE: ">=">
| <NE: "!=">
| <NEQ: "<>">
| <SC_OR: "||">
| <SC_AND: "&&">
| <INCR: "++">
| <DECR: "--">
| <PLUS: "+">
| <MINUS: "-">
| <STAR: "**">
| <SLASH: "/">
| <BIT_AND: "&">
| <BIT_OR: "|">
| <XOR: "^">
| <REM: "%">
| <LSHIFT: "<<">
| <PLUSASSIGN: "+=">
| <MINUSASSIGN: "-=">
| <STARASSIGN: "**=">
| <SLASHASSIGN: "/=">
| <ANDASSIGN: "&=">
| <ORASSIGN: "|=">
| <XORASSIGN: "^=">
| <REMASSIGN: "%=">
| <LSHIFTASSIGN: "<<=">
| <RSIGNEDSHIFTASSIGN: ">>=">
| <RUNSIGNEDSHIFTASSIGN: ">>>=">
| <ELLIPSIS: "...">
| <NOT: ("N" | "n") ("O" | "o") ("T" | "t")>
| <LIKE: ("L" | "l") ("I" | "i") ("K" | "k") ("E" | "e")>
| <IS: "is" | "IS" | "Is" | "iS">
| <IN: "in" | "IN" | "In" | "iN">
| <BETWEEN: ("B" | "b") ("E" | "e") ("T" | "t") ("W" | "w") ("E" | "e") ("E" | "e") ("N" | "n")>
| <CONTAINS: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("N" | "n") ("S" | "s")>
| <CONTAINSALL: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("N" | "n") ("S" | "s") ("A" | "a") ("
L" | "l") ("L" | "l")>
| <CONTAINSKEY: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("N" | "n") ("S" | "s") ("K" | "k") ("
E" | "e") ("Y" | "y")>
| <CONTAINSVALUE: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("N" | "n") ("S" | "s") ("V" | "v")
("A" | "a") ("L" | "l") ("U" | "u") ("E" | "e")>
| <CONTAINSTEXT: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("N" | "n") ("S" | "s") ("T" | "t") (
"E" | "e") ("X" | "x") ("T" | "t")>
| <MATCHES: ("M" | "m") ("A" | "a") ("T" | "t") ("C" | "c") ("H" | "h") ("E" | "e") ("S" | "s")>
| <TRAVERSE: ("T" | "t") ("R" | "r") ("A" | "a") ("V" | "v") ("E" | "e") ("R" | "r") ("S" | "s") ("E" | "e")>
}

<DEFAULT> TOKEN : {
<IDENTIFIER: <LETTER> (<PART_LETTER>)**>
| <#LETTER: [| <#PART_LETTER: ["0"-"9","A"-"Z","_","a"-"z"]("A"-"Z","_","a"-"z"]>.md)>
}

NON-TERMINALS
    Rid    :=    "#" <INTEGER_LITERAL> <COLON> <INTEGER_LITERAL>
        |    <INTEGER_LITERAL> <COLON> <INTEGER_LITERAL>
/** Root production. **/    OrientGrammar    :=    Statement <EOF>
    Statement    :=    ( SelectStatement | DeleteStatement | InsertStatement | UpdateStatement )
    SelectStatement    :=    <SELECT> ( Projection )? <FROM> FromClause ( <WHERE> WhereClause )? ( OrderBy )? ( Limit )? ( Ran
ge )?
    DeleteStatement    :=    <DELETE> <FROM> <IDENTIFIER> ( <WHERE> WhereClause )?
    UpdateStatement    :=    <UPDATE> ( <IDENTIFIER> | Cluster | IndexIdentifier ) ( ( <SET> UpdateItem ( "," UpdateItem )** )
 ) ( <WHERE> WhereClause )?
    UpdateItem    :=    <IDENTIFIER> <EQ> ( <NULL> | <STRING_LITERAL> | Rid | <INTEGER_LITERAL> | <FLOATING_POINT_LITERAL> | <
CHARACTER_LITERAL> | <LBRACKET> Rid ( "," Rid )** <RBRACKET> )
    UpdateAddItem    :=    <IDENTIFIER> <EQ> ( <STRING_LITERAL> | Rid | <INTEGER_LITERAL> | <FLOATING_POINT_LITERAL> | <CHARAC
TER_LITERAL> | <LBRACKET> Rid ( "," Rid )** <RBRACKET> )
    InsertStatement    :=    <INSERT> <INTO> ( <IDENTIFIER> | Cluster ) <LPAREN> <IDENTIFIER> ( "," <IDENTIFIER> ) <RPAREN> <V
ALUES> <LPAREN> InsertExpression ( "," InsertExpression ) <RPAREN>
    InsertExpression    :=    <NULL>
        |    <STRING_LITERAL>
        |    <INTEGER_LITERAL>
        |    <FLOATING_POINT_LITERAL>
        |    Rid
        |    <CHARACTER_LITERAL>
        |    <LBRACKET> Rid ( "," Rid )** <RBRACKET>
    InputParameter    :=    "?"
    Projection    :=    ProjectionItem ( "," ProjectionItem )**
    ProjectionItem    :=    "**"
        |    ( ( <NULL> | <INTEGER_LITERAL> | <STRING_LITERAL> | <FLOATING_POINT_LITERAL> | <CHARACTER_LITERAL> | FunctionCall
 | DottedIdentifier | RecordAttribute | ThisOperation | InputParameter ) ( <AS> Alias )? )
    FilterItem    :=    <NULL>
        |    Any
        |    All
        |    <INTEGER_LITERAL>
        |    <STRING_LITERAL>
        |    <FLOATING_POINT_LITERAL>
        |    <CHARACTER_LITERAL>
        |    FunctionCall
        |    DottedIdentifier
        |    RecordAttribute
        |    ThisOperation
        |    InputParameter
    Alias    :=    <IDENTIFIER>
    Any    :=    "any()"
    All    :=    "all()"
    RecordAttribute    :=    <RECORD_ATTRIBUTE>
    ThisOperation    :=    <THIS> ( FieldOperator )**
    FunctionCall    :=    <IDENTIFIER> <LPAREN> ( "**" | ( FilterItem ( "," FilterItem )** ) ) <RPAREN> ( FieldOperator )**
    FieldOperator    :=    ( <DOT> <IDENTIFIER> <LPAREN> ( FilterItem ( "," FilterItem )** )? <RPAREN> )
        |    ( "[<STRING_LITERAL> "](".md)" )
    DottedIdentifier    :=    <IDENTIFIER> ( "[WhereClause "](".md)" )+
        |    <IDENTIFIER> ( FieldOperator )+
        |    <IDENTIFIER> ( <DOT> DottedIdentifier )?
    FromClause    :=    FromItem
    FromItem    :=    Rid
        |    <LBRACKET> Rid ( "," Rid )** <RBRACKET>
        |    Cluster
        |    IndexIdentifier
        |    <IDENTIFIER>
    Cluster    :=    "cluster:" <IDENTIFIER>
    IndexIdentifier    :=    "index:" <IDENTIFIER>
    WhereClause    :=    OrBlock
    OrBlock    :=    AndBlock ( <OR> AndBlock )**
    AndBlock    :=    ( NotBlock ) ( <AND> ( NotBlock ) )**
    NotBlock    :=    ( <NOT> )? ( ConditionBlock | ParenthesisBlock )
    ParenthesisBlock    :=    <LPAREN> OrBlock <RPAREN>
    ConditionBlock    :=    TraverseCondition
        |    IsNotNullCondition
        |    IsNullCondition
        |    BinaryCondition
        |    BetweenCondition
        |    ContainsCondition
        |    ContainsTextCondition
        |    MatchesCondition
    CompareOperator    :=    EqualsCompareOperator
        |    LtOperator
        |    GtOperator
        |    NeOperator
        |    NeqOperator
        |    GeOperator
        |    LeOperator
        |    InOperator
        |    NotInOperator
        |    LikeOperator
        |    ContainsKeyOperator
        |    ContainsValueOperator
    LtOperator    :=    <LT>
    GtOperator    :=    <GT>
    NeOperator    :=    <NE>
    NeqOperator    :=    <NEQ>
    GeOperator    :=    <GE>
    LeOperator    :=    <LE>
    InOperator    :=    <IN>
    NotInOperator    :=    <NOT> <IN>
    LikeOperator    :=    <LIKE>
    ContainsKeyOperator    :=    <CONTAINSKEY>
    ContainsValueOperator    :=    <CONTAINSVALUE>
    EqualsCompareOperator    :=    <EQ>
    BinaryCondition    :=    FilterItem CompareOperator ( Rid | FilterItem )
    BetweenCondition    :=    FilterItem <BETWEEN> FilterItem <AND> FilterItem
    IsNullCondition    :=    FilterItem <IS> <NULL>
    IsNotNullCondition    :=    FilterItem <IS> <NOT> <NULL>
    ContainsCondition    :=    FilterItem <CONTAINS> <LPAREN> OrBlock <RPAREN>
    ContainsAllCondition    :=    FilterItem <CONTAINSALL> <LPAREN> OrBlock <RPAREN>
    ContainsTextCondition    :=    FilterItem <CONTAINSTEXT> ( <STRING_LITERAL> | DottedIdentifier )
    MatchesCondition    :=    FilterItem <MATCHES> <STRING_LITERAL>
    TraverseCondition    :=    <TRAVERSE> ( <LPAREN> <INTEGER_LITERAL> ( "," <INTEGER_LITERAL> ( "," TraverseFields )? )? <RPA
REN> )? <LPAREN> OrBlock <RPAREN>
    TraverseFields    :=    <STRING_LITERAL>
    OrderBy    :=    <ORDER> <BY> <IDENTIFIER> ( "," <IDENTIFIER> )** ( <DESC> | <ASC> )?
    Limit    :=    <LIMIT> <INTEGER_LITERAL>
    Range    :=    <RANGE> Rid ( "," Rid )?

DOCUMENT END
```

Entry Points Since OrientDB v 1.7

The entry points for creating a new Index Engine are two:

- OIndexFactory
- OIndexEngine

Implementing OIndexFactory

Create your own facory that implements OIndexFactory.

In your factory you have to declare:

1. Which types of index you support
2. Which types of algorithms you support

and you have to implements the createIndex method

Example of custom factory for Lucene Indexing

```
package com.orientechnologies.lucene;

import java.util.Collections;
import java.util.HashSet;
import java.util.Set;

import com.orientechnologies.lucene.index.OLuceneFullTextIndex;
import com.orientechnologies.lucene.index.OLuceneSpatialIndex;
import com.orientechnologies.lucene.manager.*;
import com.orientechnologies.lucene.shape.OShapeFactoryImpl;
import com.orientechnologies.orient.core.db.record.ODatabaseRecord;
import com.orientechnologies.orient.core.db.record.OIdentifiable;
import com.orientechnologies.orient.core.exception.OConfigurationException;
import com.orientechnologies.orient.core.index.OIndexFactory;
import com.orientechnologies.orient.core.index.OIndexInternal;
import com.orientechnologies.orient.core.metadata.schema.OClass;
import com.orientechnologies.orient.core.record.impl.ODocument;

/**
 * Created by enricorisa on 21/03/14.
 */
public class OLuceneIndexFactory implements OIndexFactory {

  private static final Set<String> TYPES;
  private static final Set<String> ALGORITHMS;
  public static final String       LUCENE_ALGORITHM = "LUCENE";

  static {
    final Set<String> types = new HashSet<String>();
    types.add(OClass.INDEX_TYPE.UNIQUE.toString());
    types.add(OClass.INDEX_TYPE.NOTUNIQUE.toString());
    types.add(OClass.INDEX_TYPE.FULLTEXT.toString());
    types.add(OClass.INDEX_TYPE.DICTIONARY.toString());
    types.add(OClass.INDEX_TYPE.SPATIAL.toString());
    TYPES = Collections.unmodifiableSet(types);
  }

  static {
    final Set<String> algorithms = new HashSet<String>();
    algorithms.add(LUCENE_ALGORITHM);
    ALGORITHMS = Collections.unmodifiableSet(algorithms);
  }

  public OLuceneIndexFactory() {
  }

  @Override
  public Set<String> getTypes() {
    return TYPES;
  }

  @Override
  public Set<String> getAlgorithms() {
    return ALGORITHMS;
  }

  @Override
  public OIndexInternal<?> createIndex(ODatabaseRecord oDatabaseRecord, String indexType, String algorithm,
      String valueContainerAlgorithm, ODocument metadata) throws OConfigurationException {
    return createLuceneIndex(oDatabaseRecord, indexType, valueContainerAlgorithm, metadata);
  }

  private OIndexInternal<?> createLuceneIndex(ODatabaseRecord oDatabaseRecord, String indexType, String valueContainerAlgorith
m,
      ODocument metadata) {
    if (OClass.INDEX_TYPE.FULLTEXT.toString().equals(indexType)) {
      return new OLuceneFullTextIndex(indexType, LUCENE_ALGORITHM, new OLuceneIndexEngine<Set<OIdentifiable>>(
          new OLuceneFullTextIndexManager(), indexType), valueContainerAlgorithm, metadata);
    } else if (OClass.INDEX_TYPE.SPATIAL.toString().equals(indexType)) {
      return new OLuceneSpatialIndex(indexType, LUCENE_ALGORITHM, new OLuceneIndexEngine<Set<OIdentifiable>>(
          new OLuceneSpatialIndexManager(new OShapeFactoryImpl()), indexType), valueContainerAlgorithm);
    }
    throw new OConfigurationException("Unsupported type : " + indexType);
  }
}
```

To plug your factory create in your project under META-INF/services a text file called `com.orientechnologies.orient.core.index.OIndexFactory` and write inside your factory

Example

```
com.orientechnologies.lucene.OLuceneIndexFactory
```

Implementing OIndexEngine

To write a new Index Engine implements the OIndexEngine interface.

The main methods are:

- get
- put

get `V get(Object key);`

You have to return a Set of OIdentifiable or OIdentifiable if your index is unique, associated with the key. The key could be:

- The value if you are indexing a single field (Integer,String,Double..etc).
- OCompositeKey if you are indexing two or more fields

put `void put(Object key, V value);`

- The key is the value to be indexed. Could be as written before
- The value is a Set of OIdentifiable or OIdentifiable associated with the key

Create Index from SQL

You can create an index with your Index Engine with sql with this syntax

```
CREATE INDEX Foo.bar ON Foo (bar) NOTUNIQUE ENGINE CUSTOM
```

where CUSTOM is the name of your index engine

Caching

OrientDB has several caching mechanisms that act at different levels. Look at this picture:

- **Local cache** is one per database instance (and per thread in multi-thread environment)
- **Storage**, it could cache depending on the implementation. This is the case for the **Local Storage** (disk-based) that caches file reads to reduce I/O requests
- **Command Cache**

How cache works?

Local Mode (embedded database)

When the client application asks for a record OrientDB checks:

- if a **transaction** has begun then it searches inside the transaction for changed records and returns it if found
- if the **Local cache** is enabled and contains the requested record then return it
- otherwise, at this point the record is not in cache, then asks for it to the **Storage** (disk, memory)

Client-Server Mode (remote database)

When the client application asks for a record OrientDB checks:

- if a **transaction** has begun then it searches inside the transaction for changed records and returns it if found
- if the **Local cache** is enabled and contains the requested record then return it
- otherwise, at this point the record is not in cache, then asks for it to the **Server** through a TCP/IP call
- in the server, if the **Local cache** is enabled and contains the requested record then return it
- otherwise, at this point the record is also not cached in the server, then asks for it to the **Storage** (disk, memory)

Record cache

Local cache

Local cache acts at database level. Each database instance has a Local cache enabled by default. This cache keeps the used records. Records will be removed from heap if two conditions will be satisfied:

1. There are no links to these records from outside of the database
2. The Java Virtual Machine doesn't have enough memory to allocate new data

Empty Local cache

To remove all the records in Local cache you can invoke the `invalidate()` method:

```
db.getLocalCache().invalidate();
```

Transactions

A transaction comprises a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. Transactions in a database environment have two main purposes:

- to provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status
- to provide isolation between programs accessing a database concurrently. If this isolation is not provided, the program's outcome are possibly erroneous.

A database transaction, by definition, must be atomic, consistent, isolated and durable. Database practitioners often refer to these properties of database transactions using the acronym ACID. --- Wikipedia

OrientDB is an ACID compliant DBMS.

> **NOTE**: OrientDB keeps the transaction on client RAM, so the transaction size is affected by the available RAM (Heap memory) on JVM. For transactions involving many records, consider to split it in multiple transactions.

ACID properties

Atomicity

"Atomicity requires that each transaction is 'all or nothing': if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes. To the outside world, a committed transaction appears (by its effects on the database) to be indivisible ("atomic"), and an aborted transaction does not happen." - WikiPedia

Consistency

"The consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including but not limited to constraints, cascades, triggers, and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors do not violate any defined rules." - WikiPedia

OrientDB uses the MVCC to assure consistency. The difference between the management of MVCC on transactional and not-transactional cases is that with transactional, the exception rollbacks the entire transaction before to be caught by the application.

Look at this example:

Sequence	Client/Thread 1	Client/Thread 2	Version of record X
1	Begin of Transaction		
2	read(x)		10
3		Begin of Transaction	
4		read(x)	10
5		write(x)	10
6		commit	10 -> 11
7	write(x)		10
8	commit		10 -> 11 = Error, in database x already is at 11

Isolation

"The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e. one after the other. Providing isolation is the main goal of concurrency control. Depending on concurrency control method, the effects of an incomplete transaction might not even be visible to another transaction." - WikiPedia

OrientDB has different levels of isolation based on settings and configuration:

- `READ COMMITTED` , the default and the only one available with `remote` protocol
- `REPEATABLE READS` , allowed only with `plocal` and `memory` protocols. This mode consumes more memory than `READ COMMITTED` , because any read, query, etc. keep the records in memory to assure the same copy on further access

To change default Isolation Level, use the Java API:

