Table of Contents

OrientDB Manual - version 2.1.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.

Past releases

- v1.7.8
- v2.0.x

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** This 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** OrientDB 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:
    - 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.

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
> ```

Installing OrientDB

There are two methods available to install OrientDB, with some variations on each depending on your operating system. The first method is to download a binary package from OrientDB. The other method is to compile the package from the source code.

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://orientdb.com/download.php?file=orientdb-community-2.1.2.tar.gz
```

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-2.1.2` 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 2.2.x release (2.2.0-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

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-2.1.10-SNAPSHOT.dir
48814386 26 Jan 09:57 orientdb-community-2.1.10-SNAPSHOT.tar.gz
53542231 26 Jan 09:58 orientdb-community-2.1.10-SNAPSHOT.zip
$
```

The directory `orientdb-community-2.1.10-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 2.1.x
$ mvn clean install
```

builds the `2.1.x` branch, instead of `master`.

Other Resources

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

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

Install as Service on Unix/Linux

Following the installation guide above, whether you chose 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.

Installing OrientDB on a systemd-based Linux distribution requires that you write a service file set to use the `orientdb.sh` script in launching the database server. Place this file in the systemd configuration directory, (for instance, `/etc/systemd/` :

```
# vi /etc/systemd/system/orientdb.service

[Unit]
Description=OrientDB Server
After=network.target
After=syslog.target

[Install]
WantedBy=multi-user.target

[Service]
Type=forking
ExecStart=$ORIENTDB_HOME/bin/orientdb.sh start
ExecStop=$ORIENTDB_HOME/bin/orientdb.sh stop
ExecStatus=$ORIENTDB_HOME/bin/orientdb.sh status
```

You may want to use the absolute path instead of the environmental 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

For Mac OS X, create an alias to 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>
```

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 native binaries download area...**: you will see the index **commons/daemon/binaries/** (even if the title in the page reports **Index of dist/commons**).
3. Click on **windows**. Now you can see the index of **commons/daemon/binaries/windows**.
4. Click on **commons-daemon-1.0.7-bin-windows.zip**. The download starts.
5. Unzip the file in a directory of your choice. The content of the archive is depicted below:

```
commons-daemon-1.0.7-bin-windows
|
\---amd64
    |
    \---prunsrv.exe
|
\---ia64
    |
    \---prunsrv.exe
|
\---LICENCE.txt
|
\---NOTICE.txt
|
\---prunmgr.exe
|
\---prunsrv.exe
|
\---RELEASE-NOTES.txt
```

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** and **prunmgr** according to the architecture of your machine.

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
OrientDBGraphX.X.X.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\*" --JvmOptions "-Dfile.Encoding=%ORIENTDB_ENCODING%;-Djava.util.logging.config.file="%LOG_FI
LE%";-Dorientdb.config.file="%CONFIG_FILE%";-Dorientdb.www.path="%WWW_PATH%";-Dlog.console.level=%LOG_CONSOLE_LEVEL%;-Dlog.fil
e.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 1.0rc5**). You can also use *%ORIENTDB_HOME%\service\OrientDBGraphw.exe* to manage and monitor the *OrientDBGraph* service.

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

OrientDB is the first Multi-Model Open Source NoSQL DBMS that combines the power of graphs and the flexibility of documents into one scalable, high-performance operational database.

This repository is a dockerfile for creating an orientdb image with :

- explicit orientdb version (orientdb-2.0) for image cache stability
- init by supervisord
- config, databases and backup folders expected to be mounted as volumes

And lots of information from my orientdb+docker explorations. Read on!

Building the image on your own

1. Clone this project to a local folder:

   ```
   git clone https://github.com/orientechnologies/orientdb-docker.git
   ```

2. Build the image:

   ```
   docker build -t <YOUR_DOCKER_HUB_USER>/orientdb-2.0 .
   ```

3. Push it to your Docker Hub repository (it will ask for your login credentials):

   ```
   docker push <YOUR_DOCKER_HUB_USER>/orientdb-2.0
   ```

All examples below are using an image from nesrait/orientdb-2.0. If you build your own image please find/replace "nesrait" with your Docker Hub user.

Running Orientdb

To run the image, run:

```
docker run --name orientdb -d -v <config_path>:/opt/orientdb/config -v <databases_path>:/opt/orientdb/databases -v <backup_pat
h>:/opt/orientdb/backup -p 2424 -p 2480 nesrait/orientdb-2.0
```

The docker image contains a unconfigured Orientdb installation and for running it, you need to provide your own config folder from which OrientDB will read its startup settings.

The same applies for the databases folder which if local to the running container would go away as soon as it died/you killed it.

The backup folder only needs to be mapped if you activate that setting on your OrientDB configuration file.

Persistent distributed storage using BTSync

If you're not running OrientDB in a distributed configuration you need to take special care to backup your database (in case your host goes down).

Below is a simple, yet hackish, way to do this: using BTSync data containers to propagate the OrientDB config, LIVE databases and backup folders to remote location(s). Note: don't trust the remote copy of the LIVE database folder unless the server is down and it has correctly flushed changes to disk.

1. Create BTSync shared folders on any remote location for the various folder you want to replicate

   1.1. config: orientdb configuration inside the config folder

1.2. databases: the LIVE databases folder

1.3. backup: the place where OrientDB will store the zipped backups (if you activate the backup in the configuration file)

2. Take note of the BTSync folder secrets CONFIG_FOLDER_SECRET, DATABASES_FOLDER_SECRET, BACKUP_FOLDER_SECRET

3. Launch BTSync data containers for each of the synched folder you created giving them proper names:

```
docker run -d --name orientdb_config -v /opt/orientdb/config nesrait/btsync /opt/orientdb/config CONFIG_FOLDER_SECRET
docker run -d --name orientdb_databases -v /opt/orientdb/databases nesrait/btsync /opt/orientdb/databases DATABASES_FOLDE
R_SECRET
docker run -d --name orientdb_backup -v /opt/orientdb/backup nesrait/btsync /opt/orientdb/backup BACKUP_FOLDER_SECRET
```

4. Wait until all files have magically appeared inside your BTSync data volumes: ```bash docker run --rm -i -t --volumes-from orientdb_config --volumes-from orientdb_databases --volumes-from orientdb_backup ubuntu du -h /opt/orientdb/config /opt/orientdb/databases /opt/orientdb/backup

~~~
5. Finally you're ready to start your OrientDB server
  ```bash
docker run --name orientdb -d \
            --volumes-from orientdb_config \
            --volumes-from orientdb_databases \
            --volumes-from orientdb_backup \
            -p 2424 -p 2480 \
            nesrait/orientdb-2.0
~~~

OrientDB distributed

If you're running OrientDB distributed* you won't have the problem of losing the contents of your databases folder since they are already replicated to the other OrientDB nodes. From the setup above simply leave out the "--volumes-from orientdb_databases" argument and OrientDB will use the container storage to hold your databases' files.

*note: some extra work might be needed to correctly setup hazelcast running inside docker containers (see this discussion).

Ad-hoc backups

With OrientDB 2.0 we can now create ad-hoc backups by taking advantage of the new backup.sh script:

- Using the orientdb_backup data container that was created above:

```
docker run -i -t --volumes-from orientdb_config --volumes-from orientdb_backup nesrait/orientdb-2.0 ./backup.sh <dburl> <
user> <password> /opt/orientdb/backup/<backup_file> [<type>]
```

- Or using a host folder:

`docker run -i -t --volumes-from orientdb_config -v <host_backup_path>:/backup nesrait/orientdb-2.0 ./backup.sh <dburl> <user> <password> /backup/<backup_file> [<type>]`

Either way, when the backup completes you will have the backup file located outside of the OrientDB container and read for safekeeping.

Note: I haven't tried the non-blocking backup (type=lvm) yet but found this discussion about a docker LVM dependency issue.

Running the Orientdb console

```
docker run --rm -it \
            --volumes-from orientdb_config \
            --volumes-from orientdb_databases \
            --volumes-from orientdb_backup \
            nesrait/orientdb-2.0 \
            /opt/orientdb/bin/console.sh
```

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 in 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

There are various methods you can use to connect to your database server and the individual databases, once the server is running, such as 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` on Unix-based systems and on Windows `console.bat`.

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 run while only connected to the 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.

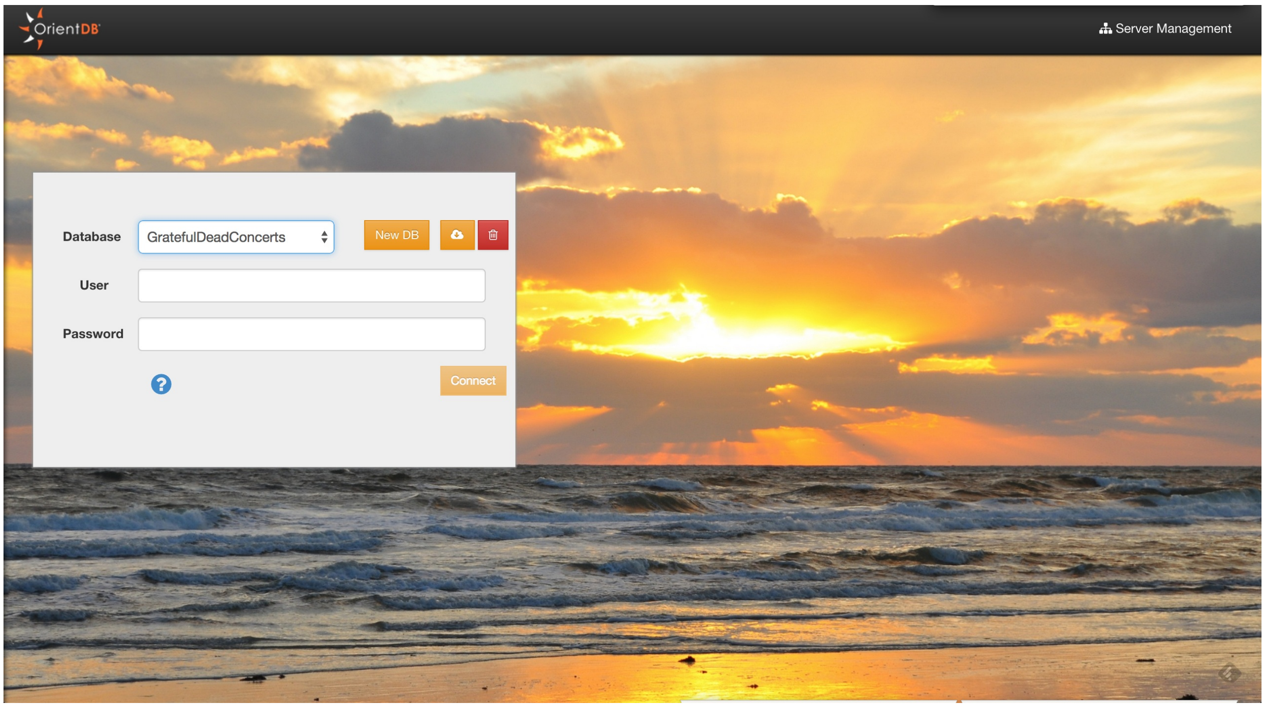
Run the Studio

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

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.

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

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.

Basic Concepts

The 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.

Documents

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>` .

RecordBytes

In addition to the Document record type, OrientDB can also load and store binary data. The RecordBytes record type is similar to the `BLOB` data type in Relational databases.

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 `invoice2015` and `invoice2016` . 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 = 2012
```

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:invoice2012
```

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

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 1:*n* 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 thier 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 RecordId	`ORecord`	- -	ORecord
10	Embedded list	The Records are contained inside the owner. The contained records have no RecordIds 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 RecordId 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 RecordIds 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 RecordIds are stored	`List<? extends ORecord`	0 41,000,000 items	String
				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 RecordIds 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 RecordIds 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 `17` , 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.

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 a reduced 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 retire 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 teh `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.

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.

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 it 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
         return new Random().nextInt(clusters.length);
      }

      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/service`. 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'
```

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 `CLASSES` command in the console:

```
orientdb> 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 them for use. As a concept, the class in OrientDB has the closest relationship with the table in relational databases, but (unlike tables) classes can be schema-less, schema-full or mixed. Classes can inherit from other classes, creating trees of classes. Each class has its own cluster or clusters, (created by default, if none are defined).

> 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.

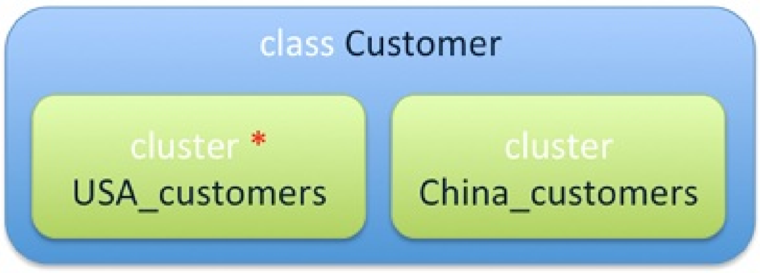
To list all the configured clusters on your system, use the `CLUSTERS` command in the console:

```
orientdb> 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.

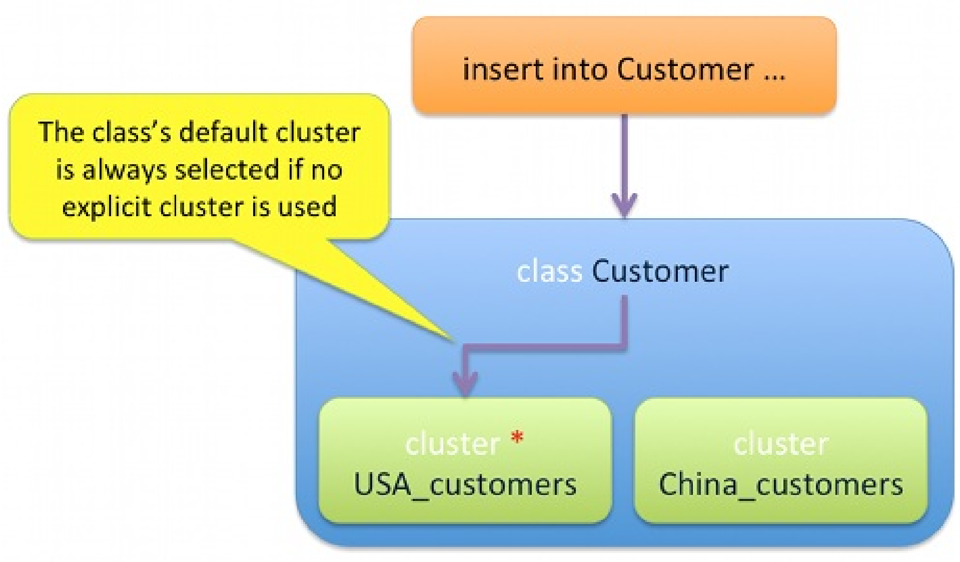
While the default strategy is that each class maps to one cluster, a class can rely on multiple clusters. For instance, you can spawn records physically in multiple locations, thereby creating multiple clusters.