```
db.begin()
db.getTransaction().setIsolationLevel(OTransaction.ISOLATION_LEVEL.REPEATABLE_READ);
```

Using `remote` access all the commands are executed on the server, so out of transaction scope. Look below for more information.

Look at this examples:

Sequence	Client/Thread 1	Client/Thread 2
1	Begin of Transaction	
2	read(x)	
3		Begin of Transaction
4		read(x)
5		write(x)
6		commit
7	read(x)	
8	commit	

At operation 7 the client 1 continues to read the same version of x read in operation 2.

Sequence	Client/Thread 1	Client/Thread 2
1	Begin of Transaction	
2	read(x)	
3		Begin of Transaction
4		read(y)
5		write(y)
6		commit
7	read(y)	
8	commit	

At operation 7 the client 1 reads the version of y which was written at operation 6 by client 2. This is because it never reads y before.

Breaking of ACID properties when using remote protocol and Commands (SQL, Gremlin, JS, etc)

Transactions are client-side only until the commit. This means that if you're using the "remote" protocol the server can't see local changes.

In this scenario you can have different isolation levels with commands. This issue will be solved with OrientDB v3.0 where the transaction will be flushed to the server before to execute the command.

Durability

"Durability means that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter). To defend against power loss, transactions (or their effects) must be recorded in a non-volatile memory." - WikiPedia

Fail-over

An OrientDB instance can fail for several reasons:

- HW problems, such as loss of power or disk error
- SW problems, such as a Operating System crash
- Application problem, such as a bug that crashes your application that is connected to the Orient engine.

You can use the OrientDB engine directly in the same process of your application. This gives superior performance due to the lack of inter-process communication. In this case, should your application crash (for any reason), the OrientDB Engine also crashes.

If you're using an OrientDB Server connected remotely, if your application crashes the engine continue to work, but any pending transaction owned by the client will be rolled back.

Auto-recovery

At start-up the OrientDB Engine checks to if it is restarting from a crash. In this case, the auto-recovery phase starts which rolls back all pending transactions.

OrientDB has different levels of durability based on storage type, configuration and settings.

Transaction types

No Transaction

Default mode. Each operation is executed instantly.

Calls to `begin()` , `commit()` and `rollback()` have no effect.

Optimistic Transaction

This mode uses the well known Multi Version Control System (MVCC) by allowing multiple reads and writes on the same records. The integrity check is made on commit. If the record has been saved by another transaction in the interim, then an OConcurrentModificationException will be thrown. The application can choose either to repeat the transaction or abort it.

> **NOTE**: OrientDB keeps the transaction on client RAM, so the transaction size is affected by the available RAM (Heap) memory on JVM. For transactions involving many records, consider to split it in multiple transactions.

With Graph API transaction begins automatically, with Document API is explicit by using the `begin()` method. With Graphs you can change the consistency level.

Example with Document API:

```
db.open("remote:localhost:7777/petshop");

try{
  db.begin(TXTYPE.OPTIMISTIC);
  ...
  // WRITE HERE YOUR TRANSACTION LOGIC
  ...
  db.commit();
}catch( Exception e ){
  db.rollback();
} finally{
  db.close();
}
```

In Optimistic transaction new records take temporary Record IDs to avoid to ask to the server a new Record ID every time. Temporary Record IDs have Cluster Id -1 and Cluster Position < -1. When a new transaction begun the counter is reset to -1:-2. So if you create 3 new records you'll have:

- -1:-2
- -1:-3
- -1:-4

At commit time, these temporary records Record IDs will be converted in the final ones.

Pessimistic Transaction

This mode is not yet supported by the engine.

Nested transactions and propagation

OrientDB doesn't support nested transaction. If further `begin()` are called after a transaction is already begun, then the current transaction keeps track of call stack to let to the final commit() call to effectively commit the transaction. Look at Transaction Propagation more information.

Record IDs

OrientDB uses temporary Record IDs with transaction as scope that will be transformed to finals once the transactions is successfully committed to the database. This avoid to ask for a free slot every time a client creates a record.

Tuning

In some situations transactions can improve performance, typically in the client/server scenario. If you use an Optimistic Transaction, the OrientDB engine optimizes the network transfer between the client and server, saving both CPU and bandwidth.

For further information look at Transaction tuning to know more.

Distributed environment

Transactions can be committed across a distributed architecture. Look at Distributed Transactions for more information.

Hooks (Triggers)

Hooks work like triggers and enable a user's application to intercept internal events before and after each CRUD operation against records. You can use them to write custom validation rules, to enforce security, or even to orchestrate external events like replicating against a Relational DBMS.

OrientDB supports two kinds of Hooks:

- Dynamic Hooks, defined at the schema and/or document level
- Native Java Hooks, defined as Java classes

What use? Pros/Cons?

Depends on your goal: Java Hooks are faster. Write a Java Hook if you need the best performance on execution. Dynamic Hooks are more flexible, can be changed at run-time, and can run per document if needed, but are slower than Java Hooks.

Dynamic Hooks

Dynamic Hooks are more flexible than Java Hooks because they can be changed at run-time and can run per document if needed, but are slower than Java Hooks. Look at Hooks for more information.

To execute hooks against your documents, let your classes extend the `OTriggered` base class. Then define a custom property for the event you're interested in. The available events are:

- `onBeforeCreate` , called **before** creating a new document
- `onAfterCreate` , called **after** creating a new document
- `onBeforeRead` , called **before** reading a document
- `onAfterRead` , called **after** reading a document
- `onBeforeUpdate` , called **before** updating a document
- `onAfterUpdate` , called **after** updating a document
- `onBeforeDelete` , called **before** deleting a document
- `onAfterDelete` , called **after** deleting a document

Dynamic Hooks can call:

- Functions written in SQL, Javascript, or any other language supported by OrientDB and the JVM
- Java static methods

Class level hooks

Class level hooks are defined for all the documents that relate to a class. Below is an example to setup a hook that acts at the class level against Invoice documents.

```
CREATE CLASS Invoice EXTENDS OTriggered
ALTER CLASS Invoice CUSTOM onAfterCreate=invoiceCreated
```

Now let's create the function `invoiceCreated` in Javascript that prints on the server console the invoice number created.

```
CREATE FUNCTION invoiceCreated "print('\\nInvoice created: ' + doc.field('number'));" LANGUAGE Javascript
```

Now try the hook by creating a new `Invoice` document.

```
INSERT INTO Invoice CONTENT { number: 100, notes: 'This is a test' }
```

And this will appear in the server console:

```
Invoice created: 100
```

Document level hook

If you need to define a special action against only one or more documents let your class extend the `OTriggered` class.

Example: To execute a trigger, as a Javascript function, against an existing Profile class, for all the documents with property `account = 'Premium'` . The trigger will be called to prevent deletion of documents:

```
ALTER CLASS Profile SUPERCLASS OTriggered
UPDATE Profile SET onBeforeDelete = 'preventDeletion' WHERE account = 'Premium'
```

And now let's create the `preventDeletion()` Javascript function.

```
CREATE FUNCTION preventDeletion "throw new java.lang.RuntimeException('Cannot delete Premium profile ' + doc)" LANGUAGE Javasc
ript
```

And now test the hook by trying to delete a `Premium` account.

```
DELETE FROM #12:1

java.lang.RuntimeException: Cannot delete Premium profile profile#12:1{onBeforeDelete:preventDeletion,account:Premium,name:Jil
l} v-1 (<Unknown source>#2) in <Unknown source> at line number 2
```

(Native) Java Hooks

Java Hooks are the fastest hooks. Write a Java Hook if you need the best performance on execution. Look at Hooks for more information.

The ORecordHook interface

A hook is an implementation of the interface ORecordHook:

```
public interface ORecordHook {
  public enum TYPE {
    ANY,
    BEFORE_CREATE, BEFORE_READ, BEFORE_UPDATE, BEFORE_DELETE,
    AFTER_CREATE, AFTER_READ, AFTER_UPDATE, AFTER_DELETE
  };

  public void onTrigger(TYPE iType, ORecord<?> iRecord);
}
```

The ORecordHookAbstract abstract class

OrientDB comes with an abstract implementation of the ORecordHook interface called ORecordHookAbstract.java. It switches the callback event, calling separate methods for each one:

```
public abstract class ORecordHookAbstract implements ORecordHook {
  public void onRecordBeforeCreate(ORecord<?> iRecord){}
  public void onRecordAfterCreate(ORecord<?> iRecord){}
  public void onRecordBeforeRead(ORecord<?> iRecord){}
  public void onRecordAfterRead(ORecord<?> iRecord){}
  public void onRecordBeforeUpdate(ORecord<?> iRecord){}
  public void onRecordAfterUpdate(ORecord<?> iRecord){}
  public void onRecordBeforeDelete(ORecord<?> iRecord){}
  public void onRecordAfterDelete(ORecord<?> iRecord){}
  ...
}
```

The ODocumentHookAbstract abstract class

When you want to catch an event from a Document only, the best way to create a hook is to extend the `ODocumentHookAbstract` abstract class. You can specify what classes you're interested in. In this way the callbacks will be called only on documents of the specified classes. Classes are polymorphic so filtering works against specified classes and all sub-classes.

You can specify only the class you're interested or the classes you want to exclude. Example to include only the `Client` and `Provider` classes:

```
public class MyHook extends ODocumentHookAbstract {
  public MyHook() {
    setIncludeClasses("Client", "Provider");
  }
}
```

Example to get called for all the changes on documents of any class but `Log` :

```
public class MyHook extends ODocumentHookAbstract {
  public MyHook() {
    setExcludeClasses("Log");
  }
}
```

Access to the modified fields

In Hook methods you can access dirty fields and the original values. Example:

```
for( String field : document.getDirtyFields() ) {
  Object originalValue = document.getOriginalValue( field );
  ...
}
```

Self registration

Hooks can be installed on certain database instances, but in most cases you'll need to register it for each instance. To do this programmatically you can intercept the `onOpen()` and `onCreate()` callbacks from OrientDB to install hooks. All you need is to implement the `ODatabaseLifecycleListener` interface. Example:

```
public class MyHook extends ODocumentHookAbstract implements ODatabaseLifecycleListener {
  public MyHook() {
    // REGISTER MYSELF AS LISTENER TO THE DATABASE LIFECYCLE
    Orient.instance().addDbLifecycleListener(this);
  }
  ...
  @Override
  public void onOpen(final ODatabase iDatabase) {
    // REGISTER THE HOOK
    ((ODatabaseComplex<?>)iDatabase).registerHook(this);
  }

  @Override
  public void onCreate(final ODatabase iDatabase) {
    // REGISTER THE HOOK
    ((ODatabaseComplex<?>)iDatabase).registerHook(this);
  }

  @Override
  public void onClose(final ODatabase iDatabase) {
    // REGISTER THE HOOK
    ((ODatabaseComplex<?>)iDatabase).unregisterHook(this);
  }
  ...
  public RESULT onRecordBeforeCreate(final ODocument iDocument) {
    // DO SOMETHING BEFORE THE DOCUMENT IS CREATED
    ...
  }
  ...
}
```

Hook example

In this example the events `before-create` and `after-delete` are called during the `save()` of the `Profile` object where:

- `before-create` is used to check custom validation rules
- `after-delete` is used to maintain the references valid

```
public class HookTest extends ORecordHookAbstract {
  public saveProfile(){
    ODatabaseObjectTx database = new ODatabaseObjectTx("remote:localhost/demo");
    database.open("writer", "writer");

    // REGISTER MYSELF AS HOOK
    database.registerHook(this);

    ...
    p = new Profile("Luca");
    p.setAge(10000);
    database.save(p);
    ...
  }

  /**
   * Custom validation rules
   */
  @Override
  public void onRecordBeforeCreate(ORecord<?> iRecord){
    if( iRecord instanceof ODocument ){
      ODocument doc = (ODocument) iRecord;
      Integer age = doc .field( "age" );
      if( age != null && age > 130 )
        throw new OValidationException("Invalid age");
    }
  }

  /**
   * On deletion removes the reference back.
   */
  @Override
  public void onRecordAfterDelete(ORecord<?> iRecord){
    if( iRecord instanceof ODocument ){
      ODocument doc = (ODocument) iRecord;

      Set<OIdentifiable> friends = doc.field( "friends" );
      if( friends != null ){
        for( OIdentifiable friend : friends ){
          Set<OIdentifiable> otherFriends = ((ODocument)friend.getRecord()).field("friends");
          if( friends != null )
            friends.remove( iRecord );
        }
      }
    }
  }
}
```

For more information take a look to the HookTest.java source code.

Install server-side hooks

To let a hook be executed in the Server space you have to register it in the server `orientdb-server-config.xml` configuration file.

Write your hook

Example of a hook to execute custom validation rules:

```
public class CustomValidationRules implements ORecordHook{
  /**
   * Apply custom validation rules
   */
  public boolean onTrigger(final TYPE iType, final ORecord<?> iRecord) {
    if( iRecord instanceof ODocument ){
      ODocument doc = (ODocument) iRecord;

      switch( iType ){
        case BEFORE_CREATE:
        case BEFORE_UPDATE: {
          if( doc.getClassName().equals("Customer") ){
            Integer age = doc .field( "age" );
            if( age != null && age > 130 )
              throw new OValidationException("Invalid age");
          }
          break;
        }

        case BEFORE_DELETE: {
          if( doc.getClassName().equals("Customer") ){
            final ODatabaseRecord db = ODatabaseRecordThreadLocal.INSTANCE.get();
            if( !db.getUser().getName().equals( "admin" ) )
              throw new OSecurityException("Only admin can delete customers");
          }
          break;
        }
    }
  }
}
```

Deploy the hook

Once implemented create a `.jar` file containing your class and put it under the `$ORIENTDB_HOME/lib` directory.

Register it in the server configuration

Change the `orientdb-server-config.xml` file adding your hook inside the `<hooks>` tag. The position can be one of following values `FIRST` , `EARLY` , `REGULAR` , `LATE` , `LAST` :

```
<hook class="org.orientdb.test.MyHook" position="REGULAR"/>
```

Configurable hooks

If your hook must be configurable with external parameters write the parameters in the `orientdb-server-config.xml` file:

```
<hook class="org.orientdb.test.MyHook" position="REGULAR">
    <parameters>
        <parameter name="userCanDelete" value="admin" />
    </parameters>
</hook>
```

And in your Java class implement the config() method to read the parameter:

```
private String userCanDelete;
...
public void config(OServer oServer, OServerParameterConfiguration[] iParams) {
  for (OServerParameterConfiguration param : iParams) {
    if (param.name.equalsIgnoreCase("userCanDelete")) {
      userCanDelete = param.value;
    }
  }
}
...
```

Java Hook Tutorial

One common use case for OrientDB Hooks (a.k.a. database triggers) is to manage created and updated dates for any or all classes (a.k.a. database tables). For example, it is nice to be able to set a CreatedDate field whenever a record is created and set an UpdatedDate field whenever a record is updated, and do it in a way where you implement the logic once at the database layer and never have to worry about it again at the application layer.

The following tutorial will walk you through exactly how to accomplish this use case using an OrientDB Hook and we'll do it from the perspective of a novice Java programmer working on a Windows machine.

Assumptions

It is assumed that you have already downloaded and installed a Java JDK. In my case I downloaded Java JDK version 8 for Windows 64 bit and installed it to folder C:\Program Files\Java\jdk1.8.0_40.

It is also assumed that you have downloaded, installed, and configured a working OrientDB Server. In my case I installed it to folder C:\Program Files\orientdb-community-2.0.5.

Exact instructions for these two steps are outside the scope of this tutorial.

Initial Server Configuration File

My OrientDB server configuration file is located at C:\Program Files\orientdb-community-2.0.5\config\orientdb-server-config.xml and is configured like this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<orient-server>
    <handlers>
        <handler class="com.orientechnologies.orient.graph.handler.OGraphServerHandler">
            <parameters>
                <parameter value="true" name="enabled"/>
                <parameter value="50" name="graph.pool.max"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.orient.server.hazelcast.OHazelcastPlugin">
            <parameters>
                <parameter value="false" name="enabled"/>
                <parameter value="${ORIENTDB_HOME}/config/default-distributed-db-config.json" name="configuration.db.default"/>

                <parameter value="${ORIENTDB_HOME}/config/hazelcast.xml" name="configuration.hazelcast"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.orient.server.handler.OJMXPlugin">
            <parameters>
                <parameter value="false" name="enabled"/>
                <parameter value="true" name="profilerManaged"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.orient.server.handler.OAutomaticBackup">
            <parameters>
                <parameter value="false" name="enabled"/>
                <parameter value="4h" name="delay"/>
                <parameter value="backup" name="target.directory"/>
                <parameter value="${DBNAME}-${DATE:yyyyMMddHHmmss}.zip" name="target.fileName"/>
                <parameter value="9" name="compressionLevel"/>
                <parameter value="1048576" name="bufferSize"/>
                <parameter value="" name="db.include"/>
                <parameter value="" name="db.exclude"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.orient.server.handler.OServerSideScriptInterpreter">
            <parameters>
                <parameter value="false" name="enabled"/>
                <parameter value="SQL,JAVASCRIPT" name="allowedLanguages"/>
            </parameters>
        </handler>
        <handler class="com.orientechnologies.orient.server.token.OrientTokenHandler">
            <parameters>
                <parameter value="true" name="enabled"/>
                <parameter value="INSERT YOUR OWN HERE" name="oAuth2Key"/>
                <parameter value="60" name="sessionLength"/>
                <parameter value="HmacSHA256" name="encryptionAlgorithm"/>
            </parameters>
        </handler>
    </handlers>
    <network>
        <sockets>
            <socket implementation="com.orientechnologies.orient.server.network.OServerSSLSocketFactory" name="ssl">
                <parameters>
                    <parameter value="false" name="network.ssl.clientAuth"/>
                    <parameter value="config/cert/orientdb.ks" name="network.ssl.keyStore"/>
                    <parameter value="password" name="network.ssl.keyStorePassword"/>
                    <parameter value="config/cert/orientdb.ks" name="network.ssl.trustStore"/>
                    <parameter value="password" name="network.ssl.trustStorePassword"/>
                </parameters>
            </socket>
            <socket implementation="com.orientechnologies.orient.server.network.OServerSSLSocketFactory" name="https">
                <parameters>
                    <parameter value="false" name="network.ssl.clientAuth"/>
                    <parameter value="config/cert/orientdb.ks" name="network.ssl.keyStore"/>
                    <parameter value="password" name="network.ssl.keyStorePassword"/>
                    <parameter value="config/cert/orientdb.ks" name="network.ssl.trustStore"/>
                    <parameter value="password" name="network.ssl.trustStorePassword"/>
                </parameters>
            </socket>
        </sockets>
        <protocols>
            <protocol implementation="com.orientechnologies.orient.server.network.protocol.binary.ONetworkProtocolBinary" name=
"binary"/>
            <protocol implementation="com.orientechnologies.orient.server.network.protocol.http.ONetworkProtocolHttpDb" name="
http"/>
        </protocols>
        <listeners>
            <listener protocol="binary" socket="default" port-range="2424-2430" ip-address="0.0.0.0"/>
            <listener protocol="http" socket="default" port-range="2480-2490" ip-address="0.0.0.0">
                <commands>
                    <command implementation="com.orientechnologies.orient.server.network.protocol.http.command.get.OServerComm
andGetStaticContent" pattern="GET|www GET|studio/ GET| GET|*.htm GET|*.html GET|*.xml GET|*.jpeg GET|*.jpg GET|*.png GET|*.gif
 GET|*.js GET|*.css GET|*.swf GET|*.ico GET|*.txt GET|*.otf GET|*.pjs GET|*.svg GET|*.json GET|*.woff GET|*.ttf GET|*.svgz" st
ateful="false">
                        <parameters>
                            <entry value="Cache-Control: no-cache, no-store, max-age=0, must-revalidate\r\nPragma: no-cache" n
ame="http.cache:*.htm *.html"/>
                            <entry value="Cache-Control: max-age=120" name="http.cache:default"/>
                        </parameters>
                    </command>
                    <command implementation="com.orientechnologies.orient.graph.server.command.OServerCommandGetGephi" pattern=
"GET|gephi/*" stateful="false"/>
                </commands>
                <parameters>
                    <parameter value="utf-8" name="network.http.charset"/>
                </parameters>
            </listener>
        </listeners>
    </network>
    <storages/>
    <users>
        <user resources="*" password="INSERT YOUR OWN HERE" name="root"/>
        <user resources="connect,server.listDatabases,server.dblist" password="guest" name="guest"/>
    </users>
    <properties>
        <entry value="1" name="db.pool.min"/>
        <entry value="50" name="db.pool.max"/>
        <entry value="true" name="profiler.enabled"/>
        <entry value="info" name="log.console.level"/>
        <entry value="fine" name="log.file.level"/>
        <entry name="server.database.path" value="/data/orientdb" />
    </properties> [
</orient-server>
```

Step 1 - Install Apache Maven

Apache Maven is a useful tool for people wishing to write and compile Java programs, which you will need to do in order to create an OrientDB Java Hook.

You can download Apache Maven from https://maven.apache.org/download.cgi. Follow the installation instructions until you can open a command prompt and successfully run the command

```
$ mvn --help
```

Step 2 - Create a new Maven project

Open a command prompt and change directory to some root folder where you like to code. Feel free to use a directory inside the a repository you already have.

Before you create a Maven project, it is useful to think about how to you want to name your code package. Package organization is usually a heated discussion but I just like to keep it unique but simple. I choose to put all of my OrientDB java hooks in a package called river.hooks. Therefore, I might call my first hook river.hooks.hook1 and my next hook river.hooks.hook2.

Now create a new Maven project in the folder location you have selected by running the following command:

```
$ mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes \
      -DgroupId=river.hooks -DartifactId=hooks
```

You'll notice that the result of this command is a brand new directory structure created underneath the folder in which you ran the command. Take special note that Maven has created a file called .\hooks\pom.xml and a folder called .\hooks\src\main\java\river\hooks.

Edit pom.xml

The thing you need to pay attention to in this file is the section called dependencies. As your OrientDB Java Hook will leverage OrientDB code, you need to tell Maven to download and cache the OrientDB code libraries that your hook needs. Do this by adding the following to your pom.xml file:

```
  <dependencies>
    ...
    <dependency>
      <groupId>com.orientechnologies</groupId>
      <artifactId>orientdb-core</artifactId>
      <version>2.0.5</version>
    </dependency>
  </dependencies>
```

Create hook file(s)

Now that Maven knows that your code will build upon the oriendb-core code libraries, you can start writing your Hook file(s). Go to folder .\hooks\src\main\java\river\hooks. This is the folder where you will put your .java hook files. Go ahead and delete the placeholder App.java file that Maven created and which you don't need.

Let's start out by adding a `HookTest.java` file as follows:

```
package river.hooks;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.locks.ReentrantLock;
import com.orientechnologies.orient.core.hook.ODocumentHookAbstract;
import com.orientechnologies.orient.core.hook.ORecordHook;
import com.orientechnologies.orient.core.hook.ORecordHookAbstract;
import com.orientechnologies.orient.core.db.ODatabaseLifecycleListener;
import com.orientechnologies.orient.core.db.ODatabase;
import com.orientechnologies.orient.core.record.ORecord;
import com.orientechnologies.orient.core.record.impl.ODocument;

public class HookTest extends ODocumentHookAbstract implements ORecordHook {
  public HookTest() {
    setExcludeClasses("Log"); //if comment out this one line or leave off the constructor entirely then OrientDB fails on ever
y command
  }

  @Override
  public DISTRIBUTED_EXECUTION_MODE getDistributedExecutionMode() {
    return DISTRIBUTED_EXECUTION_MODE.BOTH;
  }

  public RESULT onRecordBeforeCreate( ODocument iDocument ) {
    System.out.println("Ran create hook");
    return ORecordHook.RESULT.RECORD_NOT_CHANGED;
  }

  public RESULT onRecordBeforeUpdate( ODocument iDocument ) {
    System.out.println("Ran update hook");
    return ORecordHook.RESULT.RECORD_NOT_CHANGED;
  }

}
```

What this sample code does is print out the appropriate comment every time you create or update a record of that class.

Let's add one more hook file `setCreatedUpdatedDates.java` as follows:

```
package river.hooks;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.locks.ReentrantLock;
import com.orientechnologies.orient.core.hook.ODocumentHookAbstract;
import com.orientechnologies.orient.core.hook.ORecordHook;
import com.orientechnologies.orient.core.hook.ORecordHookAbstract;
import com.orientechnologies.orient.core.db.ODatabaseLifecycleListener;
import com.orientechnologies.orient.core.db.ODatabase;
import com.orientechnologies.orient.core.record.ORecord;
import com.orientechnologies.orient.core.record.impl.ODocument;

public class setCreatedUpdatedDates extends ODocumentHookAbstract implements ORecordHook {
  public setCreatedUpdatedDates() {
    setExcludeClasses("Log"); //if comment out this one line or leave off the constructor entirely then OrientDB fails on ever
y command
  }

  @Override
  public DISTRIBUTED_EXECUTION_MODE getDistributedExecutionMode() {
    return DISTRIBUTED_EXECUTION_MODE.BOTH;
  }

  public RESULT onRecordBeforeCreate( ODocument iDocument ) {
    if ((iDocument.getClassName().charAt(0) == 't') || (iDocument.getClassName().charAt(0)=='r')) {
      iDocument.field("CreatedDate", System.currentTimeMillis() / 1000l);
      iDocument.field("UpdatedDate", System.currentTimeMillis() / 1000l);
      return ORecordHook.RESULT.RECORD_CHANGED;
    } else {
      return ORecordHook.RESULT.RECORD_NOT_CHANGED;
    }
  }

  public RESULT onRecordBeforeUpdate( ODocument iDocument ) {
    if ((iDocument.getClassName().charAt(0) == 't') || (iDocument.getClassName().charAt(0)=='r')) {
      iDocument.field("UpdatedDate", System.currentTimeMillis() / 1000l);
      return ORecordHook.RESULT.RECORD_CHANGED;
    } else {
      return ORecordHook.RESULT.RECORD_NOT_CHANGED;
    }
  }

}
```

What this code does is look for any class that starts with the letters r or t and sets CreatedDate and UpdatedDate when the record gets created and sets just UpdatedDate every time the record gets updated.

Step 3 - Compile your Java hooks

In a command prompt, go to your .\hooks file and run the following commands:

```
$ mvn compile
```

This compiles your hook source code into java .class files.

```
$ mvn package
```

This zips up your compile code files with the needed directory structure into a file called .\target\hooks-1.0-SNAPSHOT.jar.

Step 4 - Move your compiled code to where OrientDB Server can find it

Finally, you need to copy your finished .jar file to the directory where your OrientDB server will look for them. This means the .\lib folder under your OrientDB Server root directory like this:

```
$ copy /Y .\target\hooks-1.0-SNAPSHOT.jar "\Program Files\orientdb-community-2.0.5\lib"
```

Step 5 - Enable your test hook in the OrientDB Server configuration file

Edit C:\Program Files\orientdb-community-2.0.5\config\orientdb-server-config.xml and add the following section near the end of the file:

```
    <hooks>
        <hook class="river.hooks.HookTest" position="REGULAR"/>
    </hooks>
    ...
</orient-server>
```

Step 6 - Restart your OrientDB Server

Once you restart your OrientDB Server, the hook you defined in orientdb-server-config.xml is now active. Launch an OrientDB console, connect it to your database, and run the following command:

```
INSERT INTO V SET ID = 1;
```

If you review your server output/log you should see the message

```
Ran create hook
```

Now run command:

```
UPDATE V SET ID = 2 WHERE ID = 1;
```

Now your server output should say

```
Ran update hook
```

Step 7 - Enable your real hook in the OrientDB Server configuration file

Edit C:\Program Files\orientdb-community-2.0.5\config\orientdb-server-config.xml and change the hooks section as follows:

```
    <hooks>
        <hook class="river.hooks.setCreatedUpdatedDates" position="REGULAR"/>
    </hooks>
    ...
</orient-server>
```

Step 8 - Restart your OrientDB Server

Now create a new class that starts with the letter `r` or `t` :

```
CREATE CLASS tTest EXTENDS V;
```

Now insert a record:

```
INSERT INTO tTest SET ID = 1
SELECT FROM tTest

----+-----+------+----+-----------+-----------
#   |@RID |@CLASS|ID  |CreatedDate|UpdatedDate
----+-----+------+----+-----------+-----------
0   |#19:0|tTest |1   |1427597275 |1427597275
----+-----+------+----+-----------+-----------
```

Even though you did not specify values to set for `CreatedDate` and `UpdatedDate`, OrientDB has set these fields automatically for you.

Now update the record:

```
UPDATE tTest SET ID = 2 WHERE ID = 1;
SELECT FROM tTest;

----+-----+------+----+-----------+-----------
#   |@RID |@CLASS|ID  |CreatedDate|UpdatedDate
----+-----+------+----+-----------+-----------
0   |#19:0|tTest |2   |1427597275 |1427597306
----+-----+------+----+-----------+-----------
```

You can see that OrientDB has changed the `UpdatedDate` but let the `CreatedDate` unchanged.

Conclusion

OrientDB Java Hooks can be an extremely valuable tool to help automate work you would otherwise have to do in application code. As many DBAs are not always Java experts, hopefully the information contained in this tutorial will give you a head start in feeling comfortable with the technology and empower you to successfully create database triggers as the need arises.

Good luck!

OrientDB Server

OrientDB Server (DB-Server from now) is a multi-threaded Java application that listens to remote commands and executes them against the Orient databases. OrientDB Server supports both binary and HTTP protocols. The first one is used by the Orient native client and the Orient Console. The second one can be used by any languages since it's based on HTTP RESTful API. The HTTP protocol is used also by the OrientDB Studio application.

Starting from v1.7 OrientDB support protected SSL connections.

❗	Even though OrientDB Server is a regular Web Server, it is not recommended to expose it directly on the Internet or public networks. We suggest to always hide OrientDB server in a private network.

Install as a service

OrientDB Server is part of Community and Enterprise distributions. To install OrientDB as service follow the following guides

- Unix, Linux and MacOSX
- Windows

Start the server

To start the server, execute bin/server.sh (or bin/server.bat on Microsoft Windows systems). By default both the binary and http interfaces are active. If you want to disable one of these change the Server configuration.

Upon startup, the server runs on port 2424 for the binary protocol and 2480 for the http one. If a port is busy the next free one will be used. The default range is 2424-2430 (binary) and 2480-2490 (http). These default ranges can be changed in in Server configuration.

Stop the server

To stop a running server, press CTRL+C in the open shell that runs the Server instance or soft kill the process to be sure that the opened databases close softly. Soft killing on Windows can be done by closing the window. On Unix-like systems, a simple kill is enough (Do not use kill -9 unless you want to force a hard shutdown).

Dump the server status

In order to display the internal status of an OrientDB server, you can send an interrupt to the process. In Unix based OS you can do that by executing `kill -5 <orientdb-server-pid>` . To know more about this topic, please visit the Server Status page.

Connect to the server

By Console

The OrientDB distribution provides the Orient Console tool as a console Java application that uses the binary protocol to work with the database.

By OrientDB Studio

Starting from the release 0.9.13 Orient comes with the OrientDB Studio application, a client-side web app that uses the HTTP protocol to work with the database.

By your application

Consider the native APIs if you use Java. For all the other languages you can use the HTTP RESTful protocol.

Distributed servers

To setup a distributed configuration look at: Distributed-Architecture.

Change the Server's database directory

By default OrientDB server manages the database under the directory "$ORIENTDB_HOME/databases" where $ORIENTDB_HOME is the OrientDB installation directory. By setting the configuration parameter `"server.database.path"` in server orientdb-server-config.xml you can specify a custom path. Example:

```
<orient-server>
  ...
  <properties>
    <entry value="C:/temp/databases" name="server.database.path" />
  </properties>
</orient-server>
```

Configuration

Plugins

Plug-ins (old name "Handler") are the way the OrientDB Server can be extended.

To write your own plug-in read below Extend the server.

Available plugins:

- Automatic-Backup
- EMail Plugin
- JMX Plugin
- Distributed-Server-Manager
- Server-side script interpreter
- Write your own

Protocols

Contains the list of protocols used by the listeners section. The protocols supported today are:

- **binary**: the Raw binary protocol used by OrientDB clients and console application.
- **http**: the HTTP RESTful protocol used by OrientDB Studio and direct raw access from any language and browsers.

Listeners

You can configure multiple listeners by adding items under the `<listeners>` tag and selecting the ip-address and TCP/IP port to bind. The protocol used must be listed in the protocols section. Listeners can be configured with single port or port range. If a range of ports is specified, then it will try to acquire the first port available. If no such port is available, then an error is thrown. By default the Server configuration activates connections from both the protocols:

- **binary**: by default the binary connections are listened to the port range 2424-2430.
- **http**: by default the HTTP connections are listened to the port range 2480-2490.

Storages

Contains the list of the static configured storages. When the server starts for each storages static configured storage enlisted check if exists. If exists opens it, otherwise creates it transparently.

By convention all the storages contained in the $ORIENT_HOME/databases are visible from the OrientDB Server instance without the need of configure them. So configure storages if:

- are located outside the default folder. You can use any environment variable in the path such the ORIENT_HOME that points to the Orient installation path if defined otherwise to the root directory where the Orient Server starts.
- want to create/open automatically a database when the server start ups

By default the **"temp"** database is always configured as in-memory storage useful to store volatile information.

Example of configuration:

```
<storage name="mydb" path="local:C:/temp/databases/mydb"
        userName="admin" userPassword="admin"
        loaded-at-startup="true" />
```

To create a new database use the CREATE DATABASE console command or create it dinamically using the Java-API.

Users

Starting from v.0.9.15 OrientDB supports per-server users in order to protect sensible operations to the users. In facts the creation of a new database is a server operation as much as the retrieving of server statistics.

Automatic password generation

When an OrientDB server starts for the first time, a new user called "root" will be generated and saved in the server configuration. This avoid security problems when, very often, the passwords remain the default ones.

Resources

User based authentication checks if the logged user has the permission to access to the requested resource. "*" means access to all the resource. This is the typical setting for the user "root". Multiple resources must be separated by comma.

Example to let to the "root" user to access to all the server commands:

```
<user name="root" resources="*" password="095F17F6488FF5416ED24E"/>
```

Example to let to the "guest" user to access only to the "info-server" command:

```
<user name="guest" resources="info-server" password="3489438DKJDK4343UDH76"/>
```

Supported resources are:

- `info-server` , to obtain statistics about the server
- `database.create` , to create a new database
- `database.exists` , to check if a database exists
- `database.delete` , to delete an existent database
- `database.share` , to share a database to another OrientDB Server node
- `database.passthrough` , to access to the hosted databases without database's authentication
- `server.config.get` , to retrieve a configuration setting value
- `server.config.set` , to set a configuration setting value

Create new user with some privileges

To configure a new user open the **config/orientdb-server-config.xml** file and add a new XML tag under the tag `<users>` :

```
<users>
    <user name="MyUser" password="MyPassword" resources="database.exists"/>
</users>
```

Extend the server

To extend the server's features look at Extends the server.

Debug the server

To debug the server configure your IDE to execute the class OServerMain:

```
com.orientechnologies.orient.server.OServerMain
```

Passing these parameters:

```
-server
-Dorientdb.config.file=config/orientdb-server-config.xml
-Dorientdb.www.path=src/site
-DORIENTDB_HOME=url/local/orientdb/releases/orientdb-1.2.0-SNAPSHOT
-Djava.util.logging.config.file=config/orientdb-server-log.properties
-Dcache.level1.enabled=false
-Dprofiler.enabled=true
```

Changing the ORIENTDB_HOME according to your path.