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

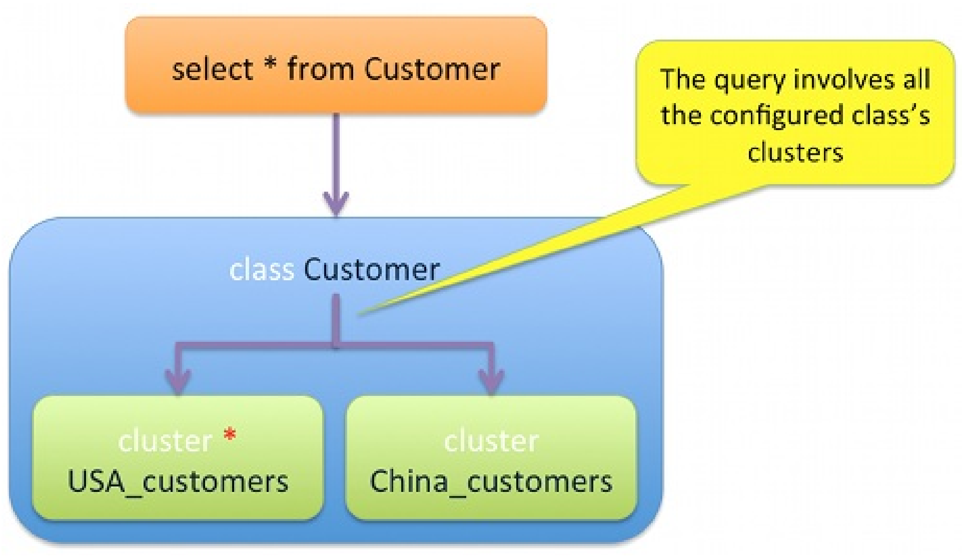
- `USA_customers`, which is a cluster that contains all customers in the United States.
- `China_customers`, which is a cluster that contains all customers in China.

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.



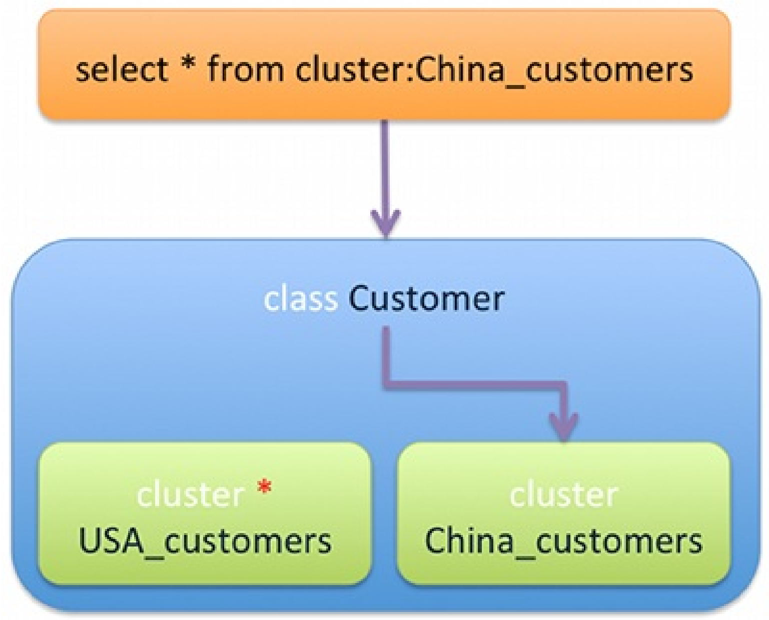
The new entry from the `INSERT` statement is added to the `USA_customers` cluster, given that it's the default. Inserting data into a non-default cluster would require that you specify the cluster you want to insert the data into in your statement.

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

In OrientDB there are two types of clusters:

- **Physical Cluster** (known as **local**) which is persistent because it writes directly to the file system
- **Memory Cluster** where everything is volatile and will be lost on termination of the process or server if the database is remote

For most cases, physical clusters are preferred because databases must be persistent. OrientDB creates physical clusters by default.

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.

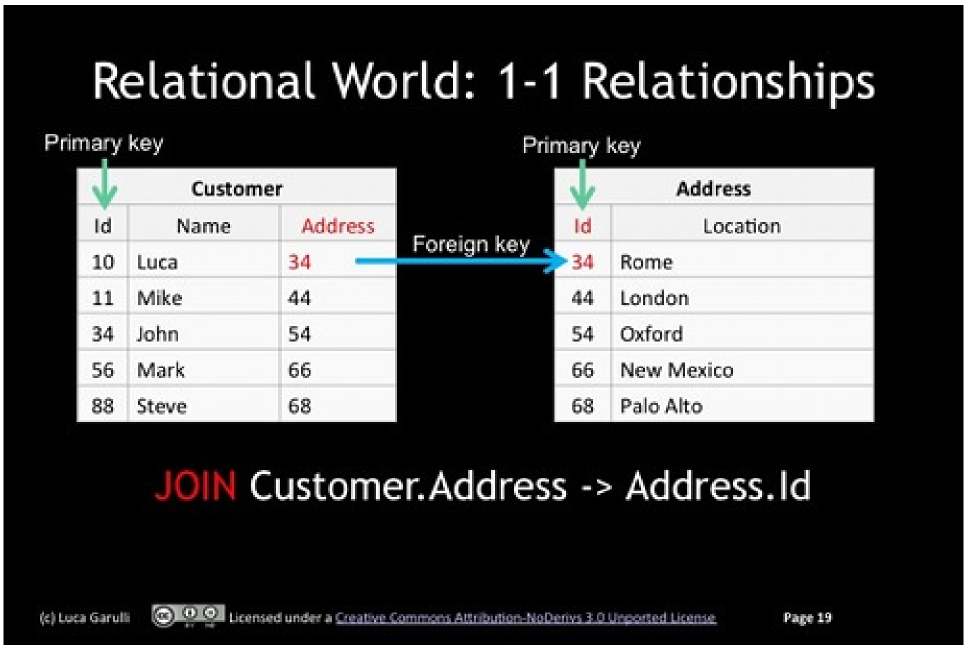
Relations in Relational Databases

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.



Relational World: 1-1 Relationships

Customer		
Id	Name	Address
10	Luca	34
11	Mike	44
34	John	54
56	Mark	66
88	Steve	68

Address	
Id	Location
34	Rome
44	London
54	Oxford
66	New Mexico
68	Palo Alto

JOIN Customer.Address -> Address.Id

(c) Luca Garulli Licensed under a Creative Commons Attribution-NoDerivs 3.0 Unported License Page 19

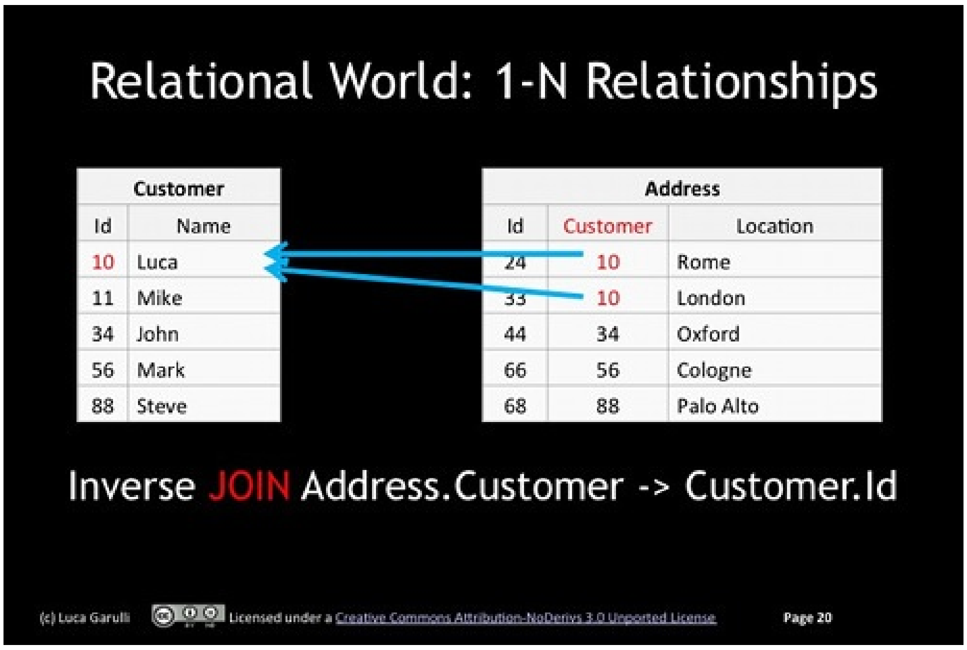
Consider a case where you want to view the address of a customer named Luca. In a Relational database, like MySQL, this is how you would query the table:

```
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.



Relational World: 1-N Relationships

Customer	
Id	Name
10	Luca
11	Mike
34	John
56	Mark
88	Steve

Address		
Id	Customer	Location
24	10	Rome
33	10	London
44	34	Oxford
66	56	Cologne
68	88	Palo Alto

Inverse JOIN Address.Customer -> Customer.Id

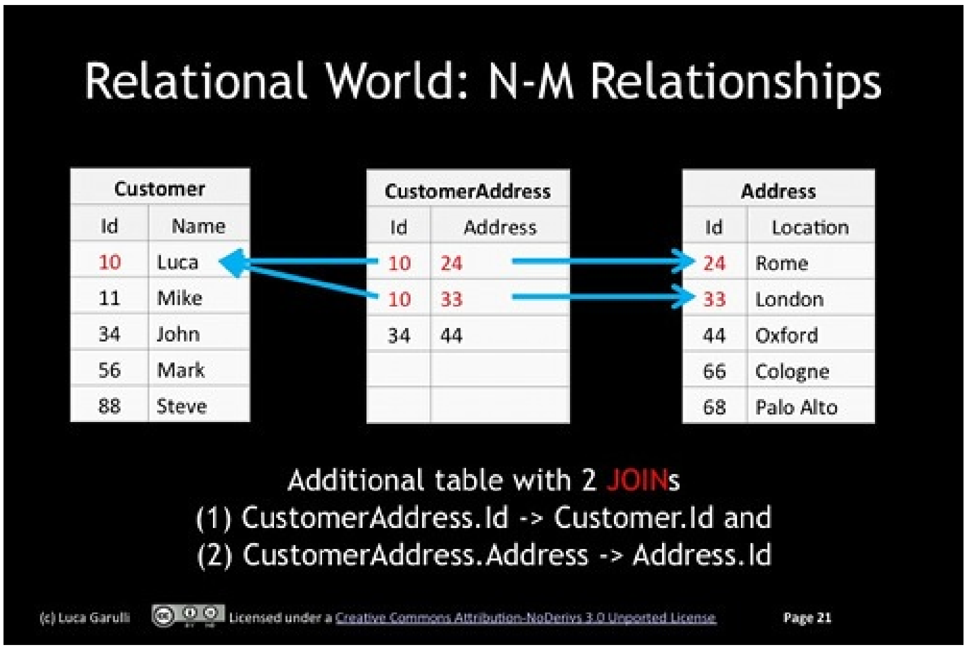
(c) Luca Garulli Licensed under a Creative Commons Attribution-NoDerivs 3.0 Unported License Page 20

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.



Relational World: N-M Relationships

Customer	
Id	Name
10	Luca
11	Mike
34	John
56	Mark
88	Steve

CustomerAddress	
Id	Address
10	24
10	33
34	44

Address	
Id	Location
24	Rome
33	London
44	Oxford
66	Cologne
68	Palo Alto

Additional table with 2 JOINs
(1) CustomerAddress.Id -> Customer.Id and
(2) CustomerAddress.Address -> Address.Id

(c) Luca Garulli Licensed under a Creative Commons Attribution-NoDerivs 3.0 Unported License Page 21

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 B.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.

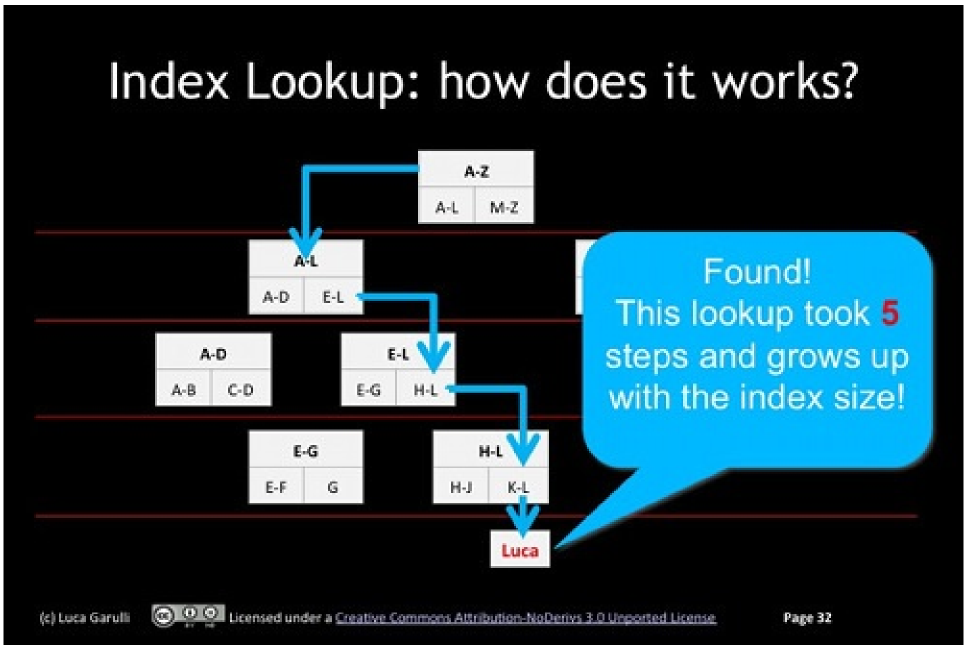
Searching for an identifier at runtime each time you execute a query, for every record will grow very expensive. The first optimization with relational databases is the use of indexing. Indexes speed up searches, but they slow down `INSERT`, `UPDATE`, and `DELETE` operations. Additionally, they occupy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


Index Lookup: how does it works?
A-Z
A-L
M-Z
A-L
A-D
E-L
A-D
A-B
C-D
E-L
E-G
H-L
E-G
E-F
G
H-L
H-J
K-L
Luca
Found! This lookup took 5 steps and grows up with the index size!
(c) Luca Garulli
Licensed under a Creative Commons Attribution-NoDerivs 3.0 Unported License
Page 32

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>` .

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
- 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 `#12:0` and `#11:2` . Create an edge that connects them.