Server Status

The server status can be obtained by executing a `kill -5 <server-pid>`. The status will be written to the standard output, by default the console output of the server. The dump contains the following information:

- Current settings
- Stack Trace of all the threads
- Replication latency (if running in HA mode)
- Replication messages statistics (if running in HA mode)
- Distributed resources locked (if running in HA mode)
- Record locks per database (if running in HA mode)

Below an example of a dump.

```
2017-08-21 16:51:25:060 WARNI Received signal: SIGTRAP [OSignalHandler]
OrientDB 2.2.27-SNAPSHOT (build b3c2429a7f0992cd650c61917a171134fa0c6794) configuration dump:
- ENVIRONMENT
  + environment.dumpCfgAtStartup = false
  + environment.concurrent = true
  + environment.lockManager.concurrency.level = 64
  + environment.allowJVMShutdown = true
...

THREAD DUMP
"SIGTRAP handler" id=226
   java.lang.Thread.State: RUNNABLE
        at sun.management.ThreadImpl.getThreadInfo1(Native Method)
        at sun.management.ThreadImpl.getThreadInfo(ThreadImpl.java:174)
        at com.orientechnologies.common.profiler.OAbstractProfiler.threadDump(OAbstractProfiler.java:432)
        at com.orientechnologies.orient.core.OSignalHandler.handle(OSignalHandler.java:76)
        at sun.misc.Signal$1.run(Signal.java:212)
        at java.lang.Thread.run(Thread.java:745)

"Thread-176" id=224
   java.lang.Thread.State: WAITING
        at sun.misc.Unsafe.park(Native Method)
        at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
        at java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039)
        at java.util.concurrent.ArrayBlockingQueue.take(ArrayBlockingQueue.java:403)
        at com.orientechnologies.agent.event.OEventController.run(OEventController.java:214)
...

REPLICATION LATENCY AVERAGE (in milliseconds)
+-------+-----+------+
|Servers|node1|node2*|
+-------+-----+------+
|node1  |     |      |
|node2* |47.32|      |
+-------+-----+------+

REPLICATION MESSAGE COUNTERS (servers: source on the row and destination on the column)
+-------+-----+------+-----+
|Servers|node1|node2*|TOTAL|
+-------+-----+------+-----+
|node1  |     |      |    0|
|node2* |   63|      |   63|
+-------+-----+------+-----+
|TOTAL  |   63|  null|   63|
+-------+-----+------+-----+

REPLICATION MESSAGE CURRENT NODE STATS
+-------+----------+--------+---------+---------------+-----+
|Servers|upd_db_cfg|exc_lock|deploy_db|deploy_delta_db|TOTAL|
+-------+----------+--------+---------+---------------+-----+
|node1  |          |        |         |               |    0|
|node2* |        17|      30|        1|             15|   63|
+-------+----------+--------+---------+---------------+-----+
|TOTAL  |        17|      30|        1|             15|   63|
+-------+----------+--------+---------+---------------+-----+

HA RESOURCE LOCKS FOR SERVER 'node2'

DATABASE 'foo_14' ON SERVER 'node2'
- HA RECORD LOCKS FOR DATABASE 'foo_14'
- MESSAGES IN QUEUES (8 WORKERS):
```

Embed the Server

Embedding an OrientDB Server inside a Java application has several advantages and interesting features:

- Java application that runs embedded with the server can bypass the remote connection and use the database directly with local mode. local and remote connections against the same database can work in concurrency: OrientDB will synchronize the access.
- You can use the Console to control it
- You can use the OrientDB Studio
- You can replicate the database across distributed standalone or embedded servers

To embed an OrientDB Server inside a Java application you have to create the `OServer` object and use a valid configuration for it.

Requirements

In order to embed the server you need to include the following jar files in the classpath:

- `orientdb-server-**.jar`

Starting from version 2.2, please set the `MaxDirectMemorySize` parameter. Setting this parameter is required. You can set it to a very high value, e.g. 512g (JVM setting):

```
-XX:MaxDirectMemorySize=512g
```

Setting `MaxDirectMemorySize` to a very high value should not concern you as it does not mean that OrientDB will consume all 512GB of memory. The size of direct memory consumed by OrientDB is limited by the size of the disk cache (variable `storage.diskCache.bufferSize`).

When you start the Server using the provided `server.sh` or `server.bat` scripts, `MaxDirectMemorySize` is set already by those scripts. But when you embed the Server it is required that you set `MaxDirectMemorySize` manually.

Note: if you are using a pom file, you may set `MaxDirectMemorySize` inside your pom in the following way:

```
<properties>
   <argLine>-XX:MaxDirectMemorySize=512g</argLine>
</properties>
```

If you start an embedded Server without setting this variable you will get a WARNING message similar to the following:

```
MaxDirectMemorySize JVM option is not set or has invalid value, that may cause out of memory errors
```

Include the commands you need

Even if most of the HTTP commands are auto registered assure to have all the commands you need. For example the static content must be registered. This is fundamental if you want to use OrientDB as Web Server providing static content like the Studio app:

```
<listener protocol="http" port-range="2480-2490" ip-address="0.0.0.0">
  <commands>
    <command implementation="com.orientechnologies.orient.server.network.protocol.http.command.get.OServerCommandGetStaticCont
ent" pattern="GET|www GET|studio/ GET| GET|*.htm GET|*.html GET|*.xml GET|*.jpeg GET|*.jpg GET|*.png GET|*.gif GET|*.js GET|*.
css GET|*.swf GET|*.ico GET|*.txt GET|*.otf GET|*.pjs GET|*.svg">
      <parameters>
        <entry value="Cache-Control: no-cache, no-store, max-age=0, must-revalidate\r\nPragma: no-cache" name="http.cache:*.ht
m *.html"/>
        <entry value="Cache-Control: max-age=120" name="http.cache:default"/>
      </parameters>
    </command>
  </commands>
</listener>
```

Use an embedded configuration

```
import com.orientechnologies.orient.server.OServerMain;

public class OrientDBEmbeddable {

 public static void main(String[] args) throws Exception {
  OServer server = OServerMain.create();
  server.startup(
   "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"yes\"?>"
   + "<orient-server>"
   + "<network>"
   + "<protocols>"
   + "<protocol name=\"binary\" implementation=\"com.orientechnologies.orient.server.network.protocol.binary.ONetworkProtocolB
inary\"/>"
   + "<protocol name=\"http\" implementation=\"com.orientechnologies.orient.server.network.protocol.http.ONetworkProtocolHttpD
b\"/>"
   + "</protocols>"
   + "<listeners>"
   + "<listener ip-address=\"0.0.0.0\" port-range=\"2424-2430\" protocol=\"binary\"/>"
   + "<listener ip-address=\"0.0.0.0\" port-range=\"2480-2490\" protocol=\"http\"/>"
   + "</listeners>"
   + "</network>"
   + "<users>"
   + "<user name=\"root\" password=\"ThisIsA_TEST\" resources=\"*\"/>"
   + "</users>"
   + "<properties>"
   + "<entry name=\"orientdb.www.path\" value=\"C:/work/dev/orientechnologies/orientdb/releases/1.0rc1-SNAPSHOT/www/\"/>"
   + "<entry name=\"orientdb.config.file\" value=\"C:/work/dev/orientechnologies/orientdb/releases/1.0rc1-SNAPSHOT/config/orie
ntdb-server-config.xml\"/>"
   + "<entry name=\"server.cache.staticResources\" value=\"false\"/>"
   + "<entry name=\"log.console.level\" value=\"info\"/>"
   + "<entry name=\"log.file.level\" value=\"fine\"/>"
   //The following is required to eliminate an error or warning "Error on resolving property: ORIENTDB_HOME"
   + "<entry name=\"plugin.dynamic\" value=\"false\"/>"
   + "</properties>" + "</orient-server>");
  server.activate();
  }
}
```

Once the embedded server is running, clients can connect using the remote connection method. For example in the console, you can connect with:

```
connect remote:localhost:{port}/{db} {user} {password}
where:
  port     : the port that the binary server listens on
             (first free port from 2424-2430 according to the configuration above)
  db       : the database name to connect to (defaults to "db" and can be set using <entry name="server.database.path" value="
db"/> in the configuration
  user     : the user to connect with (this is NOT the same as root user in the configuration)
  password : the user to connect with (this is NOT the same as root password in the configuration)
```

Use custom file for configuration

Use a regular `File` :

```
public class OrientDBEmbeddable {
  public static void main(String[] args) throws Exception {
      OServer server = OServerMain.create();
      server.startup(new File("/usr/local/temp/db.config"));
      server.activate();
  }
}
```

Use a stream for configuration

Use an `InputStream` from the class loader:

```
public class OrientDBEmbeddable {
  public static void main(String[] args) throws Exception {
      OServer server = OServerMain.create();
      server.startup(getClass().getResourceAsStream("db.config"));
      server.activate();
  }
}
```

Use a OServerConfiguration object for configuration

Or an `InputStream` from the class loader:

```
public class OrientDBEmbeddable {
  public static void main(String[] args) throws Exception {
      OServer server = OServerMain.create();
      OServerConfiguration cfg = new OServerConfiguration();
      // FILL THE OServerConfiguration OBJECT
      server.startup(cfg);
      server.activate();
  }
}
```

Shutdown

OrientDB Server creates some threads internally as non-daemon, so they run even if the main application exits. Use the `OServer.shutdown()` method to shutdown the server in soft way:

```
import com.orientechnologies.orient.server.OServerMain;

public class OrientDBEmbeddable {

  public static void main(String[] args) throws Exception {
    OServer server = OServerMain.create();
    server.startup(new File("/usr/local/temp/db.config"));
    server.activate();
    ...
    server.shutdown();
  }
}
```

Setting ORIENTDB_HOME

Some functionality wil not work properly if the system property 'ORIENTDB_HOME' is not set. You can set it programmatically like this:

```
import com.orientechnologies.orient.server.OServerMain;

public class OrientDBEmbeddable {

  public static void main(String[] args) throws Exception {
    String orientdbHome = new File("").getAbsolutePath(); //Set OrientDB home to current directory
    System.setProperty("ORIENTDB_HOME", orientdbHome);

    OServer server = OServerMain.create();
    server.startup(cfg);
    server.activate();
  }
}
```

Web Server

❗	Even thought OrientDB Server is a regular Web Server, it is not recommended to expose it directly on the Internet or public networks. We suggest to always hide OrientDB server in a private network.

Global settings can be set at JVM startup (`java ... -D<setting>="<value>"`) or in `orientdb-server-config.xml` file under "properties" XML tag.

Avoid exposing OrientDB Server to a public network

By default, OrientDB listens to all the network interfaces (`0.0.0.0`). It's strongly suggested to do not open OrientDB server on public networks. To force OrientDB to bind only one network, please edit the file `config/orientdb-server-config.xml` file and replace `"0.0.0.0` with `127.0.0.1` if you want only locally clients can access to the server, or any other valid IP you want to publish OrientDB. This is the default configuration:

```
<listeners>
  <listener protocol="binary" socket="default" port-range="2424-2430" ip-address="0.0.0.0"/>
  <listener protocol="http" socket="default" port-range="2480-2490" ip-address="0.0.0.0">
</listeners>
```

To bind OrientDB server only to the local server, change it into:

```
<listeners>
  <listener protocol="binary" socket="default" port-range="2424-2430" ip-address="127.0.0.1"/>
  <listener protocol="http" socket="default" port-range="2480-2490" ip-address="127.0.0.1">
</listeners>
```

Maximum content length

OrientDB by default allow request content of maximum 1MB. To change this limitation set the global configuration `network.http.maxLength` to the needed value.

Charset

OrientDB uses UTF-8 as default charset. To change it set the global configuration `network.http.charset` .

JSONP

JSONP is supported by OrientDB Web Server, but disabled by default. To enable it set the coniguration `network.http.jsonp=true`

This is a global setting, so you can set it at JVM startup (`java ... -Dnetwork.http.jsonp=true`) or by setting it as property in `orientdb-server-config.xml` file under "properties" XML tag.

Cross site

Cross site requests are disabled by default.

To enable it, set a couple of additional headers in `orientdb-server-config.xml` under the HTTP listener XML tag:

```
<listener protocol="http" ip-address="0.0.0.0" port-range="2480-2490" socket="default">
  <parameters>
    <parameter name="network.http.additionalResponseHeaders" value="Access-Control-Allow-Origin: *;Access-Control-Allow-Creden
tials: true" />
  </parameters>
</listener>
```

This setting is also global, so you can set it at JVM startup (`java ... -Dnetwork.http.additionalResponseHeaders="Access-Control-Allow-Origin: *;Access-Control-Allow-Credentials: true"`) or by setting it as property in `orientdb-server-config.xml` file under "properties" XML tag.

Clickjacking

Look also: Clickjacking on WikiPedia and Clickjacking on OWASP

OrientDB allows to disable Clickjacking by setting the additional header `X-FRAME-OPTIONS` to `DENY` in all the HTTP response.

To enable it, set a couple of additional headers in `orientdb-server-config.xml` under the HTTP listener XML tag:

```
<listener protocol="http" ip-address="0.0.0.0" port-range="2480-2490" socket="default">
  <parameters>
    <parameter name="network.http.additionalResponseHeaders" value="X-FRAME-OPTIONS: DENY" />
  </parameters>
</listener>
```

This setting is also global, so you can set it at JVM startup (`java ... -Dnetwork.http.additionalResponseHeaders="X-FRAME-OPTIONS: DENY"`) or by setting it as property in `orientdb-server-config.xml` file under "properties" XML tag.

OrientDB System Database

Introduced in 2.2, OrientDB now uses a "system database" to provide additional capabilities.

The system database, currently named *OSystem*, is created when the OrientDB server starts, if the database does not exist.

Features

Here's a list of some of the features that the system database may support:

- A new class of user called the *system user*
- A centralized location for configuration files
- Logging of per-database and global auditing events
- Recording performance metrics about the server and its databases

System Users

A third type of user now exists, called a *system user*. A *system user* is similar in concept to a *server user* but resides in the system database as an *OUser* record. See System Users.

Accessing The System Database via Studio

By default, the OrientDB system database will not be displayed in the *Studio* drop-down list of databases. To enable this, add a "server.listDatabases.system" resource to the "guest" server user.

Here's an example from *orientdb-server-config.xml*:

```
<user resources="connect,server.listDatabases,server.listDatabases.system" password="*****" name="guest"/>
```

Schema

Currently, the system database has no specialized class schema, but this will changes as more features are added that utilize the private database.

System Users

Traditionally, in OrientDB, there have been two types of users: a *server user* and a *database user*.

A *server user* is specified in the `<users>` section of the *orientdb-server-config.xml* file or as a user object in the ODefaultPasswordAuthenticator section of *security.json*. A *server user* typically has a name, a password, and a list of permitted resources to server-related activities that are not specific to a single database. An example is "server.info" which permits returning such things as the currently active client connections, the databases on the server, and a list of global properties.

A *database user* resides in each database as an *OUser* record, and its associated roles and permissions apply only to that database.

A third type of user now exists, called a *system user*. A *system user* is similar in concept to a *server user* but resides in the system database as an *OUser* record. Like a *database user*, a *system user* is assigned roles that are comprised of resources and permissions. See Database Security. What's unique about a *system user* is that its roles may be specified per-database or against the server, depending on the chosen resource.

Authenticator

To support the *system users* a new authenticator has been added, called *OSystemUserAuthenticator*. It is enabled, by default, in the *security.json* configuration file, and has been added in the chain of authenticators. See New Security Features.

Users and Roles

To create new *System users*, they are added to the system database, and the procedure is identical to adding a user as outlined in Database Security.

The main difference between a *system user* and a *database user* comes with roles. A role can be created that allows a *system user* to have access to a specific database, multiple databases, or all databases while still supporting the standard database resources (such as *database*, *database.class*, *database.cluster*, *database.command*).

The way this is achieved is by adding a property to the ORole record, called *dbFilter*. As an example, say we have an OrientDB class called *Car*, and we want to create a role to allow a user to modify all the *Car* records. For a regular *database user*, we'd do something like this:

```
INSERT INTO ORole SET name = "carAdmin", mode = 0
UPDATE ORole PUT rules = "database.class.Car", 15 WHERE name = "carAdmin"
```

These commands will create a new role called *carAdmin* with a mode of "deny all but", and will add a rule to the role granting all permissions (create, read, update, delete) to modify the *Car* class.

To make this work for a *system user*, we make a small change by adding a *dbFilter* property to the role. Here's an example allowing a role to access the *MyCarDB* database:

```
UPDATE ORole SET dbFilter = ["MyCarDB"] WHERE name = "carAdmin"
```

The *dbFilter* property is a list of strings indicating which databases a role is allowed to access. A wildcard ("*") may be specified to permit access to all databases.

Server Resources

What's unique about a *system user* is that it's essentially a hybrid of a *server user* and a *database user*. A *server user* has access to a finite number of server-related resources (such as *server.info*, *server.listDatabases*, and *db.copy*). A *system user* also has access to the same server resources as a *server user* has, but the resource name is added as a rule of a role.

Here's an example of adding the resource *server.info* to a role called *sysinfo*.

```
UPDATE ORole PUT rules = "server.info", 16 WHERE name = "sysinfo"
```

Notice that a new permission (16 - execute) has been specified for the *server.info* rule. Server resources typically have execute-only permission.

The previously mentioned, database-specific property, *dbFilter*, is not set on the role since *server.info* is specific to the server and not an individual database.

System Users Implementation

The new OrientDB security system installs a specialized *OSecurityShared* class, called *OSecurityExternal*, that supports external authentication of users, meaning that a username can be authenticated outside the realm of a local database. Typically, there is a "chain of authenticators", specified in the *security.json* configuration file, in the "authentication" section, that will be checked to see if the username can be authenticated. The authenticator for the *system users* is called *OSystemUserAuthenticator*.

If authentication is successful, *OSecurityExternal.authenticate()* then calls `Orient.instance().getSecurity().getSystemUser()` with the authenticated username and the name of the local database being opened. (`Orient.instance().getSecurity()` returns an instance of the system security module, *ODefaultServerSecurity*.) If a *system user* is found, `getSystemUser()` returns a derived *OUser* class instance, called *OSystemUser*.

Database Open

When a database is opened, the ODatabaseDocumentTx.open() method calls `final OUser usr = metadata.getSecurity().authenticate(iUserName, iUserPassword);` . If the new security system is enabled, the call to `metadata.getSecurity()` will return an instance of *OSecurityExternal*. The `OSecurityExternal.authenticate()` method calls `Orient.instance().getSecurity().authenticate(iUserName, iUserPassword)` . The method, `Orient.instance().getSecurity()` returns an instance of *ODefaultServerSecurity*, and its `authenticate()` method is where the chain of installed authenticators is called.

OSystemUserAuthenticator

OSystemUserAuthenticator implements a method, `public String authenticate(final String username, final String password)` , from the *OSecurityAuthenticator* interface. This method queries the system database for an *OUser* record that matches the specified username and, if found, validates the provided password. If successful, the username is returned, otherwise null.

OSystemUser

As mentioned previously, when `Orient.instance().getSecurity().getSystemUser()` is called, if a system user is found, an *OSystemUser* instance is returned. *OSystemUser* extends *OUser* and implements a new constructor, `public OSystemUser(final ODocument iSource, final String dbName)` , as well as a newly overrided method, `protected ORole createRole(final ODocument roleDoc)` .

The "database name" (dbName) parameter in *OSystemUser*'s constructor is used to filter which roles in the system database are associated with a user. A list property, called *dbFilter*, can be set on an ORole record to assigned the database name. If *dbName* is null, then only roles without a *dbFilter* property are associated with the *OSystemUser*.

OSystemRole

In conjunction with *OSystemUser* is a new class, called *OSystemRole*, which derives from *ORole* and adds a new public method, `List<String> getDbFilter()` . The *OSystemUser* `createRole()` method creates and returns *OSystemRole* instances.

Multi Tenant

There are at many ways to build multi-tenant applications on top of OrientDB, in this page we are going to analyze pros and cons of three of the most used approaches.

One database per tenant

With this solution, each tenant is a database. The OrientDB server allows to host multiple databases.

Pros:

- Easy to use: to create/drop a new tenant, simply create/drop the database
- Easy to scale up by moving the database on different servers

Cons:

- Hard to create reports and analytics cross tenant, it requires to execute the same query against all the databases

Specific clusters per tenant

With this solution, each tenant is stored in one or more clusters of the same database. For example, the class `Product` could have the following clusters: `Product_ClientA` , `Product_ClientB` , `Product_ClientC` . In this way a query against a specific cluster will be used to retrieve data from one tenant only. Example to retrieve all the products of 2016, ordered by the most recent, only for the tenant "ClientC": `select * from cluster:Product_ClientC where date >= '2016-01-01' order by date desc` .

Instead, a query against the class `Product` will return a cross-tenant result. Example: `select * from Product where date >= '2016-01-01' order by date desc` .

Pros:

- Easy to create reports and analytics cross tenant, because it's just one database
- It's possible to scale up on multiple servers by using sharding

Cons:

- The maximum number of clusters per database is 32,768. To bypass this limitation, use multiple databases
- No security out of the box, it's entirely up to the application to isolate access between tenants

Use Partitioned Graphs

By using the OrientDB's record level security, it's possible to have multiple partitions of graphs accessible by different users. For more information look at Partitioned Graphs.

Pros:

- Easy to create reports and analytics cross tenant, because it's just one database

Cons:

- Performance: record level security has an overhead, specially with queries
- Scales worse than "One database per tenant" solution, because the database will end up to be much bigger
- Sharding is not possible because records of multiple tenants are mixed on the same clusters
- Hard to guarantee a predictable level of service for all the users, because users connect to a tenant impact the other tenants

OrientDB Plugins

The OrientDB Server is a customizable platform to build powerful server component and applications.

Since the OrientDB server contains an integrated Web Server what about creating server side applications without the need to have a J2EE and Servlet container? By extending the server you can benefit of the best performance because you don't have many layers but the database and the application reside on the same JVM without the cost of the network and serialization of requests.

Furthermore you can package your application together with the OrientDB server to distribute just a ZIP file containing the entire Application, Web server and Database.

To customize the OrientDB server you have two powerful tools:

- Handlers
- Custom commands

To debug the server while you develop new feature follow Debug the server.

Handlers (Server Plugins)

Handlers are plug-ins and starts when OrientDB starts.

To create a new handler create the class and register it in the OrientDB server configuration.

Create the Handler class

A Handler must implements the OServerPlugin interface or extends the OServerPluginAbstract abstract class.

Below an example of a handler that print every 5 seconds a message if the "log" parameters has been configured to be "true":

```
package orientdb.test;

public class PrinterHandler extends OServerPluginAbstract {
  private boolean    log = false;

  @Override
  public void config(OServer oServer, OServerParameterConfiguration[] iParams) {
    for (OServerParameterConfiguration p : iParams) {
      if (p.name.equalsIgnoreCase("log"))
        log = true;
    }

    Orient.getTimer().schedule( new TimerTask() {
      @Override
      public void run() {
        if( log )
          System.out.println("It's the PrinterHandler!");
      }
    }, 5000, 5000);
  }

  @Override
  public String getName() {
    return "PrinterHandler";
  }
}
```

Register the handler

Once created, register it to the server configuration in **orientdb-server-config.xml** file:

```
<orient-server>
  <handlers>
    <handler class="orientdb.test.PrinterHandler">
      <parameters>
        <parameter name="log" value="true"/>
      </parameters>
    </handler>
  </handlers>
  ...
```

Note that you can specify arbitrary parameters in form of name and value. Those parameters can be read by the **config()** method. In this example a parameter "log" is read. Look upon to the example of handler to know how to read parameters specified in configuration.

Steps to register a function as a Plugin in OrientDB

In this case we'll create a plugin that only registers one function in OrientDB: **pow** (returns the value of the first argument raised to the power of the second argument). We'll also support Modular exponentiation.

The syntax will be `pow(<base>, <power> [, <mod>])` .

- you should have a directory structure like this

```
.
├─ src
|   └─ main
|       ├─ assembly
|       |   └─ assembly.xml
|       ├─ java
|       |   └─ com
|       |       └─ app
|       |           └─ OPowPlugin.java
|       └─ resources
|           └─ plugin.json
|
└─ pom.xml
```

OPowPlugin.java

```
package com.app;

import com.orientechnologies.common.log.OLogManager;
import com.orientechnologies.orient.core.command.OCommandContext;
import com.orientechnologies.orient.core.db.record.OIdentifiable;
import com.orientechnologies.orient.core.sql.OSQLEngine;
import com.orientechnologies.orient.core.sql.functions.OSQLFunctionAbstract;
import com.orientechnologies.orient.server.OServer;
import com.orientechnologies.orient.server.config.OServerParameterConfiguration;
import com.orientechnologies.orient.server.plugin.OServerPluginAbstract;
import java.util.ArrayList;
import java.util.List;

public class OPowPlugin extends OServerPluginAbstract {

    public OPowPlugin() {
    }

    @Override
    public String getName() {
        return "pow-plugin";
    }

    @Override
    public void startup() {
        super.startup();
        OSQLEngine.getInstance().registerFunction("pow", new OSQLFunctionAbstract("pow", 2, 3) {
            @Override
            public String getSyntax() {
                return "pow(<base>, <power> [, <mod>])";
            }
            @Override
            public Object execute(Object iThis, OIdentifiable iCurrentRecord, Object iCurrentResult, final Object[] iParams, O
CommandContext iContext) {
                if (iParams[0] == null || iParams[1] == null) {
                    return null;
                }
                if (!(iParams[0] instanceof Number) || !(iParams[1] instanceof Number)) {
                    return null;
                }

                final long base = ((Number) iParams[0]).longValue();
                final long power = ((Number) iParams[1]).longValue();

                if (iParams.length == 3) { // modular exponentiation
                    if (iParams[2] == null) {
                        return null;
                    }
                    if (!(iParams[2] instanceof Number)) {
                        return null;
                    }

                    final long mod = ((Number) iParams[2]).longValue();
                    if (power < 0) {
                        OLogManager.instance().warn(this, "negative numbers as exponent are not supported");
                    }
                    return modPow(base, power, mod);
                }

                return power > 0 ? pow(base, power) : 1D / pow(base, -power);
            }
        });
        OLogManager.instance().info(this, "pow function registered");
    }

    private double pow(long base, long power) {
        double r = 1;
        List<Boolean> bits = bits(power);
        for (int i = bits.size() - 1; i >= 0; i--) {
            r *= r;
            if (bits.get(i)) {
                r *= base;
            }
        }

        return r;
    }

    private double modPow(long base, long power, long mod) {
        double r = 1;
        List<Boolean> bits = bits(power);
        for (int i = bits.size() - 1; i >= 0; i--) {
            r = (r * r) % mod;
            if (bits.get(i)) {
                r = (r * base) % mod;
            }
        }

        return r;
    }

    private List<Boolean> bits(long n) {
        List<Boolean> bits = new ArrayList();
        while (n > 0) {
            bits.add(n % 2 == 1);
            n /= 2;
        }

        return bits;
    }

    @Override
    public void config(OServer oServer, OServerParameterConfiguration[] iParams) {

    }

    @Override
    public void shutdown() {
        super.shutdown();
    }
}
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.app</groupId>
    <artifactId>pow-plugin</artifactId>
    <version>2.0.7</version>
    <packaging>jar</packaging>

    <name>pow-plugin</name>

    <properties>
        <orientdb.version>2.0.7</orientdb.version>
    </properties>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>2.4</version>
                <configuration>
                    <descriptors>
                        <descriptor>src/main/assembly/assembly.xml</descriptor>
                    </descriptors>
                </configuration>
                <executions>
                    <execution>
                        <id>make-assembly</id>
                        <!-- this is used for inheritance merges -->
                        <phase>package</phase>
                        <!-- bind to the packaging phase -->
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration></configuration>
                    </execution>
                </executions>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>com.orientechnologies</groupId>
            <artifactId>orientdb-core</artifactId>
            <version>${orientdb.version}</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>com.orientechnologies</groupId>
            <artifactId>orientdb-server</artifactId>
            <version>${orientdb.version}</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>
</project>
```

assembly.xml

```
<assembly>
    <id>dist</id>
    <formats>
        <format>jar</format>
    </formats>
    <includeBaseDirectory>false</includeBaseDirectory>
    <dependencySets>
        <dependencySet>
            <outputDirectory/>
            <unpack>true</unpack>
            <includes>
                <include>${groupId}:${artifactId}</include>
            </includes>
        </dependencySet>
    </dependencySets>
</assembly>
```

plugin.json

```
{
    "name" : "pow-plugin",
    "version" : "2.0.7",
    "javaClass": "com.app.OPowPlugin",
    "parameters" : {},
    "description" : "The Pow Plugin",
    "copyrights" : "No copyrights"
}
```

- Build the project and then:

```
cp target/pow-plugin-2.0.7-dist.jar $ORIENTDB_HOME/plugins/
```

You should see the following in OrientDB server log:

```
INFO  Installing dynamic plugin 'pow-plugin-2.0.7-dist.jar'... [OServerPluginManager]
INFO  pow function registered [OPowPlugin]
```

And now you can:

```
orientdb {db=Pow}> select pow(2,10)

----+------+------
#   |@CLASS|pow
----+------+------
0   |null  |1024.0
----+------+------

orientdb {db=Pow}> select pow(2,10,5)

----+------+----
#   |@CLASS|pow
----+------+----
0   |null  |4.0
----+------+----
```

This small project is available here.

Creating a distributed change manager

As more complete example let's create a distributed record manager by installing hooks to all the server's databases and push these changes to the remote client caches.

```
public class DistributedRecordHook extends OServerHandlerAbstract implements ORecordHook {
  private boolean log = false;

  @Override
  public void config(OServer oServer, OServerParameterConfiguration[] iParams) {
    for (OServerParameterConfiguration p : iParams) {
      if (p.name.equalsIgnoreCase("log"))
        log = true;
    }
  }

  @Override
  public void onAfterClientRequest(final OClientConnection iConnection, final byte iRequestType) {
    if (iRequestType == OChannelBinaryProtocol.REQUEST_DB_OPEN)
      iConnection.database.registerHook(this);
    else if (iRequestType == OChannelBinaryProtocol.REQUEST_DB_CLOSE)
      iConnection.database.unregisterHook(this);
  }

  @Override
  public boolean onTrigger(TYPE iType, ORecord<?> iRecord) {
    try {
      if (log)
        System.out.println("Broadcasting record: " + iRecord + "...");

      OClientConnectionManager.instance().broadcastRecord2Clients((ORecordInternal<?>) iRecord, null);
    } catch (Exception e) {
      e.printStackTrace();
    }
    return false;
  }

  @Override
  public String getName() {
    return "DistributedRecordHook";
  }
}
```

Custom commands

Custom commands are useful when you want to add behavior or business logic at the server side.

A Server command is a class that implements the OServerCommand interface or extends one of the following abstract classes:

- OServerCommandAuthenticatedDbAbstract if the command requires an authentication at the database
- OServerCommandAuthenticatedServerAbstract if the command requires an authentication at the server

The Hello World Web

To learn how to create a custom command, let's begin with a command that just returns "Hello world!".

OrientDB follows the convention that the command name is:

`OServerCommand<method><name>` Where:

- **method** is the HTTP method and can be: GET, POST, PUT, DELETE
- **name** is the command name

In our case the class name will be "OServerCommandGetHello". We want that the use must be authenticated against the database to execute it as any user.

Furthermore we'd like to receive via configuration if we must display the text in Italic or not, so for this purpose we'll declare a parameter named "italic" of type boolean (true or false).

```
package org.example;

public class OServerCommandGetHello extends OServerCommandAuthenticatedDbAbstract {
  // DECLARE THE PARAMETERS
  private boolean italic = false;

  public OServerCommandGetHello(final OServerCommandConfiguration iConfiguration) {
    // PARSE PARAMETERS ON STARTUP
    for (OServerEntryConfiguration par : iConfiguration.parameters) {
      if (par.name.equals("italic")) {
        italic = Boolean.parseBoolean(par.value);
      }
    }
  }

  @Override
  public boolean execute(final OHttpRequest iRequest, OHttpResponse iResponse) throws Exception {
    // CHECK THE SYNTAX. 3 IS THE NUMBER OF MANDATORY PARAMETERS
    String[] urlParts = checkSyntax(iRequest.url, 3, "Syntax error: hello/<database>/<name>");

    // TELLS TO THE SERVER WHAT I'M DOING (IT'S FOR THE PROFILER)
    iRequest.data.commandInfo = "Salutation";
    iRequest.data.commandDetail = "This is just a test";

    // GET THE PARAMETERS
    String name = urlParts[2];

    // CREATE THE RESULT
    String result = "Hello " + name;
    if (italic) {
      result = "<i>" + result + "</i>";
    }

    // SEND BACK THE RESPONSE AS TEXT
    iResponse.send(OHttpUtils.STATUS_OK_CODE, "OK", null, OHttpUtils.CONTENT_TEXT_PLAIN, result);

    // RETURN ALWAYS FALSE, UNLESS YOU WANT TO EXECUTE COMMANDS IN CHAIN
    return false;
  }

  @Override
  public String[] getNames() {
    return new String[]{"GET|hello/* POST|hello/*"};
  }
}
```

Once created the command you need to register them through the **orientdb-server-config.xml** file. Put a new tag `<command>` under the tag `commands` of `<listener>` with attribute `protocol="http"` :

```
  ...
  <listener protocol="http" port-range="2480-2490" ip-address="0.0.0.0">
    <commands>
      <command implementation="org.example.OServerCommandGetHello" pattern="GET|hello/*">
        <parameters>
          <entry name="italic" value="true"/>
        </parameters>
      </command>
    </commands>
  </listener>
```

Where:

- **implementation** is the full class name of the command
- **pattern** is how the command is called in the format: `<HTTP-method>|<name>` . In this case it's executed on HTTP GET with the URL: `/<name>`
- **parameters** specify parameters to pass to the command on startup
- **entry** is the parameter pair name/value

To test it open a browser at this address:

```
http://localhost/hello/demo/Luca
```

You will see:

```
Hello Luca
```

Complete example

Below a more complex example taken by official distribution. It is the command that executes queries via HTTP. Note how to get a database instance to execute operation against the database:

```
public class OServerCommandGetQuery extends OServerCommandAuthenticatedDbAbstract {
  private static final String[] NAMES = { "GET|query/*" };

  @Override
  public boolean execute(OHttpRequest iRequest, OHttpResponse iResponse) throws Exception {
    String[] urlParts = checkSyntax(
        iRequest.url,
        4,
        "Syntax error: query/<database>/sql/<query-text>[/<limit>][/<fetchPlan>].<br/>Limit is optional and is setted to 20 by
 default. Set expressely to 0 to have no limits.");

    int limit = urlParts.length > 4 ? Integer.parseInt(urlParts[4]) : 20;
    String fetchPlan = urlParts.length > 5 ? urlParts[5] : null;
    String text = urlParts[3];

    iRequest.data.commandInfo = "Query";
    iRequest.data.commandDetail = text;

    ODatabaseDocumentTx db = null;

    List<OIdentifiable> response;

    try {
      db = getProfiledDatabaseInstance(iRequest);
      response = (List<OIdentifiable>) db.command(new OSQLSynchQuery<OIdentifiable>(text, limit).setFetchPlan(fetchPlan)).exec
ute();

    } finally {
      if (db != null) {
        db.close();
      }
    }

    iResponse.writeRecords(response, fetchPlan);
    return false;
  }

  @Override
  public String[] getNames() {
    return NAMES;
  }
}
```

Include JARS in the classpath

If your extensions need additional libraries put the additional jar files under the `/lib` folder of the server installation.

Debug the server

To debug your plugin you can start your server in debug mode.

Parameter	Value
Main class	`com.orientechnologies.orient.server.OServerMain`
JVM parameters	`-server -DORIENTDB_HOME=/opt/orientdb -Dorientdb.www.path=src/site -Djava.util.logging.config.file=${ORIENTDB_HOME}/config/orientdb-server-log.properties -Dorientdb.config.file=${ORIENTDB_HOME}/config/orientdb-server-config.xml`

Automatic Backup Server Plugin

Using this server plugin, OrientDB executes regular backups on the databases. It implements the Java class:

```
com.orientechnologies.orient.server.handler.OAutomaticBackup
```

Plugin Configuration

Beginning with version 2.2, OrientDB manages the server plugin configuration from a separate JSON. You can update this file manually or through OrientDB Studio.