```
orientdb> CREATE EDGE Friend FROM #12:0 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   | #12:0     |
-------+-------------+-------------+---------+-----------+
```

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 `#12:0` 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.

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. Connections that use the Binary Protocol can *early load* records on the client's. On traversing of connected records, the client hasn't to execute further remote calls to the server, because the requested records are already on the client's cache
2. Connections that use the HTTP/JSON Protocol can *expand the resulting JSON* to include connected records as embedded in the same JSON. This is useful on 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 tells 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 to third level. You can also use the partial range syntax: `[-3]` means from the first to fourth level 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 taht 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, it would require seven different loads to build the return 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 directly a single record to 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 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 UNION function to create a unique result set:

```
SELECT expand( records ) from (
  SELECT union( 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 union( 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 wit 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:

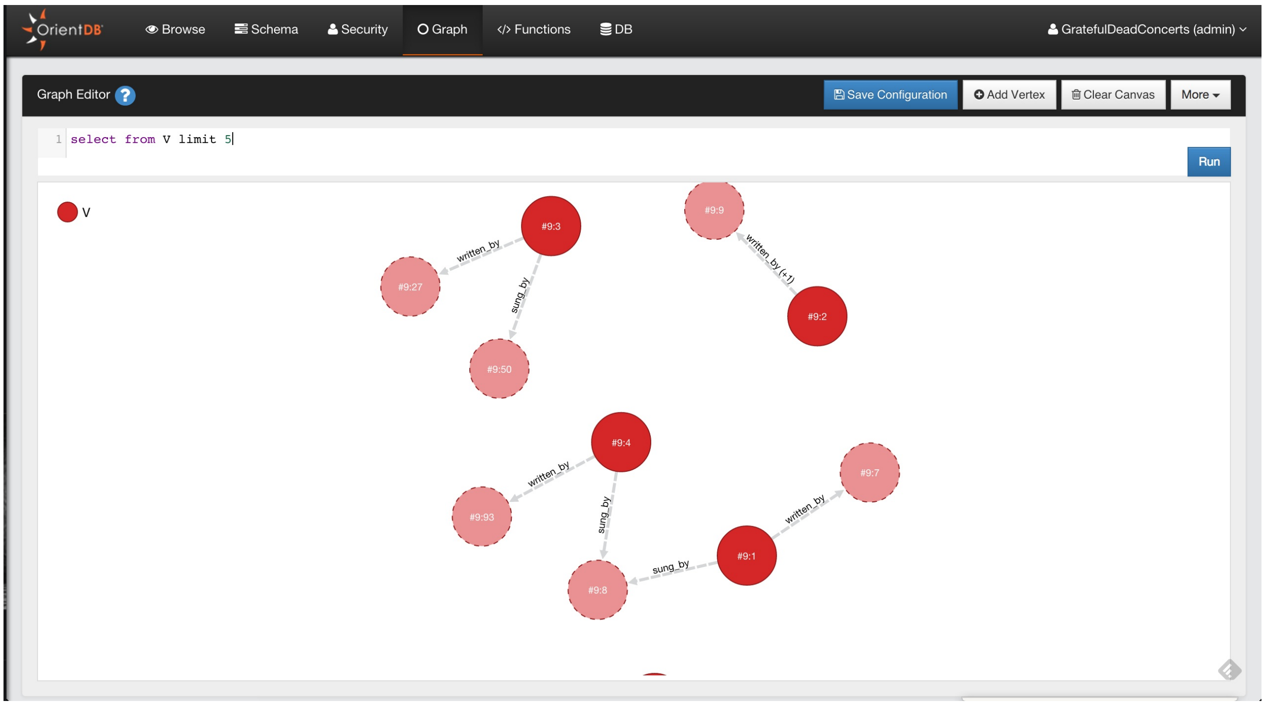
```
select from queue limit 1
```

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:
>
> - Command Reference
> - Backup and Restore
> - Export and Import
> - Logging
> - Studio
> - Trouble shooting
> - Performance Tuning
> - ETL Tools

Console Tool

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
```

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
`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:
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 in 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.

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.

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.

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 - 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>]]
```

- **`<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.

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>]
      [-includeSchemsa=<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 v1 = g.addVertex();
  [Started multi-line command.  Type just 'end' to finish and execute.]

  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.

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> [-preserveClusterIDs = <true|false>]
                             [-merge = <true|false>]
                             [-migrateLinks = <true|false>]
                             [-rebuildIndexes = <true|false>]
```

- **`<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.
- **`-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 givne 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 - `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.

Syntax

```
LIST CONNECTIONS
```

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 - `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 - `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 1.7.0 -> 1.7.8). 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			
2.0.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.			
1.2.x	1.3.x	n.a.	Changed v2.2.x	OK	OK	OK	Need export & Re-import	12, 13	n.a.

References:

- Binary Network Protocol: Network Binary Protocol
- HTTP Network Protocol: OrientDB REST

Migrate from LOCAL storage engine to PLOCAL

Starting from version 1.5.x OrientDB comes with a brand new storage engine: PLOCAL (Paginated LOCAL). It's persistent like the LOCAL, but stores information in a different way. Below are the main differences with LOCAL:

- records are stored in cluster files, while with LOCAL was split between cluster and data-segments
- more durable than LOCAL because the append-on-write mode
- minor contention locks on writes: this means more concurrency
- it doesn't use Memory Mapping techniques (MMap) so the behavior is more "predictable"

To migrate your LOCAL storage to the new PLOCAL, you need to export and reimport the database using PLOCAL as storage engine. Follow the steps below:

1) open a new shell (Linux/Mac) or a Command Prompt (Windows)

2) export the database using the console. Example by exporting the database under /temp/db:

```
$ bin/console.sh (or bin/console.bat under Windows)
orientdb> CONNECT DATABASE local:/temp/db admin admin
orientdb> EXPORT DATABASE /temp/db.json.gzip
orientdb> DISCONNECT
```

3) now always in the console create a new database using the "plocal" engine:

a) on a local filesystem:

```
  orientdb> CREATE DATABASE plocal:/temp/newdb admin admin plocal graph
```

b) on a remote server (use the server's credentials to access):

```
  orientdb> CREATE DATABASE remote:localhost/newdb root password plocal graph
```

4) now always in the console import the old database in the new one:

```
orientdb> IMPORT DATABASE /temp/db.json.gzip -preserveClusterIDs=true
orientdb> QUIT
```

5) If you access to the database in the same JVM remember to change the URL from "local:" to "plocal:"

Migrate graph to RidBag

As of OrientDB 1.7 the RidBag is default collection that manages adjacency relations in graphs. While the older database managed by an MVRB-Tree are fully compatible, you can update your database to the more recent format.

You can upgrade your graph via console or using the ORidBagMigration class

Using console

- Connect to database `CONNECT plocal:databases/GratefulDeadConcerts`
- Run `upgrade graph` command

Using the API

- Create OGraphMigration instance. Pass database connection to constructor.
- Invoke method execute()

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?

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`.

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 progress-

Parallel queries

Starting from v2.2, the OrientDB SQL executor will decide if execute or not a query in parallel. Before v2.2 executing parallel queries could be done only manually by appending the `PARALLEL` keyword at the end of SQL SELECT. Issue 4578.

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 89.

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")` is not supported anymore. Issue 4505.

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.

Security and speed

OrientDB v2.2 increase security by using SALT. This means that hashing of password is much slower than OrientDB v2.1. You can configure the number of cycle for SALT: more is harder to decode but is slower. Change 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`

To improve performance consider also avoiding opening and closing connection, but rather using a connection pool.

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 not supported anymore. This is because we simplified the `.field()` method to don't accept expressoion anymore. This allows to boost up performance on such used method. 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"
```

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

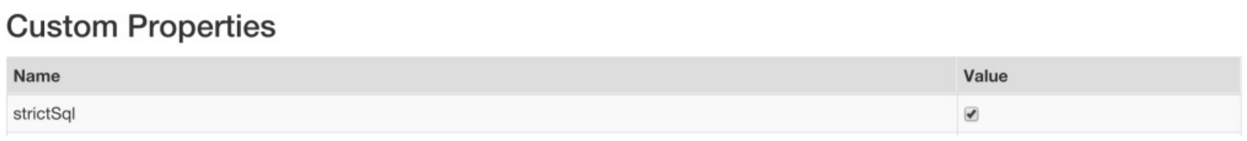
In 2.0.x difference() function had inconsistent behavior: it actually worked as a symmetric difference (see 4366, 3969) In 2.1 it was refactored to perform normal difference (https://proofwiki.org/wiki/Definition:Set_Difference) and another function was created for symmetric difference (called "symmetricDifference()").

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 back 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.

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.

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 back up.
- **`<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.

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 non-blocking LVM backup, using a remote database hosted at `localhost` :

```
$ $ORIENTDB_HOME/bin/backup.sh remote:localhost/testdb \
      root rootpasswd \
      /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.
```

> **NOTE** Non-blocking backups require that the operating system support LVM. For more information, see
>
> - LVM
> - File system snapshots with LVM
> - LVM snapshot backup

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

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 -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 and can export in GraphML, an XML-based file format for graphs. Given that OrientDB can read GraphML, it is relatively straightforward to import data from Neo4j into OrientDB. You can manage the imports using the Console or the Java API.

> Neo4j is a registered trademark of Neo Technology, Inc. For more information on the differences between Neo4j and OrientDB, see OrientDB vs. Neo4j.

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.

Exammples

- 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")
;
```

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 reocrd 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. Follow these steps to use the module:

- Clone the repository on your computer, by executing:
  - `git clone https://github.com/orientechnologies/orientdb-etl.git`
- Compile the module, by executing:
  - `mvn clean install`
- Copy `script/oetl.sh` (or .bat under Windows) to $ORIENTDB_HOME/bin
- Copy `target/orientdb-etl-2.0-SNAPSHOT.jar` to $ORIENTDB_HOME/lib

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 Block

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 Block

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:

- `"file"`
- `"input"`
- `"http"`

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"
   }
}
```

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:{}}]}"
```

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`
`"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`
`"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 HH:MM"
    }
}
```

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 = ${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`
`"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` .

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 }
        }
      ]
    }
```

Import Database of Beers in OrientDB

First, create a new folder somewhere on your hard drive. For this test we'll assume `/temp/openbeer`.

```
$ mkdir /temp/openbeer
```

Download Beers Database in CSV format

```
$ curl http://openbeerdb.com/data_files/openbeerdb_csv.zip > openbeerdb_csv.zip
$ unzip openbeerdb_csv.zip
```

Install OrientDB

```
$ curl "http://orientdb.com/download.php?email=unknown@unknown.com&file=orientdb-community-2.0.9.zip&os=multi" > orientdb-comm
unity-2.0.9.zip
$ unzip orientdb-community-2.0.9.zip
```

Import Beer Categories

These are the first 2 lines of `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" }
       ]
    }
  }
}
```

Now to import it into OrientDB, move into the "bin" directory of OrientDB distribution.

```
$ cd orientdb-community-2.0.9/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 2 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'll 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 import the styles.

```
$ ./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 2 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 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 2 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 `breweries.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: 27 errors are due to the 27 wrong content lines that have no id.

This database is available online. Install it with:

- Studio: in the login page press the "Cloud" button, put server's credential and press on download button on "OpenBeer" line
- Download it manually from http://orientdb.com/public-databases/OpenBeer.zip and unzip it in a OpenBeer folder inside OrientDB's server "databases" directory

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 form 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

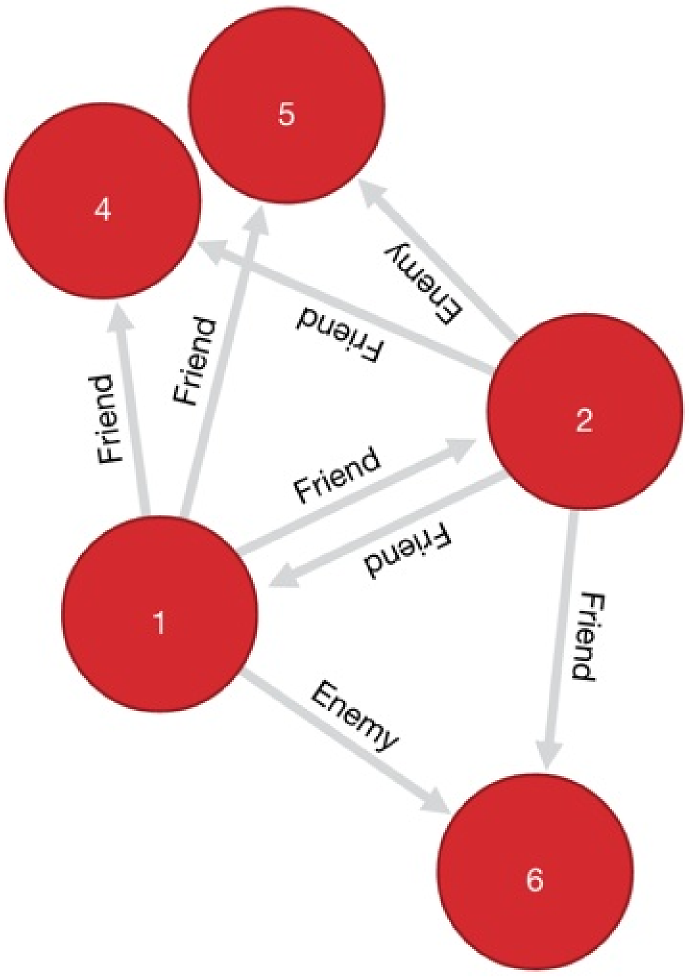
```
        "standardElementConstraints": false,
```

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:


5
4
Enemy
Friend
Friend
Friend
2
Friend
Friend
Friend
1
Enemy
6

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
```

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
```

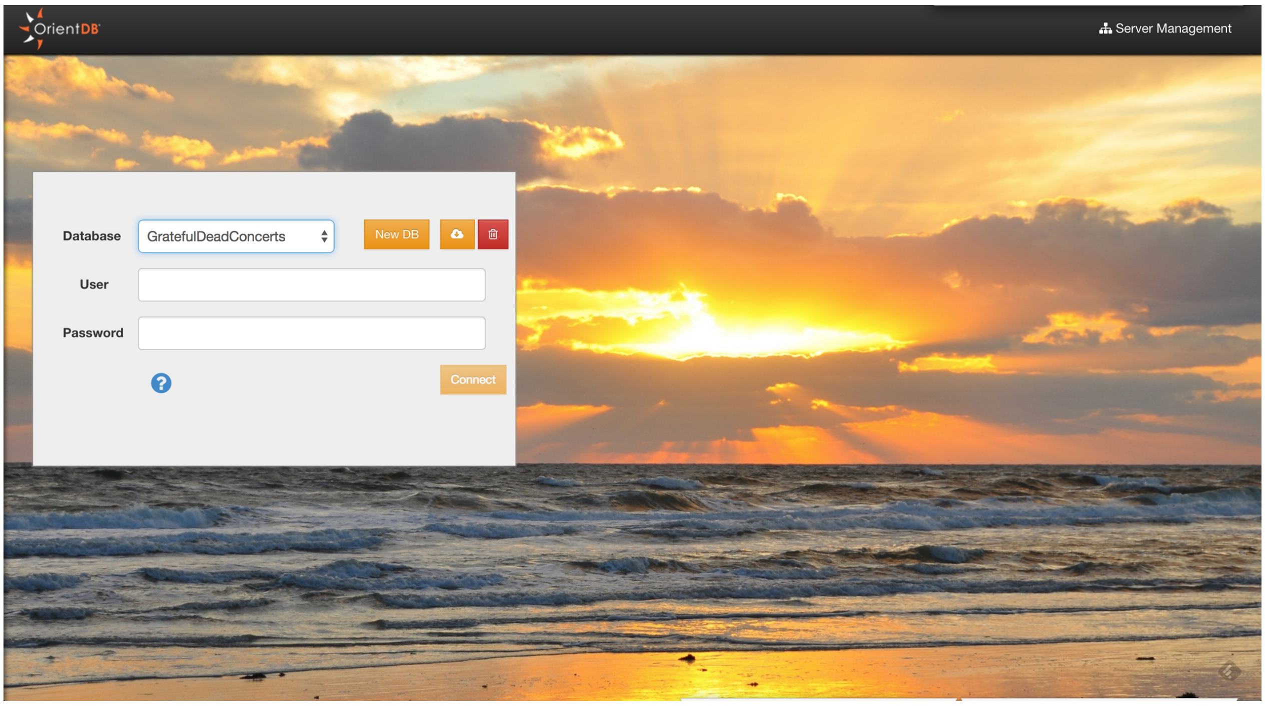
Studio Home page

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 new Studio 2.0 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