To enable automatic backups, use the following `<handler>` section in the `config/orientdb-server-config.xml` configuration file:

```
<!-- AUTOMATIC BACKUP, TO TURN ON SET THE 'ENABLED' PARAMETER TO 'true' -->
<handler class="com.orientechnologies.orient.server.handler.OAutomaticBackup">
    <parameters>
        <parameter name="enabled" value="false"/>
        <!-- LOCATION OF JSON CONFIGURATION FILE -->
        <parameter name="config" value="${ORIENTDB_HOME}/config/automatic-backup.json"/>
    </parameters>
</handler>
```

This section tells the OrientDB server to read the file at `$ORIENTDB_HOME/config/automatic-backup.json` for the automatic backup configuration.

```
{
  "enabled": true,
  "mode": "FULL_BACKUP",
  "exportOptions": "",
  "delay": "4h",
  "firstTime": "23:00:00",
  "targetDirectory": "backup",
  "targetFileName": "${DBNAME}-${DATE:yyyyMMddHHmmss}.zip",
  "compressionLevel": 9,
  "bufferSize": 1048576
}
```

- **`"enabled"`** Defines whether it uses automatic backups. The supported values are:
  - *`true`* Enables automatic backups.
  - *`false`* Disables automatic backups. This is the default setting.
- **`"mode"`** Defines the backup mode. The supported values are:
  - *`"FULL_BACKUP"`* Executes a full backup. Prior to version 2.2, this was the only mode available. This operation blocks the database.
  - *`"INCREMENTAL_BACKUP"`* Executes an incremental backup. This is available only in the Enterprise Edition. It uses one directory per database. This operation doesn't block the database.
  - *`"EXPORT"`* Executes an database export, using gziped JSON format. This operation is not blocking.
- **`"exportOptions"`** Defines export options to use with that mode. This feature was introduced in version 2.2.
- **`"delay"`** Defines the delay time for each backup. Supports the following suffixes:
  - *`ms`* Delay measured in milliseconds.
  - *`s`* Delay measured in seconds.
  - *`m`* Delay measured in minutes.
  - *`h`* Delay measured in hours.
  - *`d`* Delay measured in days.
- **`"firstTime"`** Defines when to initiate the first backup in the schedule. It uses the format of `HH:mm:ss` in the GMT time zone, on the current day.
- **`"targetDirectory"`** Defines the target directory to write backups. By default, it is set to the `backup/` directory.
- **`"targetFileName"`** Defines the target filename. This parameter supports the use of the following variables, (that is, `"${DBNAME}-backup.zip"` produces `mydatabase-backup.zip`):

  - `${DBNAME}` Renders the database name.
  - `${DATE}` Renders the current date, using the Java DateTime syntax format.
- `"dbInclude"` Defines in a list the databases to include in the automatic backups. If empty, it backs up all databases.
- `"dbExclude"` Defines in a list the databases to exclude from the automatic backups.
- `"bufferSize"` Defines the in-memory buffer sizes to use in compression. By default, it is set to `1MB`. Larger buffers mean faster backups, but they in turn consume more RAM.
- `"compressionLevel"` Defines the compression level for the resulting ZIP file. By default it is set to the maximum level of `9`. Set it to a lower value if you find that the backup takes too much time.

Legacy Plugin Configuration

In versions prior to 2.2, the only option in configuring automatic backups is to use the `config/orientdb-server-config.xml` configuration file. Beginning with version 2.2 you can manage automatic backup configuration through a separate JSON file or use the legacy approach.

The example below configures automatic backups/exports on the database as a Server Plugin.

```
<!-- AUTOMATIC BACKUP, TO TURN ON SET THE 'ENABLED' PARAMETER TO 'true' -->
<handler class="com.orientechnologies.orient.server.handler.OAutomaticBackup">
  <parameters>
    <parameter name="enabled" value="false" />
     <!-- CAN BE: FULL_BACKUP, INCREMENTAL_BACKUP, EXPORT -->
     <parameter name="mode" value="FULL_BACKUP"/>
     <!-- OPTION FOR EXPORT -->
     <parameter name="exportOptions" value=""/>
    <parameter name="delay" value="4h" />
    <parameter name="target.directory" value="backup" />
    <!-- ${DBNAME} AND ${DATE:} VARIABLES ARE SUPPORTED -->
    <parameter name="target.fileName" value="${DBNAME}-${DATE:yyyyMMddHHmmss}.zip" />
    <!-- DEFAULT: NO ONE, THAT MEANS ALL DATABASES.
         USE COMMA TO SEPARATE MULTIPLE DATABASE NAMES -->
    <parameter name="db.include" value="" />
    <!-- DEFAULT: NO ONE, THAT MEANS ALL DATABASES.
         USE COMMA TO SEPARATE MULTIPLE DATABASE NAMES -->
    <parameter name="db.exclude" value="" />
    <parameter name="compressionLevel" value="9"/>
    <parameter name="bufferSize" value="1048576"/>
  </parameters>
</handler>
```

- `enabled` Defines whether it uses automatic backups. Supported values are:
  - `true` Enables automatic backups.
  - `false` Disables automatic backups. This is the default setting.
- `mode/>` Defines the backup mode. Supported values are:
  - `FULL_BACKUP` Executes a full backup. For versions prior to 2.2, this is the only option available. This operation blocks the database.
  - `INCREMENTAL_BACKUP` Executes an incremental backup. Uses one directory per database. This operation doesn't block the database.
  - * `EXPORT` Executes an export of the database in gzipped JSON format, instead of a backup. This operation doesn't block the database.
- `exportOptions` Defines export options to use with that mode. This feature was introduced in version 2.2.
- `delay` Defines the delay time. Supports the following suffixes:
  - `ms` Delay measured in milliseconds.
  - `s` Delay measured in seconds.
  - `m` Delay measured in minutes.
  - `h` Delay measured in hours.
  - `d` Delay measured in days.
- `firstTime` Defines when to initiate the first backup in the schedule. It uses the format of `HH:mm:ss` in the GMT time zone, on the current day.
- `target.directory` Defines the target directory to write backups. By default, it is set to the `backup/` directory.
- `target.fileName` Defines the target file name. The parameter supports the use of the following variables, (that is, `<parameter`

`name="target.filename" value="${DBNAME}-backup.zip"/>` produces a `mydatabase-backup.zip` file).
  - *`${DBNAME}`* Renders the database name.
  - *`${DATE}`* Renders the current date, using the Java DateTime syntax format.
- **`db.include`** Defines in a list the databases to include in the automatic backups. If left empty, it backs up all databases.
- **`db.exclude`** Defines in a list the databases to exclude from automatic backups.
- **`bufferSize`** Defines the in-memory buffer sizes to use in compression. By default it is set to `1MB`. Larger buffers mean faster backups, but use more RAM. This feature was introduced in version 1.7.
- **`compressionLevel`** Defines the compression level for the resulting ZIP file. By default, it is set to the maximum level of `9`. Set it to a lower value if you find that the backup takes too much time.

SysLog Plugin

Java class implementation:

```
com.orientechnologies.security.syslog.ODefaultSyslog
```

Available since: **v. 2.2.0**.

Introduction

Allows sending event logs to the Operating System's SYSLOG daemon.

Configuration

This plugin is configured as a Server plugin. The plugin can be easily configured by changing parameters in the `orientdb-server-config.xml` file.:

Name	Description	Type	Example	Since
enabled	true to turn on, false (default) is turned off	boolean	true	2.2.0
debug	Enables debug mode	boolean	false	2.2.0
hostname	The hostname of the syslog daemon	string	localhost	2.2.0
port	The UDP port of the syslog daemon	integer	514	2.2.0
appName	The name of the application submitting the events to SysLog	string	OrientDB	2.2.0

Default configuration in `orientdb-server-config.xml`. Example:

```
<!-- SYS LOG CONNECTOR, TO TURN ON SET THE 'ENABLED' PARAMETER TO 'true' -->
<handler class="com.orientechnologies.security.syslog.ODefaultSyslog">
    <parameters>
        <parameter name="enabled" value="true"/>
        <parameter name="debug" value="false"/>
        <parameter name="hostname" value="localhost"/>
        <parameter name="port" value="514"/>
        <parameter name="appName" value="OrientDB"/>
    </parameters>
</handler>
```

Usage

Look at Security Config.

Mail Plugin

Java class implementation:

```
com.orientechnologies.orient.server.plugin.mail.OMailPlugin
```

Available since: **v. 1.2.0**.

Introduction

Allows to send (and in future read) emails.

Configuration

This plugin is configured as a Server handler. The plugin can be configured in easy way by changing parameters:

Name	Description	Type	Example	Since
enabled	true to turn on, false (default) is turned off	boolean	true	1.2.0
`profile.<name>.mail.smtp.host`	The SMTP host name or ip-address	string	smtp.gmail.com	1.2.0
`profile.<name>.mail.smtp.port`	The SMTP port	number	587	1.2.0
`profile.<name>.mail.smtp.auth`	Authenticate in SMTP	boolean	true	1.2.0
`profile.<name>.mail.smtp.starttls.enable`	Enable the starttls	boolean	true	1.2.0
`profile.<name>.mail.smtp.user`	The SMTP username	string	yoda@starwars.com	1.2.0
`profile.<name>.mail.from`	The source's email address	string	yoda@starwars.com	1.7
`profile.<name>.mail.smtp.password`	The SMTP password	string	UseTh3F0rc3	1.2.0
`profile.<name>.mail.date.format`	The date format to use, default is "yyyy-MM-dd HH:mm:ss"	string	yyyy-MM-dd HH:mm:ss	1.2.0

Default configuration in orientdb-server-config.xml. Example:

```
<!-- MAIL, TO TURN ON SET THE 'ENABLED' PARAMETER TO 'true' -->
<handler
class="com.orientechnologies.orient.server.plugin.mail.OMailPlugin">
  <parameters>
    <parameter name="enabled" value="true" />
    <!-- CREATE MULTIPLE PROFILES WITH profile.<name>... -->
    <parameter name="profile.default.mail.smtp.host" value="smtp.gmail.com"/>
    <parameter name="profile.default.mail.smtp.port" value="587" />
    <parameter name="profile.default.mail.smtp.auth" value="true" />
    <parameter name="profile.default.mail.smtp.starttls.enable" value="true" />
    <parameter name="profile.default.mail.from" value="test@gmail.com" />
    <parameter name="profile.default.mail.smtp.user" value="test@gmail.com" />
    <parameter name="profile.default.mail.smtp.password" value="mypassword" />
    <parameter name="profile.default.mail.date.format" value="yyyy-MM-dd HH:mm:ss" />
  </parameters>
</handler>
```

Usage

The message is managed as a map of properties containing all the fields those are part of the message.

Supported message properties:

Name	Description	Mandatory	Example	Since
from	source email address	No	to : "first@mail.com", "second@mail.com"	1.7
to	destination addresses separated by commas	Yes	to : "first@mail.com", "second@mail.com"	1.2.0
cc	Carbon copy addresses separated by commas	No	cc: "first@mail.com", "second@mail.com"	1.2.0
bcc	Blind Carbon Copy addresses separated by commas	No	bcc : "first@mail.com", "second@mail.com"	1.2.0
subject	The subject of the message	No	subject : "This Email plugin rocks!"	1.2.0
message	The message's content	Yes	message : "Hi, how are you mate?"	1.2.0
date	The subject of the message. Pass a java.util.Date object or a string formatted following the rules specified in "mail.date.format" configuration parameter or "yyyy-MM-dd HH:mm:ss" is taken	No, if not specified current date is assumed	date : "2012-09-25 13:20:00"	1.2.0
attachments	The files to attach	No		1.2.0

From Server-Side Functions

The Email plugin install a new variable in the server-side function's context: "mail". "profile" attribute is the profile name in configuration.

Example to send an email writing a function in JS:

```
mail.send({
      profile : "default",
      to: "orientdb@ruletheworld.com",
      cc: "yoda@starwars.com",
      bcc: "darthvader@starwars.com",
      subject: "The EMail plugin works",
      message : "Sending email from OrientDB Server is so powerful to build real web applications!"
    });
```

On Nashorn (>= Java8) the mapping of JSON to Map is not implicit. Use this:

```
mail.send( new java.util.HashMap{
      profile : "default",
      to: "orientdb@ruletheworld.com",
      cc: "yoda@starwars.com",
      bcc: "darthvader@starwars.com",
      subject: "The EMail plugin works",
      message : "Sending email from OrientDB Server is so powerful to build real web applications!"
  });
```

From Java

```
OMailPlugin plugin = OServerMain.server().getPlugin("mail");

Map<String, Object> message = new HashMap<String, Object>();
message.put("profile", "default");
message.put("to",      "orientdb@ruletheworld.com");
message.put("cc",      "yoda@starts.com,yoda-beach@starts.com");
message.put("bcc",     "darthvader@starwars.com");
message.put("subject", "The EMail plugin works");
message.put("message", "Sending email from OrientDB Server is so powerful to build real web applications!");

plugin.send(message);
```

JMX plugin

Java class implementation:

```
com.orientechnologies.orient.server.handler.OJMXPlugin
```

Available since: **v. 1.2.0**.

Introduction

Expose the OrientDB server configuration through JMX protocol. This task is configured as a Server handler. The task can be configured in easy way by changing parameters:

- **enabled**: true to turn on, false (default) is turned off
- **profilerManaged**: manage the Profiler instance

Default configuration in orientdb-server-config.xml

```
<!-- JMX SERVER, TO TURN ON SET THE 'ENABLED' PARAMETER TO 'true' -->
<handler class="com.orientechnologies.orient.server.handler.OJMXPlugin">
  <parameters>
    <parameter name="enabled" value="false" />
    <parameter name="profilerManaged" value="true" />
  </parameters>
</handler>
```

Rexster

Rexster provides a RESTful shell to any Blueprints-complaint graph database. This HTTP web service provides: a set of standard low-level GET, POST, and DELETE methods, a flexible extension model which allows plug-in like development for external services (such as ad-hoc graph queries through Gremlin), and a browser-based interface called The Dog House.

A graph database hosted in the OrientDB can be configured in Rexster and then accessed using the standard RESTful interface powered by the Rexster web server.

Installation

You can get the latest stable release of Rexster from its Download Page. The latest stable release when this page was last updated was *2.5.0*.

Or you can build a snapshot by executing the following Git and Maven commands:

```
git clone https://github.com/tinkerpop/rexster.git
cd rexster
mvn clean install
```

Rexster is distributed as a zip file (also the building process creates a zip file) hence the installation consist of unzipping the archive in a directory of your choice. In the following sections, this directory is referred to as *$REXSTER_HOME*.

After unzipping the archive, you should copy *orient-client.jar* and *orient-enterprise.jar* in *$REXSTER_HOME/ext*. Make sure you use the same version of OrientDB as those used by Rexster. For example Rexster 2.5.0 uses OrientDB 1.7.6.

You can find more details about Rexster installation at the Getting Started page.

Configuration

Refer to Rexster's Configuration page and OrientDB specific configuration page for the latest details.

Synopsis

The Rexster configuration file *rexster.xml* is used to configure parameters such as: TCP ports used by Rexster server modules to listen for incoming connections; character set supported by the Rexster REST requests and responses; connection parameters of graph instances.

In order to configure Rexster to connect to your OrientDB graph, locate the *rexster.xml* in the Rexster directory and add the following snippet of code:

```
<rexster>
  ...
  <graphs>
    ...
    <graph>
      <graph-enabled>true</graph-enabled>
      <graph-name>my-orient-graph</graph-name>
      <graph-type>orientgraph</graph-type>
      <graph-file>url-to-your-db</graph-file>
      <properties>
        <username>user</username>
        <password>pwd</password>
      </properties>
    </graph>
  ...
  </graphs>
</rexster>
```

In the configuration file, there could be a sample `graph` element for an OrientDB instance (`<graph-name>orientdbsample<graph-name>`): you might edit it according to your needs.

The `<graph-name>` element must be unique within the list of configured graphs and reports the name used to identify your graph. The `<graph-enabled>` element states whether the graph should be loaded and managed by Rexster. Setting its contents to `false` will prevent that graph from loading to Rexster; setting explicitly to `true` the graph will be loaded. The `<graph-type>` element reports the type of graph by using an identifier (`orientgraph` for an OrientDB Graph instance) or the full name of the class that implements the GraphConfiguration interface (com.tinkerpop.rexster.OrientGraphConfiguration for an OrientDB Graph).

The `<graph-file>` element reports the URL to the OrientDB database Rexster is expected to connect to:

- `plocal:*path-to-db*` , if the graph can be accessed over the file system (e.g. `plocal:/tmp/graph/db`)
- `remote:*url-to-db*` , if the graph can be accessed over the network and/or if you want to enable multiple accesses to the graph (e.g. `remote:localhost/mydb`)
- `memory:*db-name*` , if the graph resides in memory only. Updates to this kind of graph are never persistent and when the OrientDB server ends the graph is lost

The `<username>` and `<password>` elements reports the credentials to access the graph (e.g. `admin` `admin`).

Run

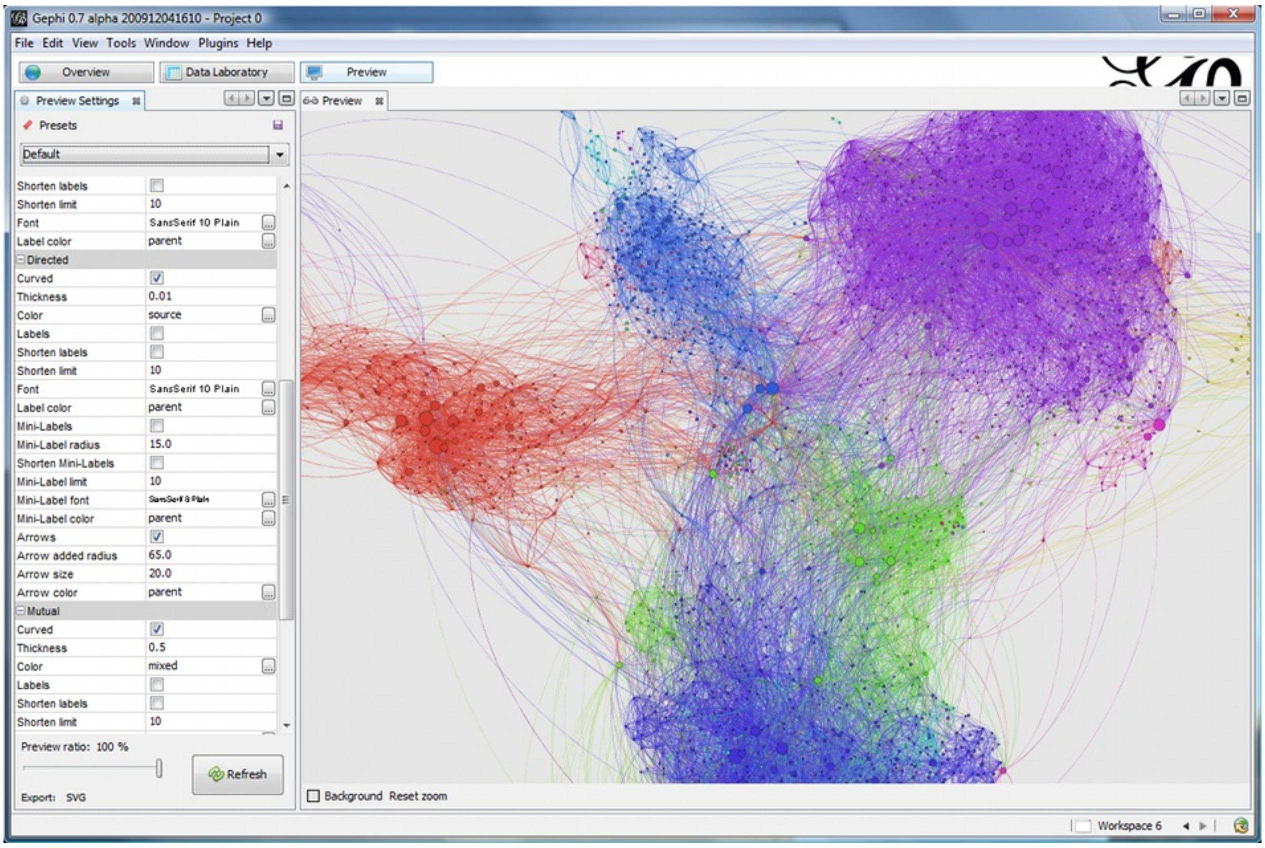
Note: only Rexster 0.5-SNAPSHOT and further releases work with OrientDB GraphEd
In this section we present a step-by-step guide to Rexster-ify an OrientDB graph.
We assume that:

- you created a Blueprints enabled graph called *orientGraph* using the class `com.tinkerpop.blueprints.pgm.impls.orientdb.OrientGraph`
- you inserted in the Rexster configuration file a `<graph>` element with the `<graph-name>` element set to `my-orient-graph` and the `graph-file` element set to `remote:orienthost/orientGraph` (if you do not remember how to do this, go back to the Configuration section).
- Be sure that the OrientDB server is running and you have properly configured the `<graph-file>` location and the access credentials of your graph.
- Execute the startup script (*$REXSTER_HOME/bin/rexster.bat* or *$REXSTER_HOME/bin/rexster.sh*)
- The shell console appears and you should see the following log message (line 10 states that the OrientDB graph instance has been loaded):

```
[INFO] WebServer - .:Welcome to Rexster:.
[INFO] GraphConfigurationContainer - Graph emptygraph - tinkergraph[vertices:0 edges:0] loaded
[INFO] RexsterApplicationGraph - Graph [tinkergraph] - configured with allowable namespace [tp:gremlin]
[INFO] GraphConfigurationContainer - Graph tinkergraph - tinkergraph[vertices:6 edges:6] loaded
[INFO] RexsterApplicationGraph - Graph [tinkergraph-readonly] - configured with allowable namespace [tp:gremlin]
[INFO] GraphConfigurationContainer - Graph tinkergraph-readonly - (readonly)tinkergraph[vertices:6 edges:6] loaded
[INFO] RexsterApplicationGraph - Graph [gratefulgraph] - configured with allowable namespace [tp:gremlin]
[INFO] GraphConfigurationContainer - Graph gratefulgraph - tinkergraph[vertices:809 edges:8049] loaded
[INFO] GraphConfigurationContainer - Graph sailgraph - sailgraph[memorystore] loaded
[INFO] GraphConfigurationContainer - Graph my-orient-graph - orientgraph[remote:orienthost/orientGraph] loaded
[INFO] GraphConfigurationContainer - Graph neo4jsample -  not enabled and not loaded.
[INFO] GraphConfigurationContainer - Graph dexsample -  not enabled and not loaded.
[INFO] MapResultObjectCache - Cache constructed with a maximum size of 1000
[INFO] WebServer - Web Server configured with com..sun..jersey..config..property..packages: com.tinkerpop.rexster
[INFO] WebServer - No servlet initialization parameters passed for configuration: admin-server-configuration
[INFO] WebServer - Rexster Server running on: [http://localhost:8182]
[INFO] WebServer - Dog House Server running on: [http://localhost:8183]
[INFO] ShutdownManager$ShutdownSocketListener - Bound shutdown socket to /127.0.0.1:8184. Starting listener thread for sh
utdown requests.
```

- Now you can use Rexster REST API and The Dog House web application to retrieve and modify the data stored in the OrientDB graph.

Gephi Visual Tool



Default

Shorten labels	☐
Shorten limit	10
Font	SansSerif 10 Plain
Label color	parent
Directed	
Curved	✓
Thickness	0.01
Color	source
Labels	☐
Shorten labels	☐
Shorten limit	10
Font	SansSerif 10 Plain
Label color	parent
Mini-Labels	☐
Mini-Label radius	15.0
Shorten Mini-Labels	☐
Mini-Label limit	10
Mini-Label font	SansSerif 8 Plain
Mini-Label color	parent
Arrows	✓
Arrow added radius	65.0
Arrow size	20.0
Arrow color	parent
Mutual	
Curved	✓
Thickness	0.5
Color	mixed
Labels	☐
Shorten labels	☐
Shorten limit	10

Preview ratio: 100 %

Export: SVG

Introduction

Gephi is a visual tool to manipulate and analyze graphs. Gephi is an Open Source project. Take a look at the amazing features.

Gephi can be used to analyze graphs extracted from OrientDB. There are 2 level of integration:

- the Streaming plugin that calls OrientDB server via HTTP. OrientDB exposes the new "/gephi" command in HTTP GET method that executes a query and returns the result set in "gephi" format.
- Gephi importer for Blueprints

In this mini guide we will take a look at the first one: the streaming plugin.

For more information:

- Gephi Graph Streaming format
- Graph Streaming plugin
- Tutorial video

Getting started

Before to start assure you've OrientDB 1.1.0-SNAPSHOT or greater.

Download and install

1. To download Gephi goto: http://gephi.org/users/download/
2. Install it, depends on your OS
3. Run Gephi
4. Click on the menu **Tools** -> **Plugins**

5. Click on the tab **Available Plugins**
6. Select the plugin **Graph Streaming**, click on the **Install** button and wait the plugin is installed

Import a graph in Gephi

Before to import a graph assure a OrientDB server instance is running somewhere. For more information watch this video.

1. Go to the **Overview** view (click on **Overview** top left button)
2. Click on the **Streaming** tab on the left
3. Click on the big + green button
4. Insert as **Source URL** the query you want to execute. Example: `http://localhost:2480/gephi/demo/sql/select%20from%20v/100` (below more information about the syntax of query)
5. Select as **Stream type** the **JSON** format (OrientDB talks in JSON)
6. Enable the **Use Basic Authentication** and insert the user and password of OrientDB database you want to access. The default user is "admin" as user and password
7. Click on **OK** button

Executing a query

The OrientDB's "/gephi" HTTP command allow to execute any query. The format is:

```
http://<host>:<port>/gephi/<database>/<language>/<query>[/<limit>]
```

Where:

- `host` is the host name or the ip address where the OrientDB server is running. If you're executing OrientDB on the same machine where Gephi is running use "localhost"
- `port` is the port number where the OrientDB server is running. By default is 2480.
- `database` is the database name
- `language`
- `query`, the query text following the URL encoding rules. For example to use the spaces use `%20`, so the query `select from v` becomes `select%20from%20v`
- `limit`, optional, set the limit of the result set. If not defined 20 is taken by default. `-1` means no limits

SQL Graph language

To use the OrientDB's SQL language use `sql` as language. For more information look at the SQL-Syntax.

For example, to return the first 1,000 vertices (class V) with outgoing connections the query would be:

```
SELECT FROM V WHERE out.size() > 0
```

Executed on "localhost" against the "demo" database + encoding becomes:

```
http://localhost:2480/gephi/demo/sql/select%20from%20V%20where%20out.size()%20%3E%200/1000
```

GREMLIN language

To use the powerful GREMLIN language to retrieve the graph or a portion of it use `gremlin` as language. For more information look at the GREMLIN syntax.

For example, to return the first 100 vertices:

```
g.V[0..99]
```

Executed on "localhost" against the "demo" database + encoding becomes:

```
http://localhost:2480/gephi/demo/gremlin/g.V%5B0..99%5D/-1
```

For more information about using Gephi look at Learn how to use Gephi

spider-box

spider-box is not really a "plug-in", but more a quick way to set up an environment to play with OrientDB in a local VM. It requires a virtualization system like Virtualbox, VM Ware Fusion or Parallels and the provisioning software Vagrant.

Once installed, you can very quickly start playing with the newest version of OrientDB Studio or the console. Or even start developing software with OrientDB as the database.

spider-box is configured mainly to build a PHP development environment. But, since it is built on Puphpet, you can easily change the configuration, so Python or even node.js is also installed. Ruby is installed automatically.

If you have questions about changing configuration or using spider-box, please do ask in an issue in the spider-box repo.

Have fun playing with OrientDB and spider-box!

Note: Neo4j and Gremlin Server are also installed, when you `vagrant up` spider-box.

Server Side Script Interpreter Plugin

Java class implementation:

```
com.orientechnologies.orient.server.handler.OServerSideScriptInterpreter
```

Available since: **v. 1.6.0**.

Introduction

Allows to execute script on the server.

Configuration

This plugin is configured as a Server handler. The plugin can be configured in easy way by changing parameters:

Name	Description	Type	Example	Since
enabled	true to turn on, false (default) is turned off	boolean	true	1.6.0
`allowedLanguages`	Array of languages allowed to be executed on the server	array of strings	SQL,Javascript	1.6.0

Default configuration in orientdb-server-config.xml. Example:

```
<!-- MAIL, TO TURN ON SET THE 'ENABLED' PARAMETER TO 'true' -->
<handler class="com.orientechnologies.orient.server.handler.OServerSideScriptInterpreter">
  <parameters>
  <parameter value="true" name="enabled"/>
  <parameter value="SQL,Javascript" name="allowedLanguages"/>
  </parameters>
</handler>
```

Usage

Look at Console Command JSS.

Contribute to OrientDB

In order to contribute issues and pull requests, please sign OrientDB's Contributor License Agreement. The purpose of this agreement is to protect users of this codebase by ensuring that all code is free to use under the stipulations of the Apache2 license.

Pushing into main repository

OrientDB uses different branches to support the development and release process. The `develop` branch contains code under development for which there's not a stable release yet. When a stable version is released, a branch for the hotfix is created. Each stable release is merged on master branch and tagged there. At the time of writing these notes, the state of branches is:

- develop: work in progress for next 3.0.x release (3.0.0-SNAPSHOT)
- 2.2.x: hot fix for next 2.2.x release (2.2.14-SNAPSHOT)
- 2.1.x: hot fix for next 2.1.x stable release (2.1.10-SNAPSHOT)
- 2.0.x: hot fix for next 2.0.x stable release (2.0.17-SNAPSHOT)
- last tag on master is 2.1.9

If you'd like to contribute to OrientDB with a patch follow the following steps:

- fork the repository interested in your change. The main one is https://github.com/orientechnologies/orientdb, while some other components reside in other projects under Orient Technologies umbrella.
- clone the forked repository
- select the branch, e.g the develop branch:
  - `git checkout develop`
- apply your changes with your favourite editor or IDE
- test that Test Suite hasn't been broken by running:
  - `mvn clean test`
- if all the tests pass, then do a **Pull Request** (PR) against the branch (e.g.: **"develop"**) on GitHub repository and write a comment about the change. Please don't send PR to "master" because we use that branch only for releasing

Documentation

If you want to contribute to the OrientDB documentation, the right repository is: https://github.com/orientechnologies/orientdb-docs. Every 24-48 hours all the contributions are reviewed and published on the public documentation.

Code formatting

You can find eclipse java formatter config file here: _base/ide/eclipse-formatter.xml.

If you use IntelliJ IDEA you can install this plugin and use formatter profile mentioned above.

Debugging

Run OrientDB as standalone server

The settings to run OrientDB Server as stand-alone (where the OrientDB's home is `/repositories/orientdb/releases/orientdb-community-2.2-SNAPSHOT`) are:

Main Class: `com.orientechnologies.orient.server.OServerMain` VM parameters:

```
-server
-DORIENTDB_HOME=/repositories/orientdb/releases/orientdb-community-2.2-SNAPSHOT
-Dorientdb.www.path=src/site
-Djava.util.logging.config.file=${ORIENTDB_HOME}/config/orientdb-server-log.properties
-Dorientdb.config.file=${ORIENTDB_HOME}/config/orientdb-server-config.xml
-Drhino.opt.level=9
```

Use classpath of module: `orientdb-graphdb`

Run OrientDB distributed

The settings to run OrientDB Server as distributed (where the OrientDB's home is `/repositories/orientdb/releases/orientdb-community-2.2-SNAPSHOT`) are:

Main Class: `com.orientechnologies.orient.server.OServerMain` VM parameters:

```
-server
-DORIENTDB_HOME=/repositories/orientdb/releases/orientdb-community-2.2-SNAPSHOT
-Dorientdb.www.path=src/site
-Djava.util.logging.config.file=${ORIENTDB_HOME}/config/orientdb-server-log.properties
-Dorientdb.config.file=${ORIENTDB_HOME}/config/orientdb-server-config.xml
-Drhino.opt.level=9
-Ddistributed=true
```

Use classpath of module: `orientdb-distributed`

In order to debug OrientDB in distributed mode, changed the scope to "runtime" in file distributed/pom.xml:

```
<groupId>com.orientechnologies</groupId>
<artifactId>orientdb-graphdb</artifactId>
<version>${project.version}</version>
<scope>runtime</scope>
```

In this way IDE like IntelliJ can start the server correctly that requires graphdb dependency.

Hackaton

Hackatons are the appointement where OrientDB old and new committers and contributors work together in few hours, on the same spot, or connected online.

The draft rules are (please contribute to improve it):

1. Committers will support contributors and new users on Hackaton
2. A new Google Hangout will be created, so if you want to attendee please send me your gmail/gtalk account
3. We'll use the hangout to report to the committer issues to close, or any questions about issues
4. We'll start from current release (1.7) and then go further (2.0, 2.1, no-release-tag)
5. If the issue is pretty old (>4 months), comment it about trying the last 1.7-rc2. We could have some chance the issue has already been fixed
6. If the problem is with a SQL query, you could try to reproduce against the GratefulDeadConcerts database or even an empty database. If you succeed on reproduce it, please comment with additional information about the issue

Contribution from Java Developers

1. If you're a Java developer and you can debug inside OrientDB code (that's would be great) you could include more useful information about the issue or even fix it
2. If you think the issue has been fixed with your patch, please run all the test cases with:
   - ant clean test
   - mvn clean test
3. If all tests pass, send us a Pull Request (see below)

Contribution to the Documentation

1. We're aware to have not the best documentation of the planet, so if you can improve on this would be awesome
2. JavaDoc, open a Java class, and:
   i. add the JavaDoc at the top of the class. This is the most important documentation in code we can have. if it's pertinent
   ii. add the JavaDoc for the public methods. It't better having a description about the method than the detail of all the parameters, exceptions, etc

Send a Pull Request!

We use GitHub and it's fantastic to work in a team. In order to make our life easier, the best way to contribute is with a Pull Request:

1. Goto your GitHub account. if you don't have it, create it in 2 minutes: www.github.com
2. Fork this project: https://github.com/orientechnologies/orientdb, or any other projects you want to contribute
3. Commit locally against the "develop" branch
4. Push your changes to your forked repository on GitHub
5. Send us a Pull Request and wait for the merging

Report an Issue

Very often when a new issue is open it lacks some fundamental information. This slows down the entire process because the first question from the OrientDB team is always "What release of OrientDB are you using?" and every time a Ferret dies in the world.

So please add more information about your issue:

1. OrientDB **release**? (If you're using a SNAPSHOT please attach also the build number found in "build.number" file)
2. What **steps** will reproduce the problem? 1. 2. 3.
3. **Settings**. If you're using custom settings please provide them below (to dump all the settings run the application using the JVM argument -Denvironment.dumpCfgAtStartup=true)
4. What is the **expected behavior** or output? What do you get or see instead?
5. If you're describing a performance or memory problem the **profiler** dump can be very useful (to dump it run the application using the JVM arguments -Dprofiler.autoDump.reset=true -Dprofiler.autoDump.interval=10 -Dprofiler.enabled=true)

Now you're ready to create a new one: https://github.com/orientechnologies/orientdb/issues/new

Get in touch

We want to make it super-easy for OrientDB users and contributors to talk to us and connect with each other, to share ideas, solve problems and help make OrientDB awesome. Here are the main channels we're running currently, we'd love to hear from you on one of them:

Google Group

OrientDB Google Group

The OrientDB Google Group (aka Community Group) is a good first stop for a general inquiry about OrientDB or a specific support issue (e.g. trouble setting OrientDB up). It's also a good forum for discussions about the roadmap or potential new functionality.