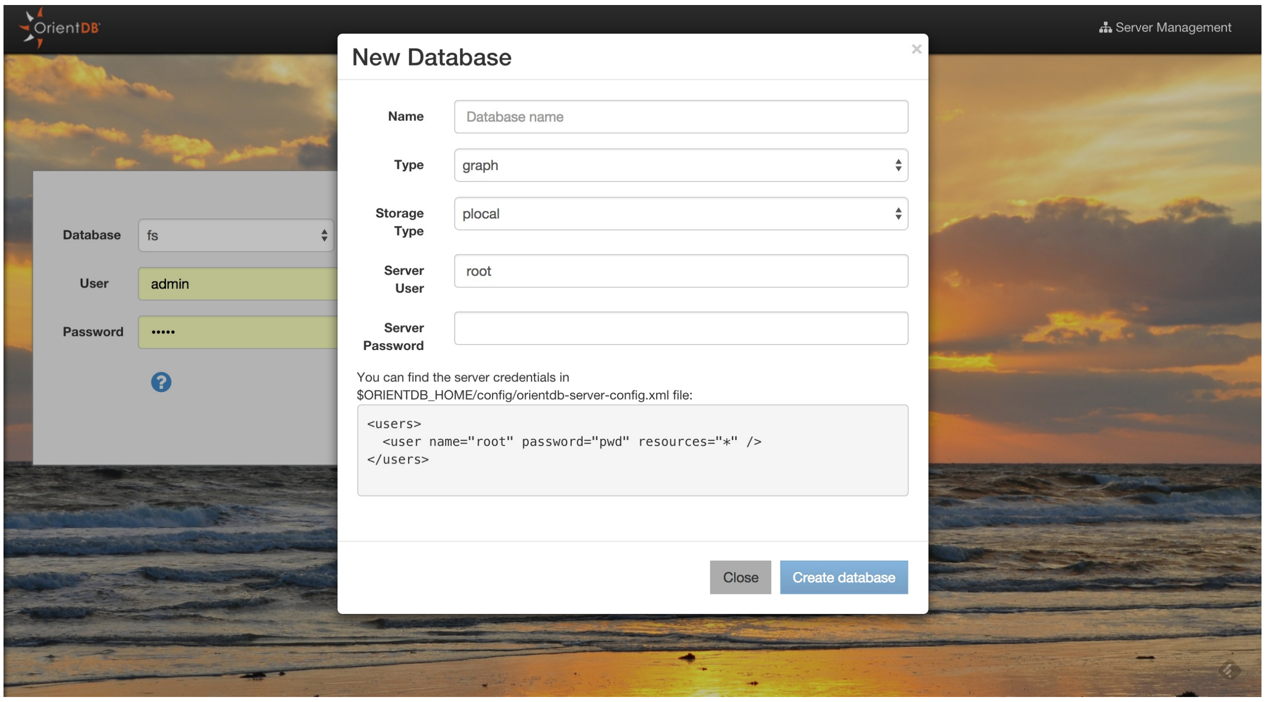
- Server User
- Server Password

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



New Database

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

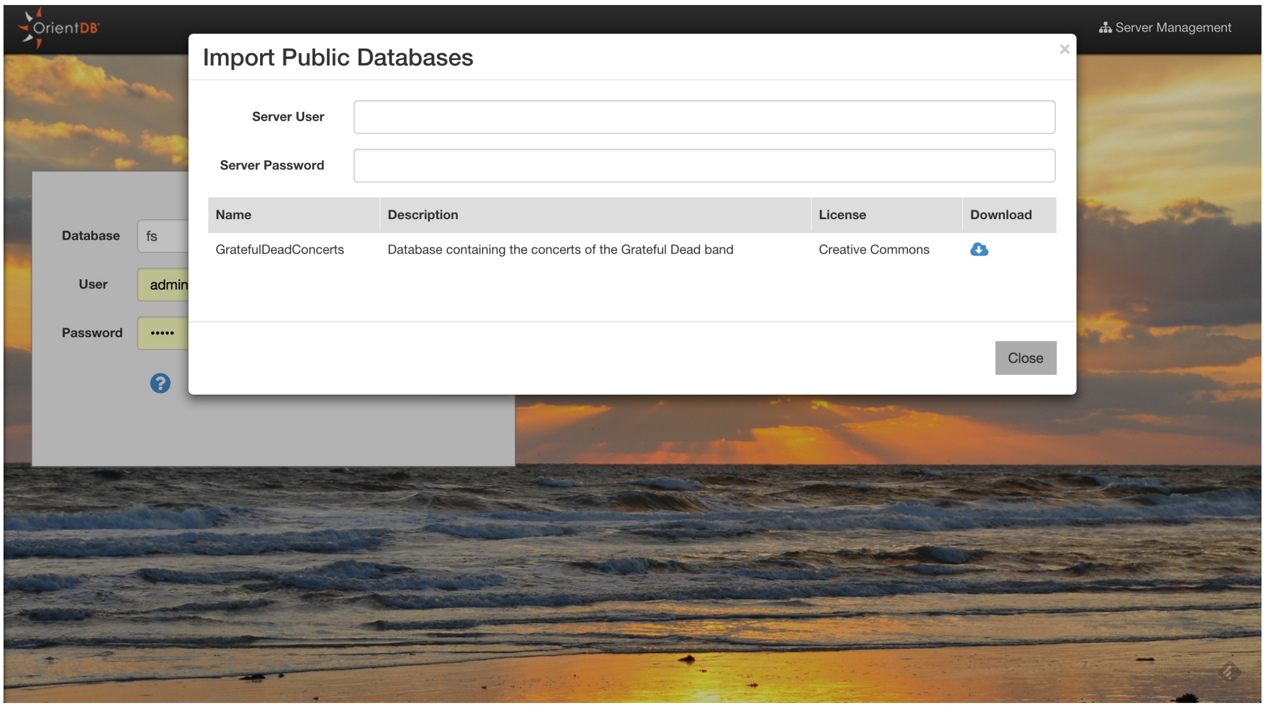
You can find the server credentials in the $ORIENTDB_HOME/config/orientdb-server-config.xml file:

```
<users>
  <user name="root" password="pwd" resources="*" />
</users>
```

Once created, Studio will automatically login to the new database.

Import a public database

Studio 2.0 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.