StackOverflow

StackOverflow OrientDB tag

Feel free to ask your questions on StackOverflow under "orientdb" and "orient-db" tags.

Gitter.io

chat on gitter

The best Web Chat, where we have an open channel. Use this if you have a question about OrientDB.

IRC

`#orientdb`

We're big fans of IRC here at OrientDB. We have a #orientdb channel on Freenode - stop by and say hi, you can even use Freenode's webchat service so don't need to install anything to access it.

Twitter

@orientdb

Follow and chat to us on Twitter.

GitHub

OrientDB issues

If you spot a bug, then please raise an issue in our main GitHub project orientechnologies/orientdb. Likewise if you have developed a cool new feature or improvement in your OrientDB fork, then send us a pull request against the "develop" branch!

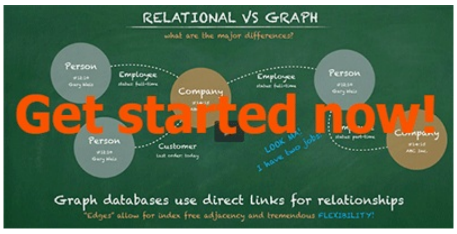
If you want to brainstorm a potential new feature, then the OrientDB Google Group (see above) is probably a better place to start.

Email

info@orientdb.com

If you want more information about Commercial Support, Consultancy or Training, email us.

More on Tutorials

 RELATIONAL VS GRAPH what are the major differences? Person Employee Company Customer Get started now! Graph databases use direct links for relationships "Edges" allow for index free adjacency and tremendous FLEXIBILITY!	We decided to provide a Getting Started video course for FREE! This course is designed to help developers become productive and familiar with OrientDB and related tools in the fastest way possible. For our initial launch, we have decided to use the Udemy.com platform to provide the most immersive, wide reaching platform possible.

This is a collection of tutorials about OrientDB.

External tutorials

Miscellaneous

- Getting Started with OrientDB|Video course by Petter Graff|
- Graph databases OrientDB to the rescue
- Graph in PHP through OrientDB
- GraphDB with flexible model

Italian

Step-by-step tutorial about the usage of OrientDB:

- Guida all'uso di OrientDB: introduzione al mondo NoSQL
- Guida all'uso di OrientDB: primo utilizzo
- Guida all'uso di OrientDB: i concetti di RecordID e Cluster
- Guida all'uso di OrientDB: Query SQL su un database NoSQL
- Guida all'uso di OrientDB: Comandi SQL
- Guida all'uso di OrientDB: Java API

HTML.it guide to OrientDB:

- Introduzione ad OrientDB

Tecnicume blog by Marco Berri:

- OrientDB - primi passi di Embedding in java
- Metodi di scrittura: ODocument e Pojo (Embedding in java)
- Import da csv relazionali, relazioni, archiviare file e query

Japanese

Try to manipulate the OrientDB from java (Part RawGraphDatabase):

- javaOrientDB(RawGraphDatabase)

Make GraphDB OrientDB app deployment experience:

1. Part 1

Step-by-step tutorial by Junji Takakura:

- Part 1
- Part 2
- Part 3

Presentations

Videos and Presentations in English

- Video *Switching from relational to the graph model* by Luca Garulli at All Your Base conference on November, 23rd 2012
    -
    - Slides
- Video **Graph databases and PHP: time for serious stuff** by Alessandro Nadalin and David Funaro at PHPcon Poland on October 21, 2011
    -
    - Slides
- Video: **Internet Apps powered by NoSQL and JavaScript** by Luca Garulli at JS Everywhere in Paris (France) on November 17th 2012
    -
    - Slides
- Video : **Switching from the relational to the graph model** by Luca Garulli at NoSQL Matters in Barcelona (Spain) on October 6th 2012
    -
    - Slides</td></tr>
- Video : **NoSQL adoption: what's the next step?** by Luca Garulli at NoSQL Matters in Cologne (Germany) on May 30th 2012
    -
    - Slides
- Video : **Design your applications using persistent Graphs and OrientDB** by Luca Garulli at NoSQL Matters in Cologne (Germany) on May 30th 2012
    -
    - Slides (English)
- Video (English): **Works with persistent graphs using OrientDB** by Luca Molino at FOSDEM on February 2012 in Bruxelles (Belgium) on Video
    - Slides (English)
- Video (pseudo-English): **Interview to Luca Garulli about OrientDB** by Tim Anglade on 2010
    -

Presentations in English

- Slides (English): OrientDB distributed architecture 1.1
- Slides (English): OrientDB the database for the Web 1.1
- What to select between the Document database and the Graph one?
- A walk in Graph Databases

Videos in Italian, Presentations in English

- Video (Italian): **Graph databases: time for the serious stuff** by Alessandro Nadalin and David Funaro at Codemotion in Rome on March 2012
    - Video
    - Slides (English)
- Video (Italian): **Dal modello Relazionale al Grafo: cosa cambia?** by Alfonso Focareta at Codemotion in Rome on March 2012
    - Video
    - Slides (English)
- Video (Italian): **Perché potresti avere bisogno di un database NOSQL anche se non sei Google o Facebook** (Italian only) by

Luca Garulli at Codemotion in Rome (Italy) on March 2011

  - http://www.orientdb.org/images/video-2011-codemotion-roma.png
  - Slides (English)

- Video (Italian): **OrientDB e lo sviluppo di WebApp** (Italian only) by Luca Garulli at NoSQL Day in Brescia on 2011
  -
  - Slides (English)

Roadmap

This page contains the roadmap with the main enhancements for the OrientDB product.

Terms

- **RC**: Release Candidate, is a beta version with potential to be a final product, which is ready to release unless significant bugs emerge. In this stage of product stabilization, all product features have been designed, coded and tested through one or more beta cycles with no known showstopper-class bug. A release is called code complete when the development team agrees that no entirely new source code will be added to this release. There could still be source code changes to fix defects, changes to documentation and data files, and peripheral code for test cases or utilities. Beta testers, if privately selected, will often be credited for using the release candidate as though it were a finished product. Beta testing is conducted in a client's or customer's location and to test the software from a user's perspective.
- **GA**: General Availability, is the stage where the software has "gone live" for usage in production. Users in production are suggested to plan a migration for the current GA evaluating pros and cons of the upgrade.

Release 3.0

```
- Development started on.: June 2016
- Expected first M1......: January 2017
- Expected first M2......: August 2017
- Expected first RC......: December 2017
- Expected final GA......: end of January 2018
```

Status

Last update: December 14, 2017

For a more detailed an updated view, look at the Roadmap 3.0 issue.

Module	Feature	Status
Core	Multi-Threads WAL	100%
Core	Index rebuild avoid using WAL	100%
Core	Improved DISKCACHE algorithm	100%
Core	Indexing of embedded properties	40%
Core	Parallel Transactions	100%
SQL	Multi-line queries in batch scripts	100%
Java API	New Multi-Model API	100%
Java API	Improve SQL UPDATE syntax	100%
Java API	Support for TinkerPop 3	100%
Remote protocol	Support for server-side transactions	100%
Remote protocol	Support for server-side cursors	100%
Remote protocol	Push messages on schema change	30%
Remote protocol	Push messages on record change	30%

Release 3.1

```
- Development started on.: -
- Expected first RC......: Q2 2018
- Expected final GA......: Q3 2018
```

Status

Last update: December 14, 2017

Module	Feature	Status
Core	WAL Compaction	30%
Core	Index per cluster	0%
Core	Override of properties	0%
Core	Compression of used space on serialization	3%
Core	Enhance isolation level also for remote commands	0%
Core	New data structure to manage edges	0%
Distributed	Auto-Sharding	10%
Distributed	Optimized replication for cross Data Center	0%
Distributed	Replication of in-memory databases	0%
Distributed	Optimized network protocol to send only the delta between updates	50%
Lucene	Faceted search	20%
SQL	Distributed SQL Executor	70%
SQL	shortestPaths() function	0%
SQL	New functions (strings, maths)	40%

Enterprise Edition

This is the main guide on using **OrientDB Enterprise Edition**. For more information look at OrientDB Enterprise Edition.

Enterprise Edition is a commercial product developed by OrientDB Ltd, the same company that lead the development of OrientDB Community Edition. Download now the 45-days trial.

OrientDB Enterprise Edition is designed specifically for applications seeking a scalable, robust and secure multi-model database. Its main goal is to save time and money on your OrientDB investment by reducing risk, cost, effort & time invested in a business critical application. It includes all Community features plus professional enterprise tools such as support for Data Centers, Query Profiler, Distributed Clustering configuration, Auditing Tools, Metrics recording, Live Monitor with configurable Alerts, Non-Stop Incremental Backups, Teleporter to import data from any Relational DBMS.

Installation

The installation procedure is the same as for the Community Edition, but using the package you download after registering on the web site.

At run-time, the Enterprise edition logs this message:

```
2016-08-04 09:38:26:589 INFO  ****************************************************************************** [OEnterpriseAgent]
2016-08-04 09:38:26:589 INFO  *                     ORIENTDB  -  ENTERPRISE EDITION                        * [OEnterpriseAgent]
2016-08-04 09:38:26:589 INFO  ****************************************************************************** [OEnterpriseAgent]
2016-08-04 09:38:26:589 INFO  * If you are in Production or Test, you must purchase a commercial license. * [OEnterpriseAgent]
2016-08-04 09:38:26:589 INFO  * For more information look at: http://orientdb.com/orientdb-enterprise/    * [OEnterpriseAgent]
2016-08-04 09:38:26:590 INFO  ****************************************************************************** [OEnterpriseAgent]
```

Upgrade from 2.1.x or previous

Before v2.2, the Enterprise Edition front end was the **Workbench** application. Starting from v2.2, the Workbench has been merged into Studio. When Studio runs on an Enterprise Edition, it enables the additional features automatically. Furthermore while the Workbench was a separate application that was connected to the servers, with the new Studio Enterprise, every server is a peer of the distributed cluster. You can configure any server by connecting to one of the server in the distributed cluster.

Explore the Enterprise Edition features:

- Dashboard
- Server Management
- Cluster Management
- Query Profiler
- Backup Management
- Studio Auditing
- Teleporter
- Data Centers

Auditing

Starting in OrientDB 2.1, the Auditing component is part of the Enterprise Edition. This page refers to the Auditing feature and how to work with it. The Studio web tool provides a GUI for Auditing that makes configuration easier. Look at the Auditing page in Studio.

By default all the auditing logs are saved as documents of class `AuditingLog`. If your account has enough privileges, you can directly query the auditing log. Example on retrieving the last 20 logs: `select from AuditingLog order by @rid desc limit 20`.

OrientDB 2.2 Starting in OrientDB 2.2, all auditing logs are now stored in the system database. The auditing log for each database is stored in a derived class of the `AuditingLog` class with this format: *databaseName*OAuditingLog.

As an example, if you have a database called *MyDB*, then the class name will be `MyDBOAuditingLog`.

Using the previous example to retrieve the last 20 log entries for a specific database, do this from within the system database: `select from MyDBOAuditingLog order by @rid desc limit 20`

Security First

For security reasons, no roles should be able to access the `AuditingLog` records. For this reason before using Auditing assure to revoke any privilege on the `AuditingLog` cluster. You can do that from Studio, security panel, or via SQL by using the SQL REVOKE command. Here's an example of revoking any access to the writer and reader roles:

```
REVOKE ALL ON database.cluster.auditinglog FROM writer
REVOKE ALL ON database.cluster.auditinglog FROM reader
```

OrientDB 2.2 This is no longer required starting in 2.2, since all auditing logs are stored in the system database. No local database user has access to the auditing logs stored in the system database. To grant access, a *system user* must be created in the system database with the appropriate role and permissions.

Polymorphism

OrientDB schema is polymorphic (taken from the Object-Oriented paradigm). This means that if you have the class "Person" and the two classes "Employee" and "Provider" that extend "Person", all the auditing settings on "Person" will be inherited by "Employee" and "Provider" (if the checkbox "polymorphic" is enabled on class "Person").

This makes your life easier when you want to profile only certain classes. For example, you could create an abstract class "Profiled" and let all the classes you want to profile extend it. Starting from v2.1, OrientDB supports multiple inheritance, so it's not a problem extending more classes.

security.json Configuration

There are two parts to enabling and configuring the auditing component. Starting in OrientDB 2.2, there is a `security.json` configuration file that resides under the `config` folder. See the OrientDB Security Configuraton documentation for more details.

The "auditing" section of the `security.json` file must be enabled for auditing to work.

Note the additional "distributed" section of "auditing" for logging distributed node events.

auditing-config.json Configuration

To configure auditing of a database, create a JSON configuration file with the name `auditing-config.json` under the database folder. This is the syntax for configuration:

```
{
  "classes": {
    "<class-name>" : {
      "polymorphic": <true|false>,
      "onCreateEnabled": <true|false>, "onCreateMessage": "<message>",
      "onReadEnabled": <true|false>, "onReadMessage": "<message>",
      "onUpdateEnabled": <true|false>, "onUpdateMessage": "<message>",
      "onUpdateChanges": <true|false>,
      "onDeleteEnabled": <true|false>, "onDeleteMessage": "<message>"
    }
  },
  "commands": [
    {
      "regex": "<regexp to match>",
      "message": "<message>"
    }
  ],
  "schema": {
      "onCreateClassEnabled": <true|false>, "onCreateClassMessage": "<message>",
      "onDropClassEnabled": <true|false>, "onDropClassMessage": "<message>"
  }
}
```

"classes"

Where:

- `classes` : contains the mapping per class. A wildcard `*` represents any class.
- `class-name` : class name to configure.
- `polymorphic` : If `true` , the auditing log also uses this class definition for all sub-classes. By default, the class definition is polymorphic.
- `onCreateEnabled` : If `true` , enables auditing of record creation events. Default is `false` .
- `onCreateMessage` : A custom message stored in the `note` field of the auditing record on create record events. It supports the dynamic binding of values, see "Customizing the Message", below.
- `onReadEnabled` : If `true` , enables auditing of record reading events. Default is `false` .
- `onReadMessage` : A custom message stored in the `note` field of the auditing record on read record events. It supports the dynamic binding of values, see "Customizing the Message", below.
- `onUpdateEnabled` : If `true` , enables auditing of record updating events. Default is `false` .
- `onUpdateMessage` : A custom message stored in the `note` field of the auditing record on update record events. It supports the dynamic binding of values, see "Customizing the Message", below.
- `onUpdateChanges` : If `true` , records all the changed field values in the `changes` property. The default is `true` if `onUpdateEnabled` is true.
- `onDeleteEnabled` : If `true` , enables auditing of delete record events. The default is `false` .
- `onDeleteMessage` : A custom message stored in the `note` field of the auditing record on delete record events. It supports the dynamic binding of values, see "Customizing the Message", below.

Customing the Message

- `${field.<field-name>}` , to use the field value. Example: `${field.surname}` to get the field "surname" from the current record in case of CRUD Auditing of classes.

Example to log all the delete operations (`class="*"`), and log all the CRUD operation on any vertex (`class="V" and polymorphic:true`):

```
{
  "classes": {
    "*" : {
      "onDeleteEnabled": true, "onDeleteMessage": "Deleted record of class ${field.@class}"
    },
    "V" : {
      "polymorphic": true,
      "onCreateEnabled": true, "onCreateMessage": "Created vertex of class ${field.@class}",
      "onReadEnabled": true, "onReadMessage": "Read vertex of class ${field.@class}",
      "onUpdateEnabled": true, "onUpdateMessage": "Updated vertex of class ${field.@class}",
      "onDeleteEnabled": true, "onDeleteMessage": "Deleted vertex of class ${field.@class}"
    }
  }
}
```

"commands"

Where:

- `regexp` : If a command is executed that matches the regular expression then the command is logged.
- `message` : The optional message that's recorded when the command is logged. It supports the dynamic binding of values, see "Customizing the Message", below.

Customize the Message

- The variable `${command}` will be substituted in the specified message, if command auditing is enabled.

"schema"

Where:

- `onCreateClassEnabled` : If `true` , enables auditing of class creation events. The default is `false` .
- `onCreateClassMessage` : A custom message stored in the `note` field of the auditing record on class creation events. It supports the dynamic binding of values, look at Customize the message.
- `onDropClassEnabled` : If `true` , enables auditing of class drop events. The default is `false` .
- `onDropClassMessage` : A custom message stored in the `note` field of the auditing record on drop class events. It supports the dynamic binding of values, look at Customize the message.

Customing the Message

- The variable `${class}` will be substituted in the specified message, if create class or drop class auditing is enabled.

Log record structure

Auditing Log records have the following structure:

Field	Type	Description	Values
`date`	DATE	Date of execution	-
`user`	LINK	User that executed the command. Can be `null` if internal user has been used	-
`operation`	BYTE	Type of operation	0=READ, 1=UPDATE, 2=DELETE, 3=CREATE, 4=COMMAND
`record`	LINK	Link to the record subject of the log	-
`note`	STRING	Optional message	-
`changes`	MAP	Only for UDPATE operation, contains the map of changed fields in the form `{"from":<old-value>, "to":<new-value>}`	-

OrientDB 2.2 Starting in 2.2, the *AuditingLog* class has changed slightly.

Field	Type	Description	Values
`database`	STRING	The name of the database of the logging event.	-
`date`	DATE	Date of execution	-
`user`	STRING	The name of the user that executed the command. This type has changed to a String and now supports system users.	-
`operation`	BYTE	Type of operation	0=READ, 1=UPDATE, 2=DELETE, 3=CREATE, 4=COMMAND
`record`	LINK	Link to the record subject of the log.	-
`note`	STRING	Optional message	-
`changes`	MAP	Only for UDPATE operation, contains the map of changed fields in the form `{"from":<old-value>, "to":<new-value>}`	-

Tutorials

This Chapter includes all the Tutorials available in this Manual.

OrientDB Release Notes

- OrientDB 2.2 Release Notes