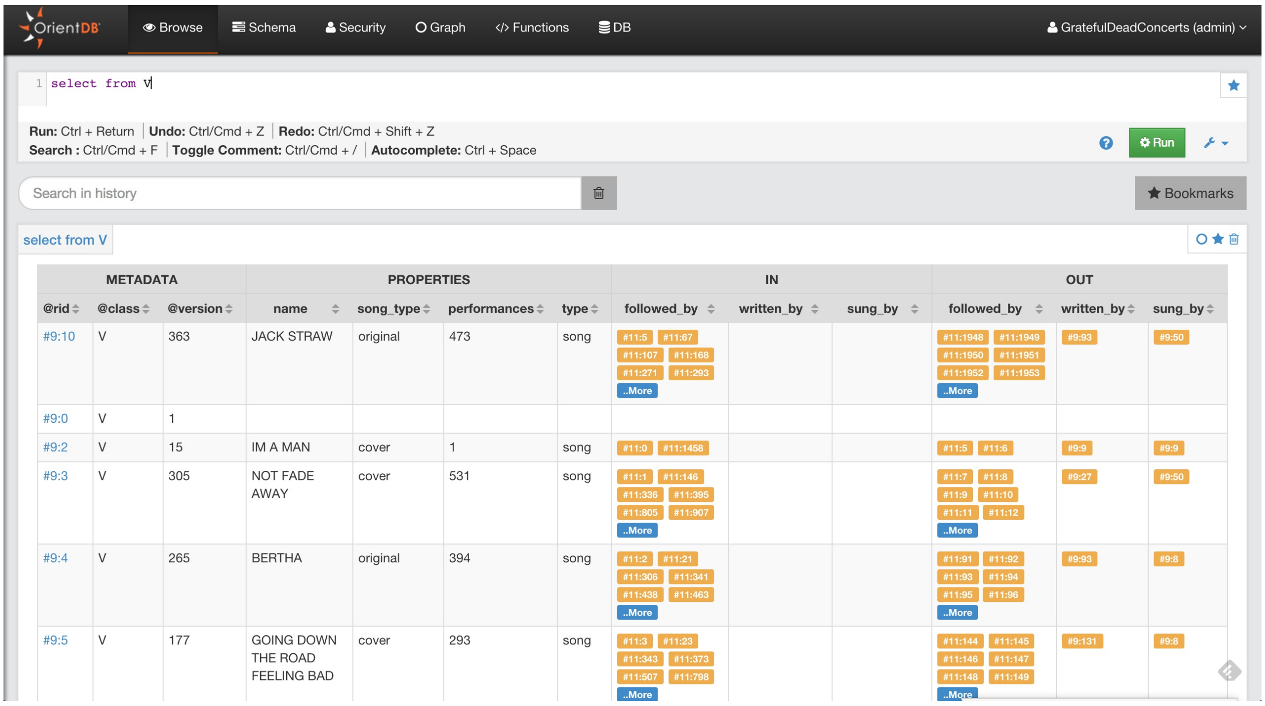
Execute a query

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
```

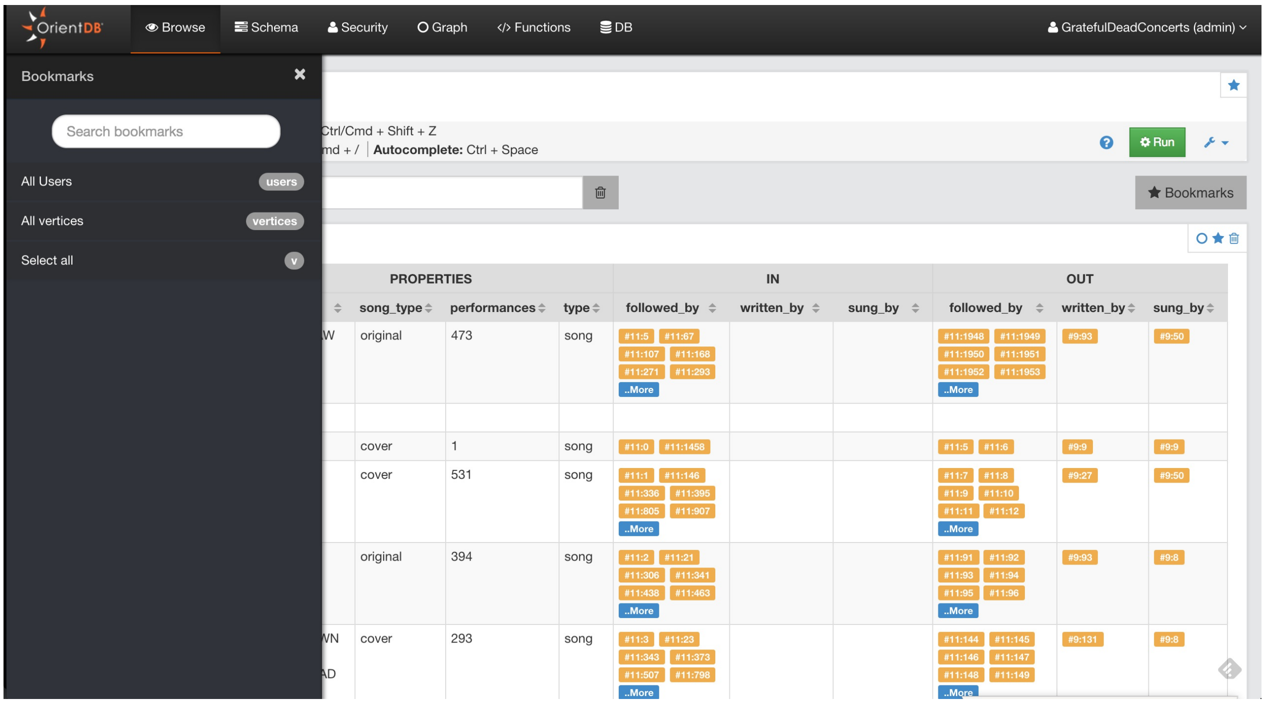
Run: Ctrl + Return | **Undo:** Ctrl/Cmd + Z | **Redo:** Ctrl/Cmd + Shift + Z
Search : Ctrl/Cmd + F | **Toggle Comment:** Ctrl/Cmd + / | **Autocomplete:** Ctrl + Space

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.



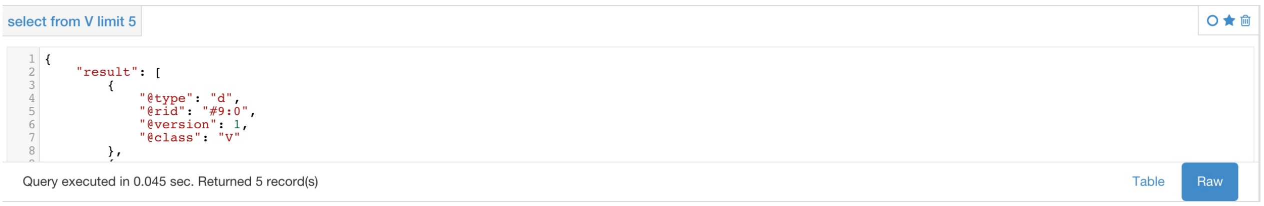
	PROPERTIES			IN			OUT		
	song_type	performances	type	followed_by	written_by	sung_by	followed_by	written_by	sung_by
W	original	473	song	#11:5 #11:67 #11:107 #11:168 #11:271 #11:293 ..More			#11:1948 #11:1949 #11:1950 #11:1951 #11:1952 #11:1953 ..More	#9:93	#9:50
	cover	1	song	#11:0 #11:1458			#11:5 #11:6	#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

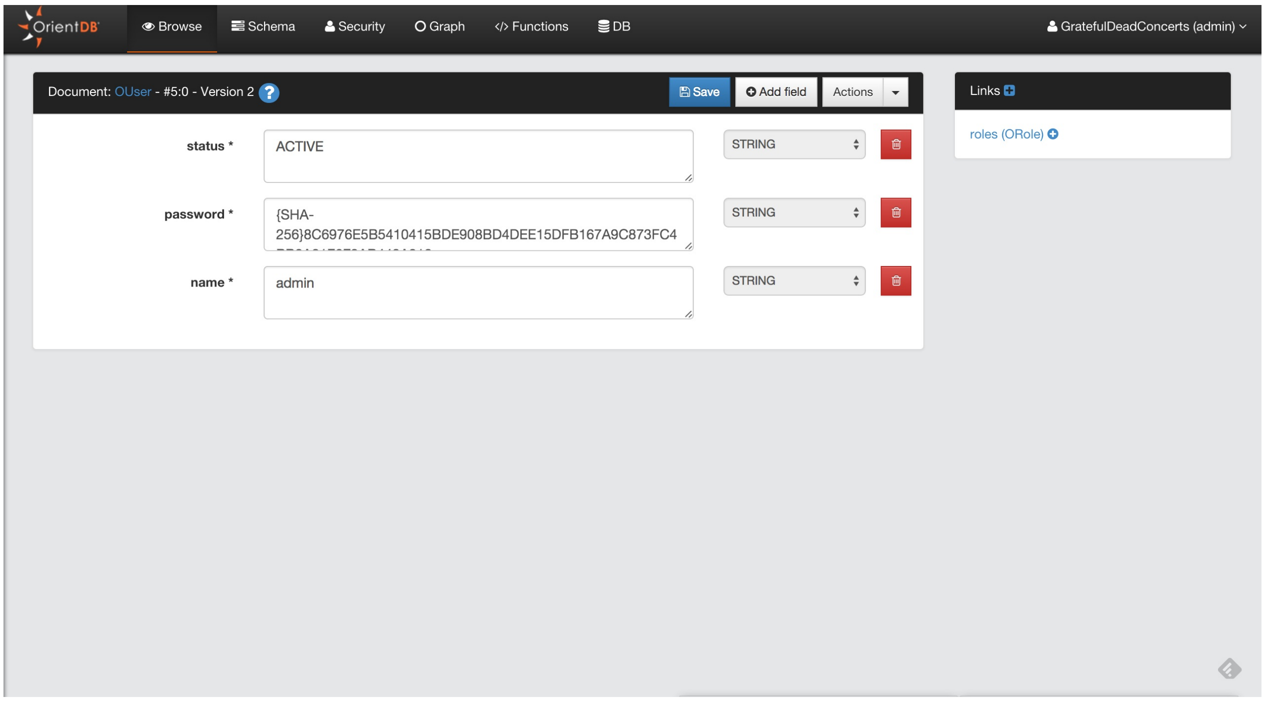
Studio communicates with the OrientDB Server using HTTP/RESt+JSON protocol. To see the output in JSON format, press the **RAW** tab.


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

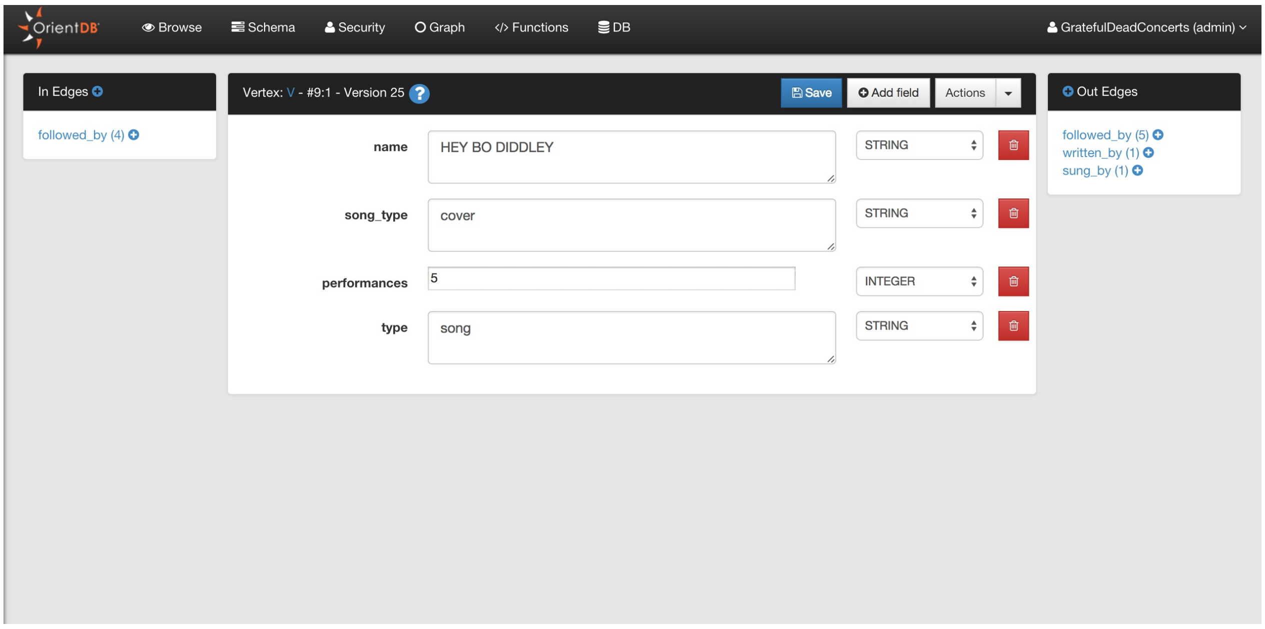


Document: OUser - #5:0 - Version 2

status *	ACTIVE	STRING
password *	{SHA-256}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4	STRING
name *	admin	STRING

roles (ORole)

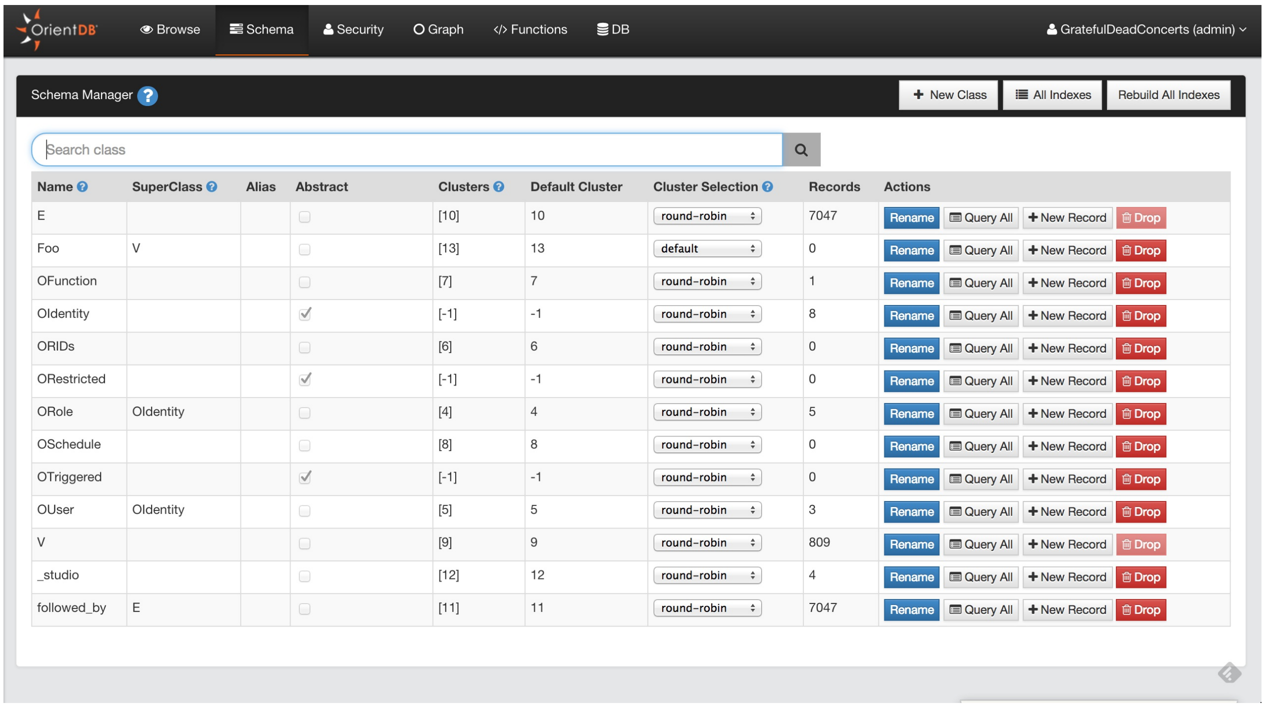
Edit Vertex



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 :

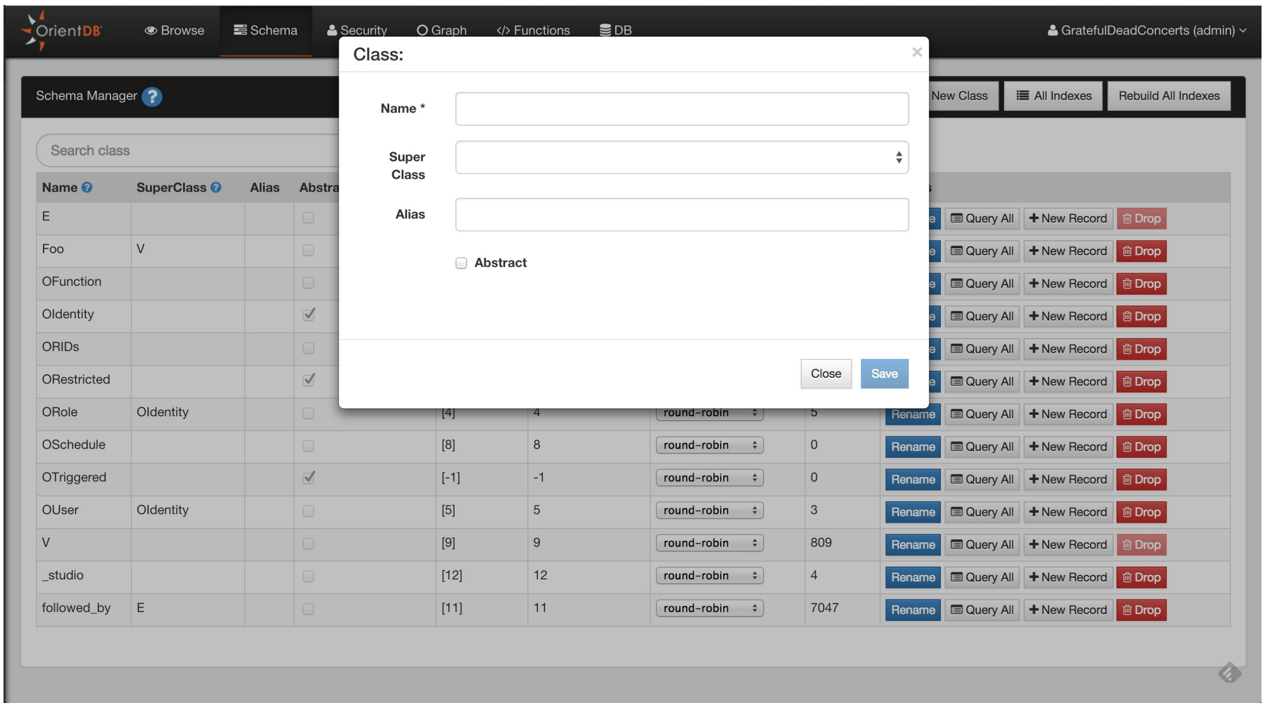
- Browse all the Classes of your database
- Create a new Class
- Rename/Drop a Class
- Change the cluster selection for a Class
- Edit a class by clicking on a class row in the table
- View all indexes created

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

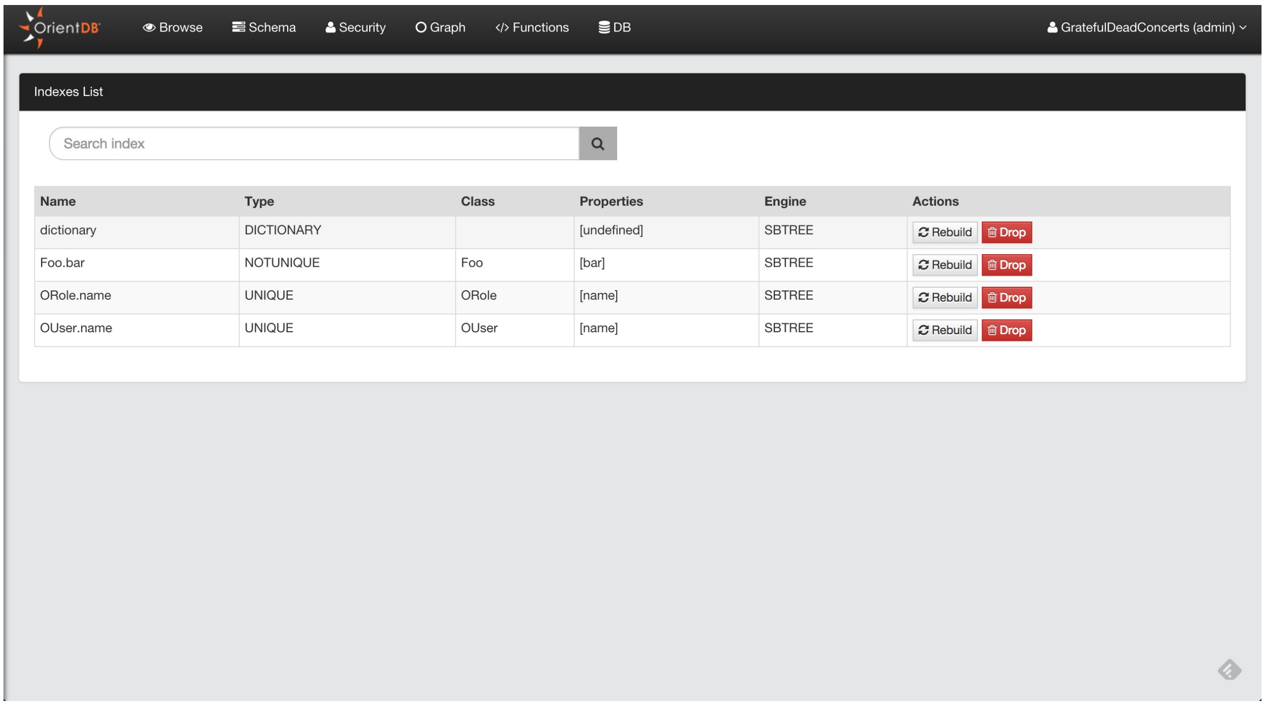
Abstract

Close Save

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

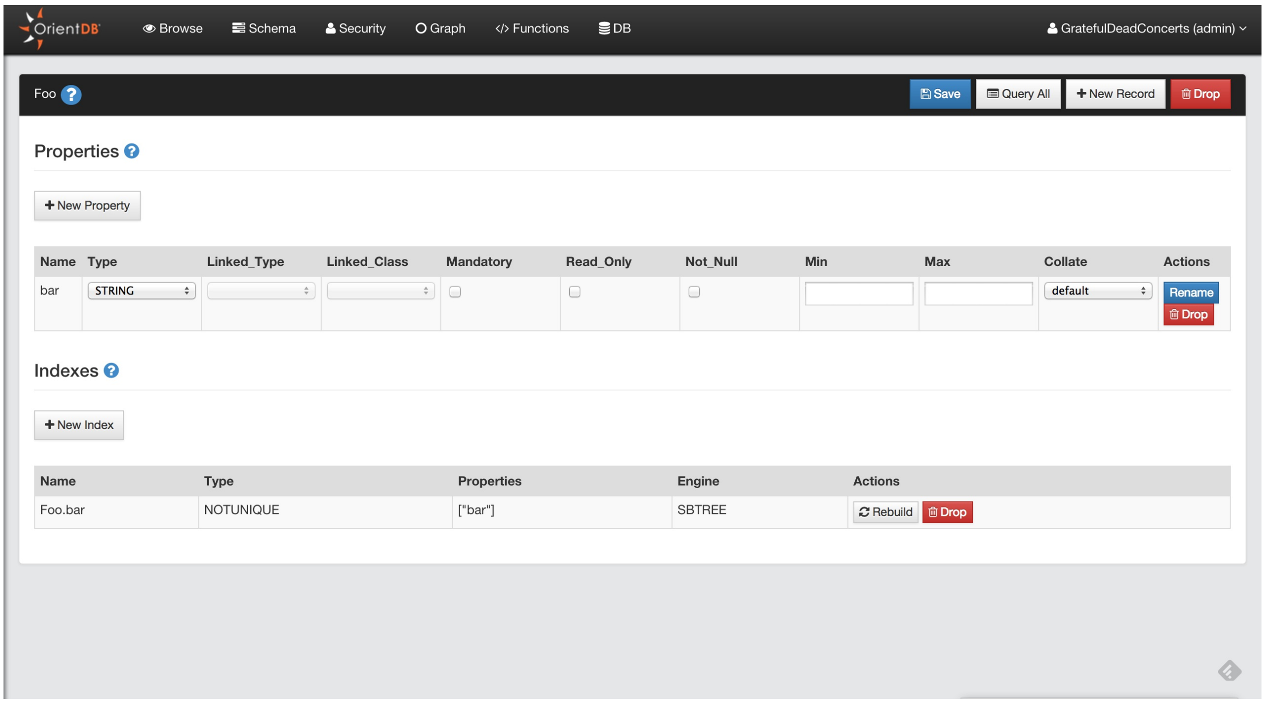
View all indexes

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.



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

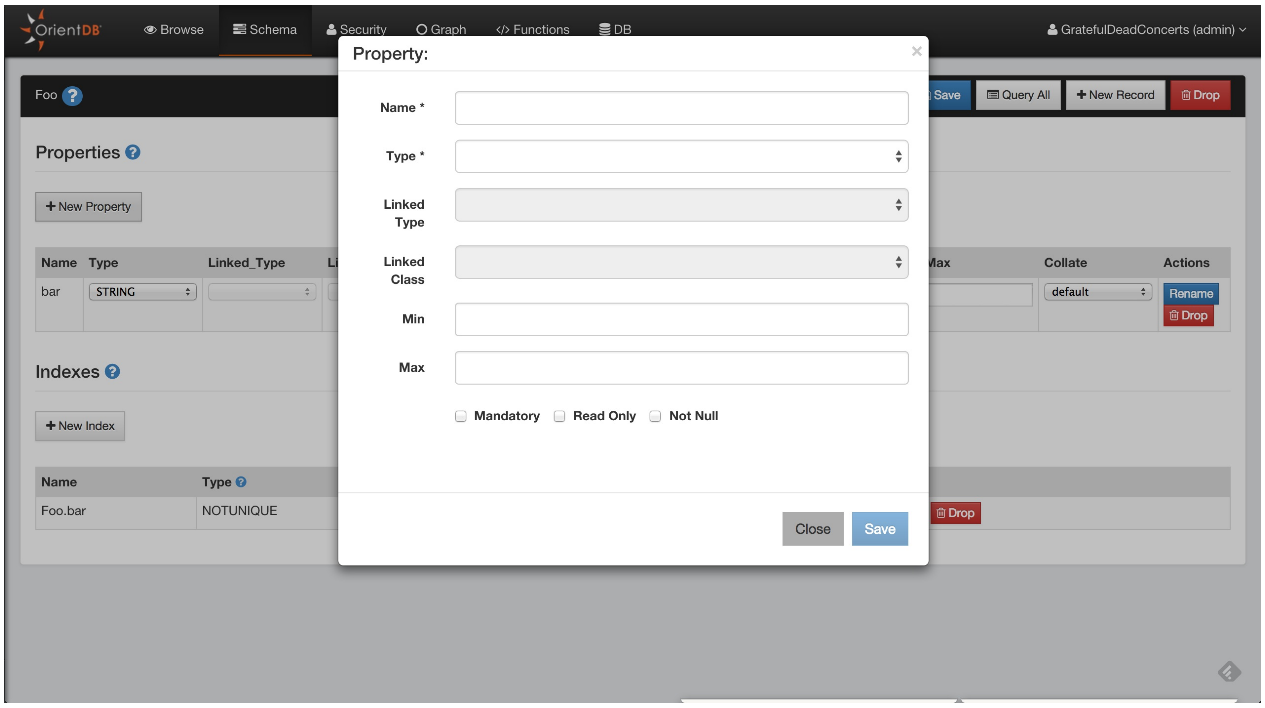


Name	Type	Linked_Type	Linked_Class	Mandatory	Read_Only	Not_Null	Min	Max	Collate	Actions
bar	STRING			☐	☐	☐			default	Rename Drop

Name	Type	Properties	Engine	Actions
Foo.bar	NOTUNIQUE	["bar"]	SBTREE	Rebuild Drop

Property

Add Property



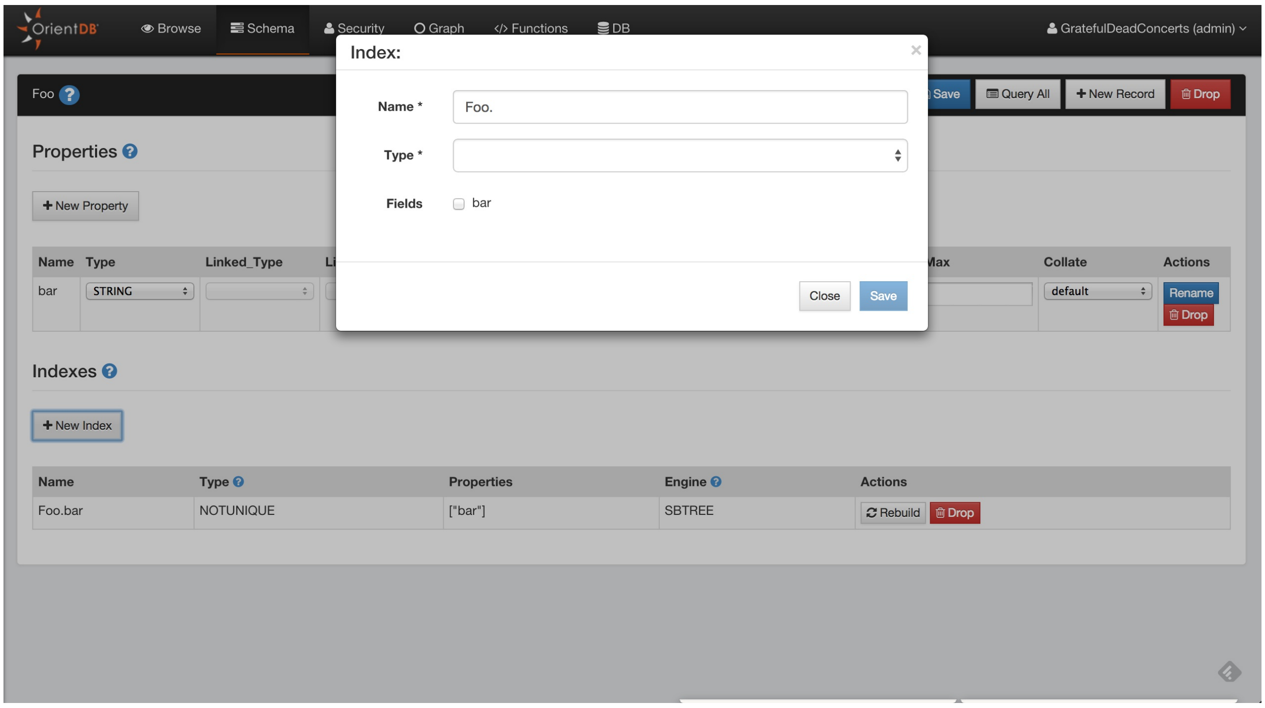
Property:

Name *	
Type *	
Linked Type	
Linked Class	
Min	
Max	

☐ Mandatory ☐ Read Only ☐ Not Null

Indexes

Create new index



Name * Foo.

Type *

Fields ☐ bar

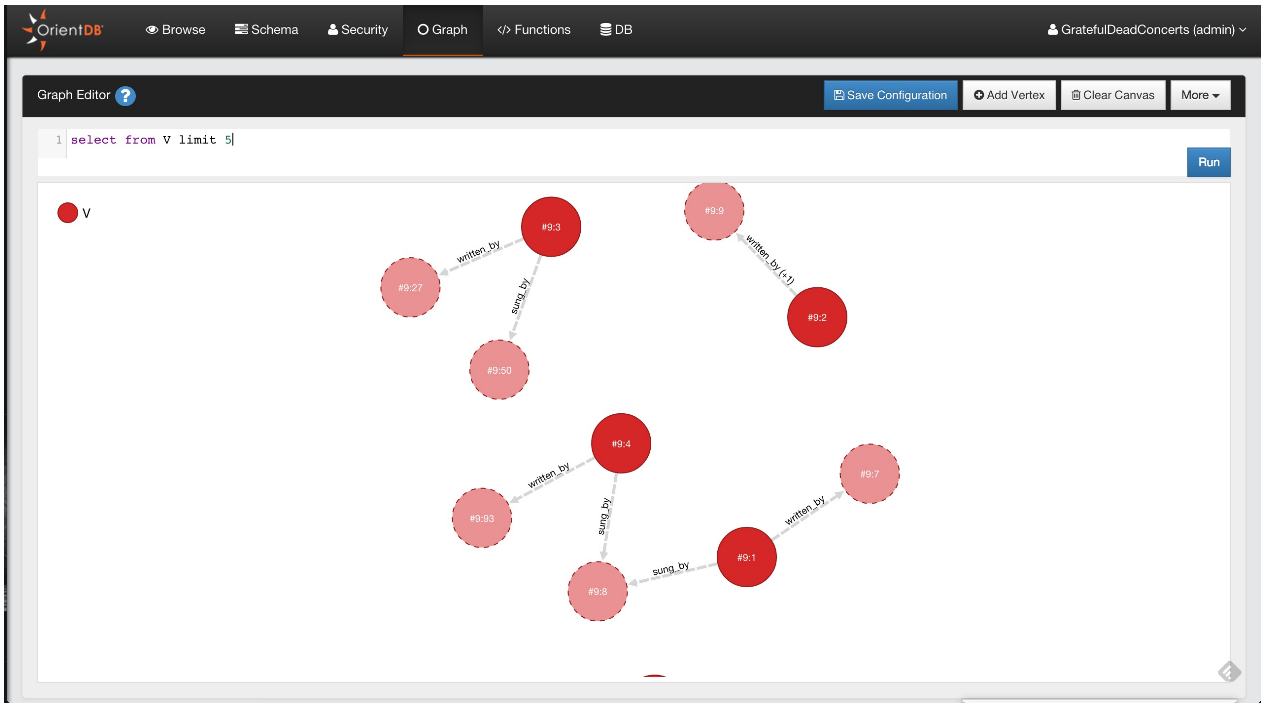
Name	Type	Linked_Type	Max	Collate	Actions
bar	STRING			default	Rename Drop

Name	Type	Properties	Engine	Actions
Foo.bar	NOTUNIQUE	["bar"]	SBTREE	Rebuild Drop

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
```

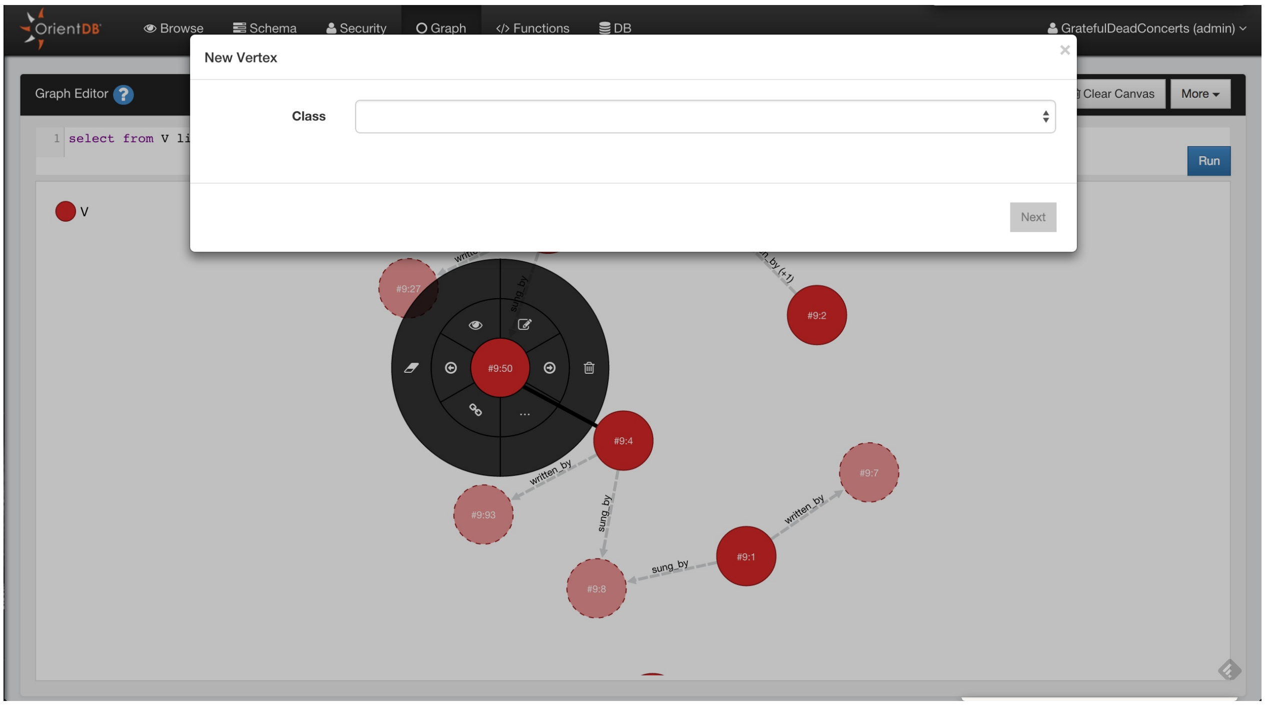
Supported operations in the Graph Editor are:

- 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

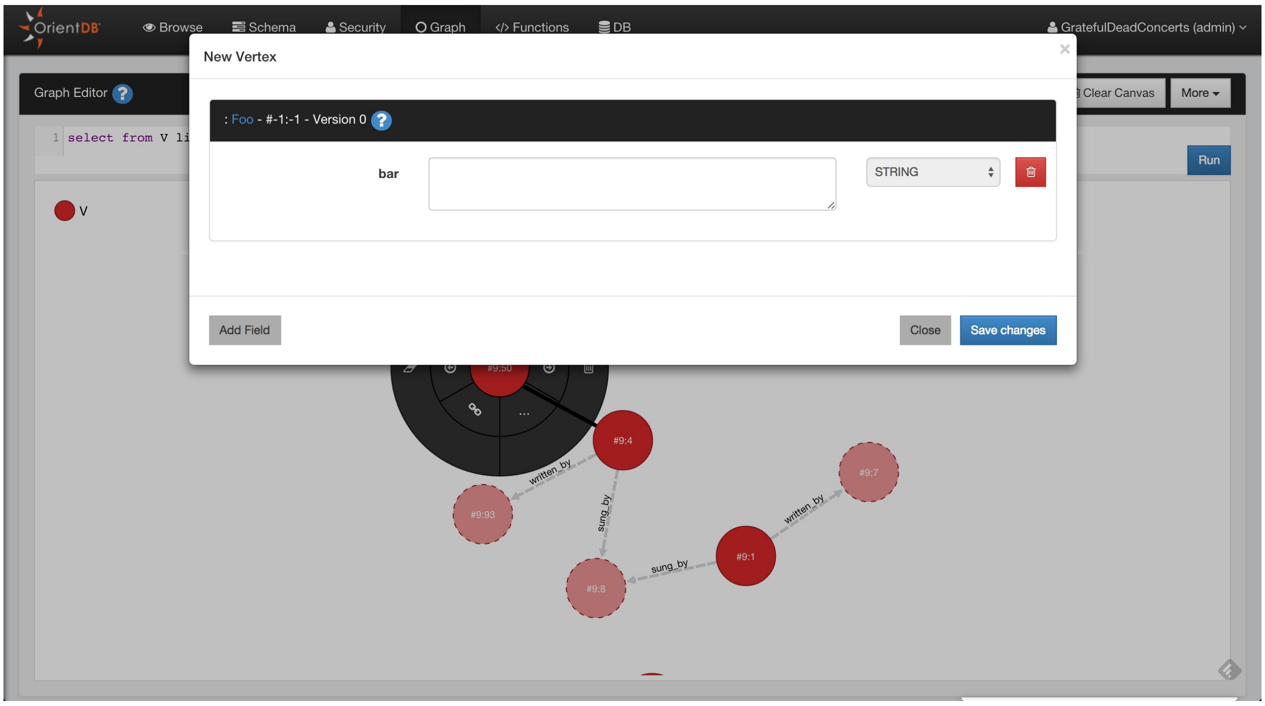
To add a new Vertex in your Graph Database and in the Graph Canvas area you have to press the button **Add Vertex**. This operation is done in two steps.

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



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

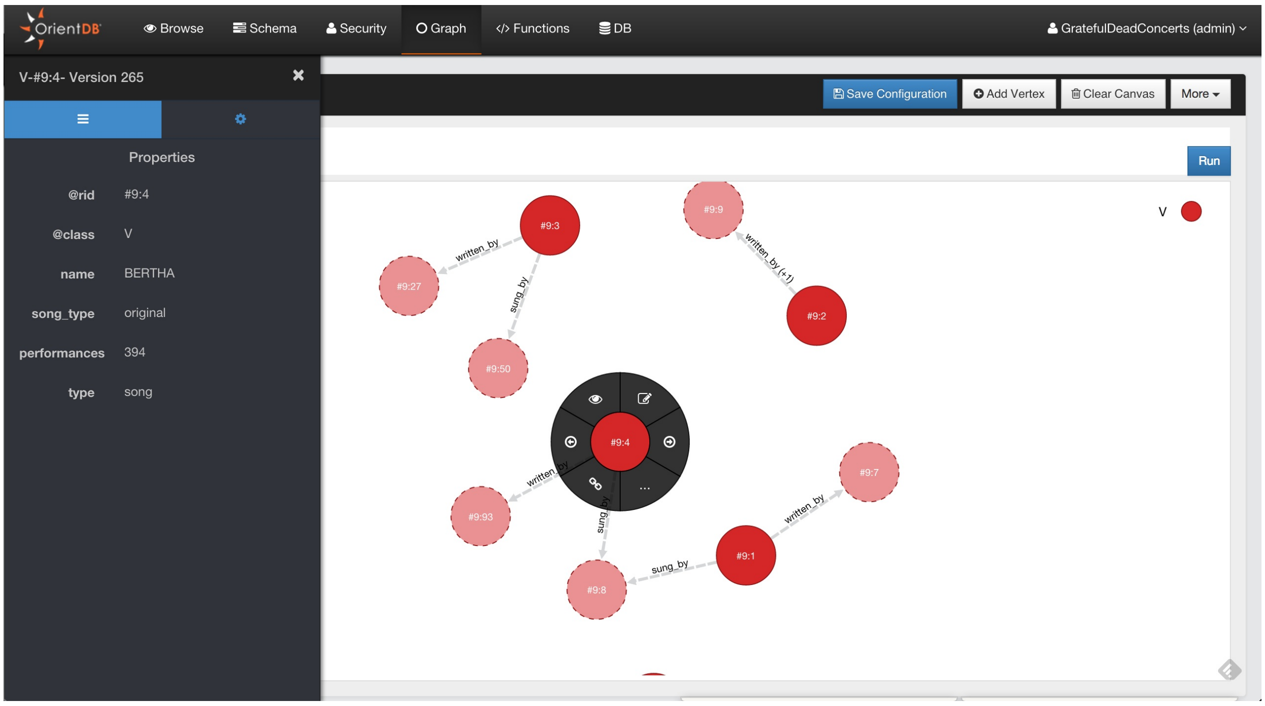
Open the circular menu , open the sub-menu by passing hover the mouse to the menu entry more (...) and then click the **eraser** icon.

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.

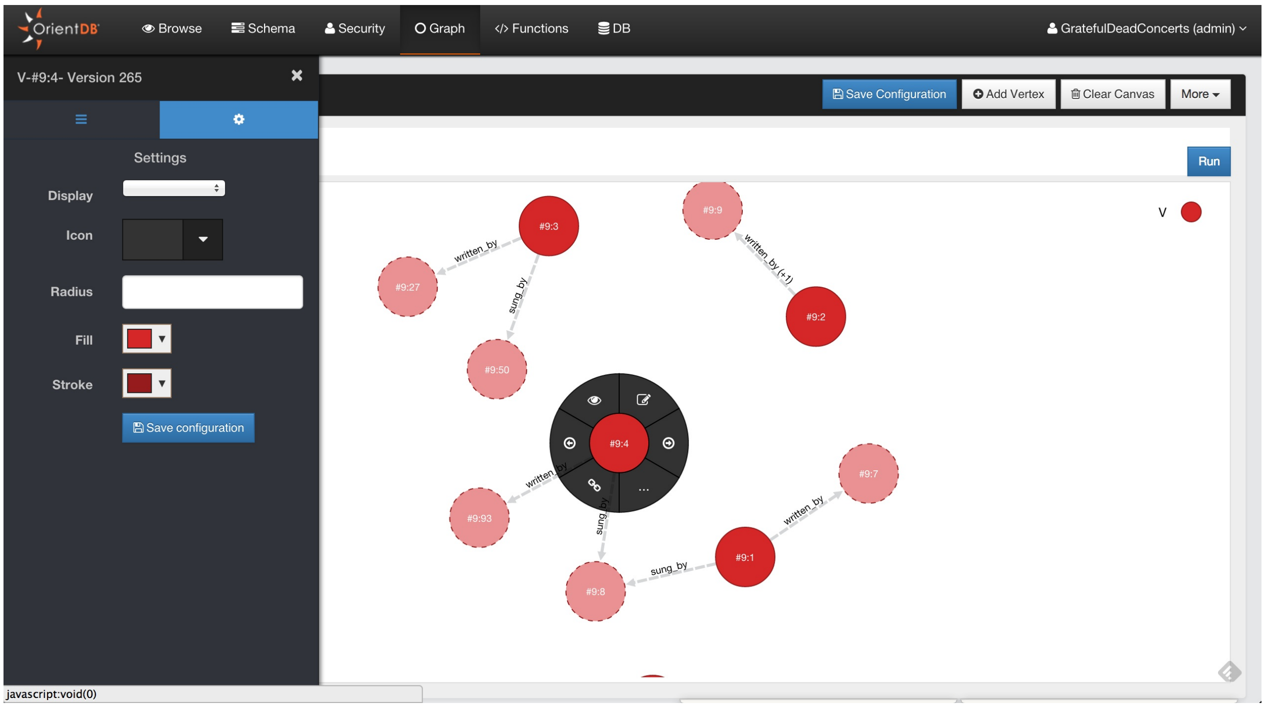
Inspect Vertices

If you want to take a quick look to the Vertex property, click to the **eye** icon.



@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

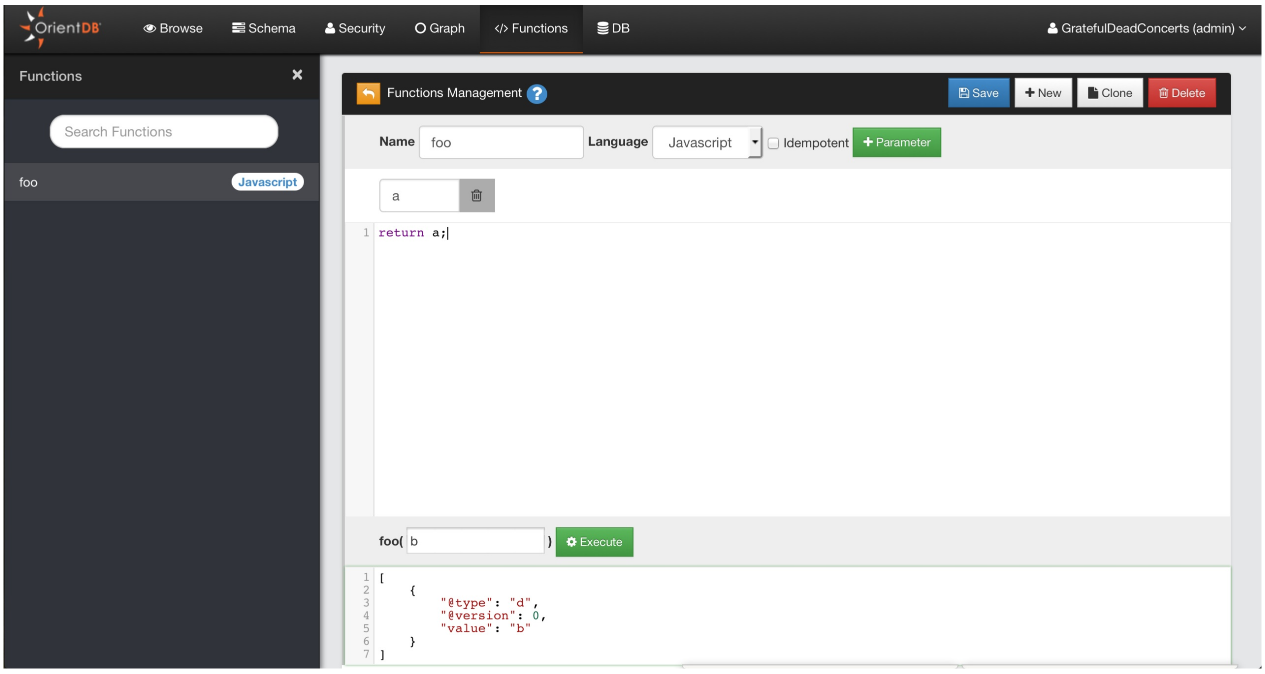
Delete Edges between Vertices

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.



Name foo Language Javascript

a

```
return a;
```

foo(b)

```
[
    {
        "@type": "d",
        "@version": 0,
        "value": "b"
    }
]
```

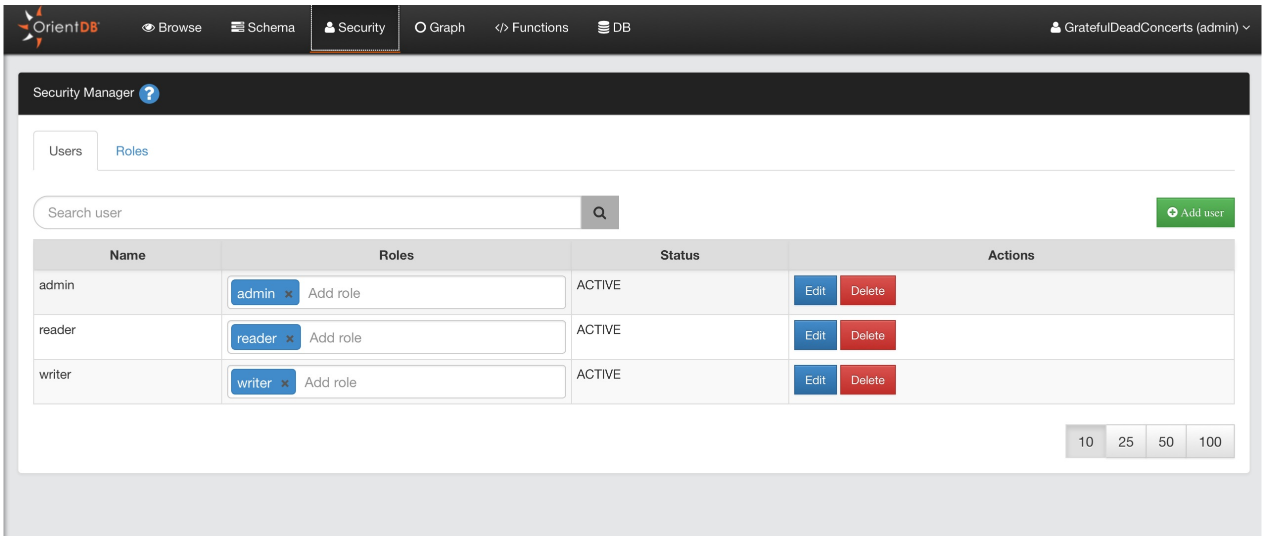
Security

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

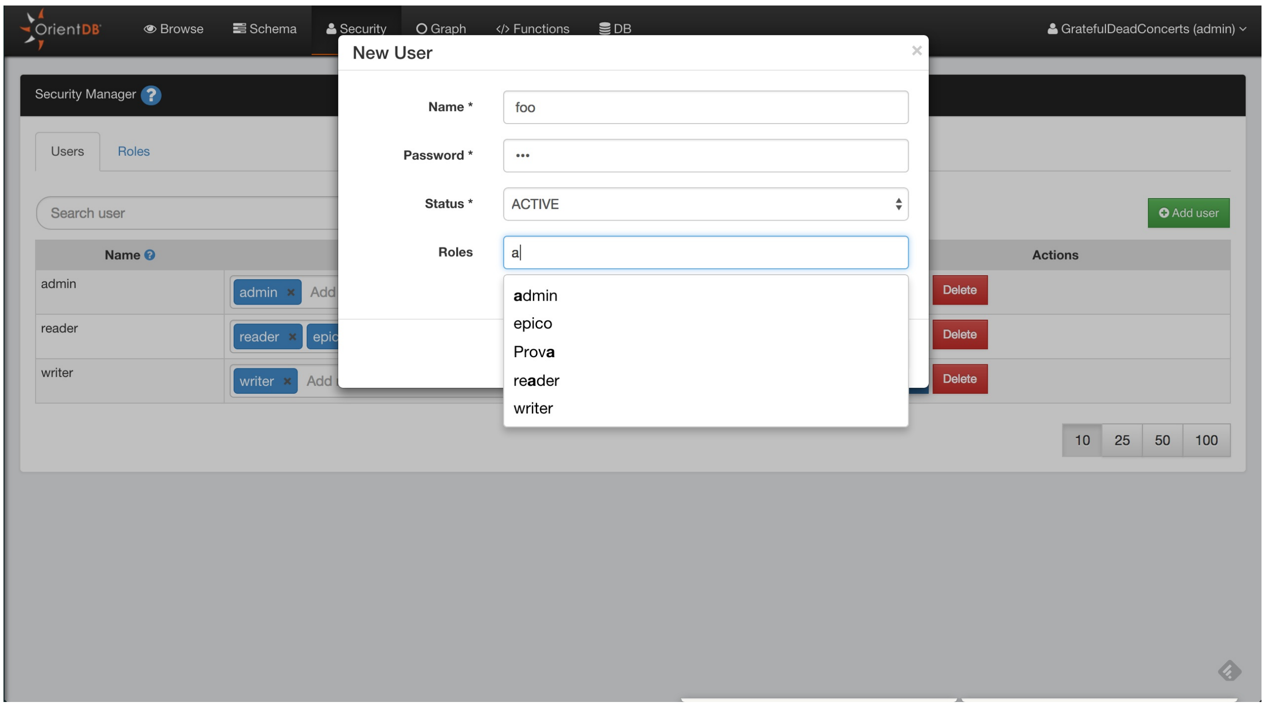


Name	Roles	Status	Actions
admin	admin × Add role	ACTIVE	Edit Delete
reader	reader × Add role	ACTIVE	Edit Delete
writer	writer × Add role	ACTIVE	Edit Delete

10 25 50 100

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.



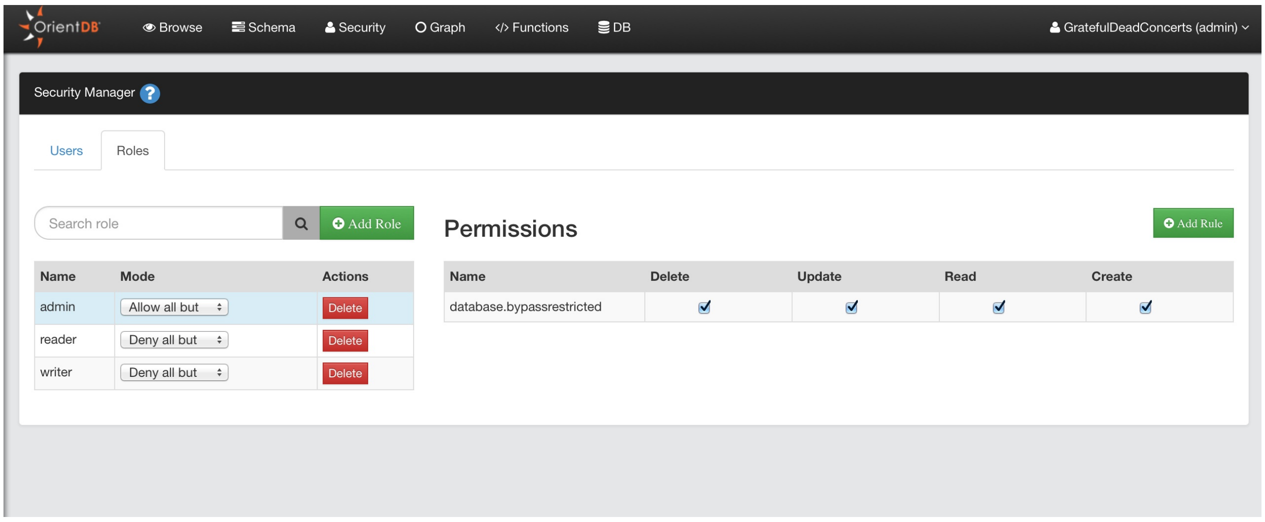
Name * foo
Password * •••
Status * ACTIVE
Roles a

admin
epico
Prova
reader
writer

Roles

Here you can manage the database roles:

- Search Role
- Add Role
- Delete Role
- Edit Role



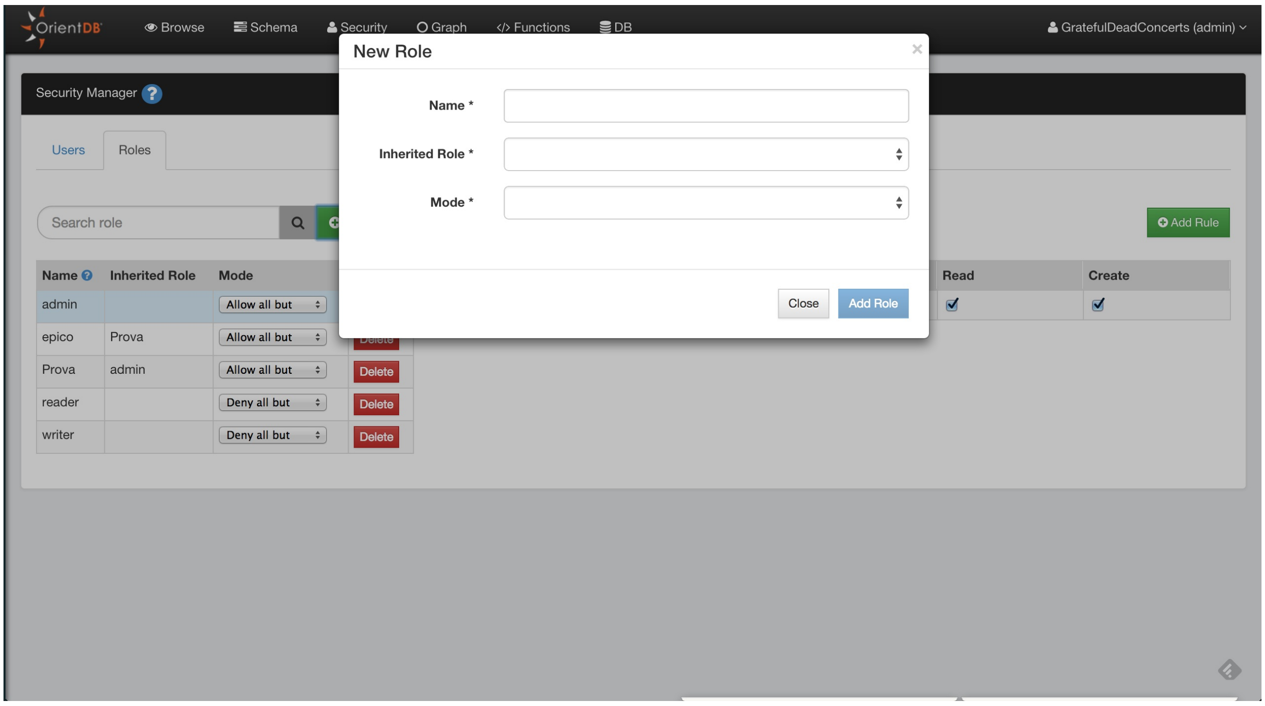
Permissions

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 *	

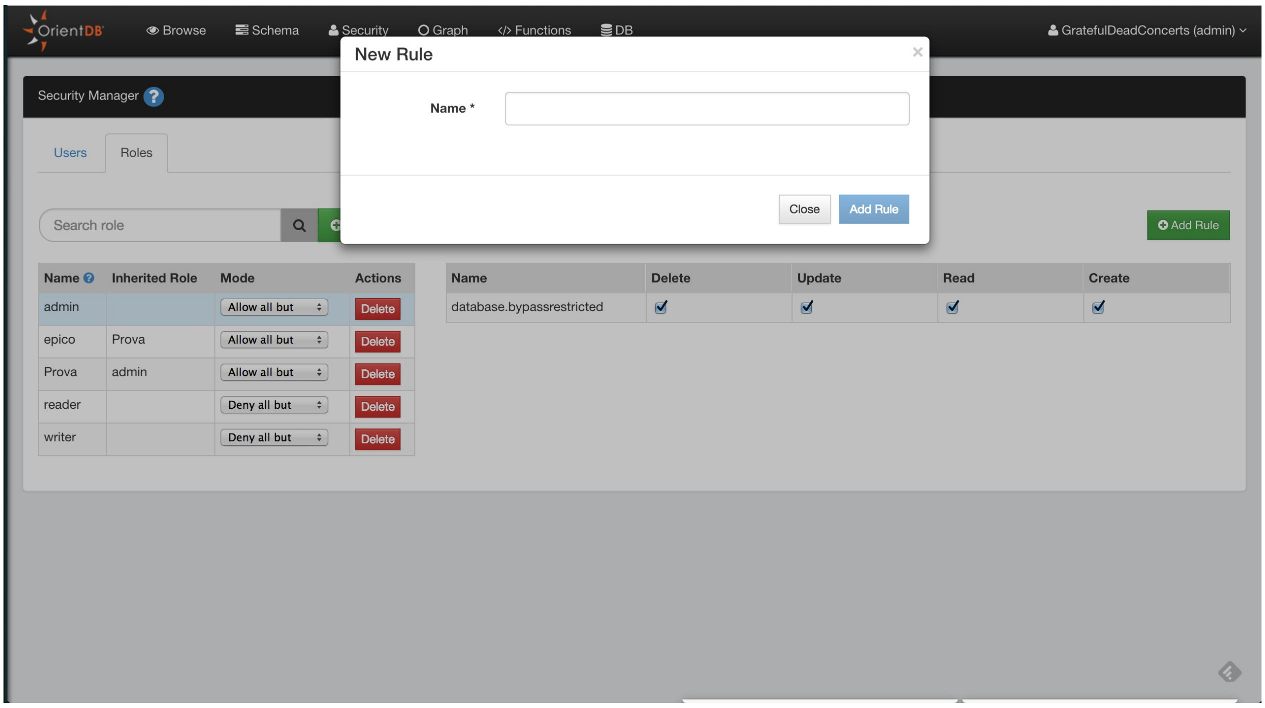
Name	Inherited Role	Mode	
admin		Allow all but	
epico	Prova	Allow all but	Delete
Prova	admin	Allow all but	Delete
reader		Deny all but	Delete
writer		Deny all but	Delete

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 *

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	✓	✓	✓	✓

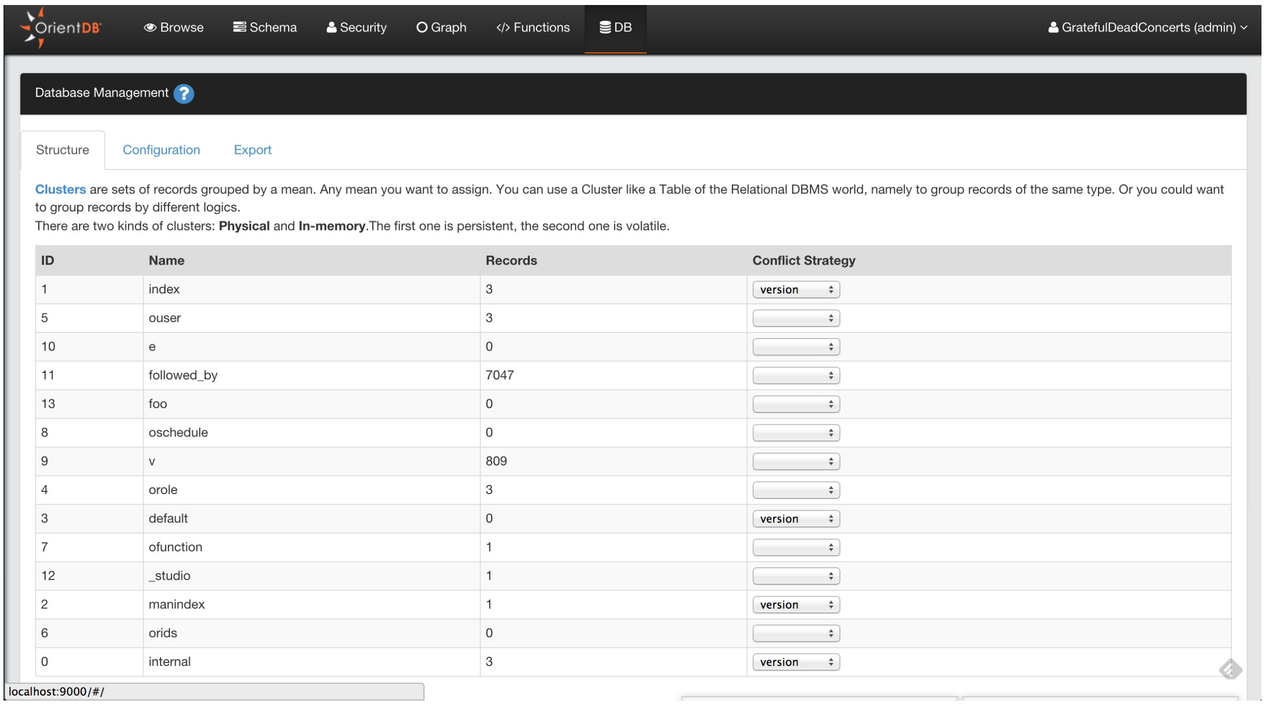
Database Management

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



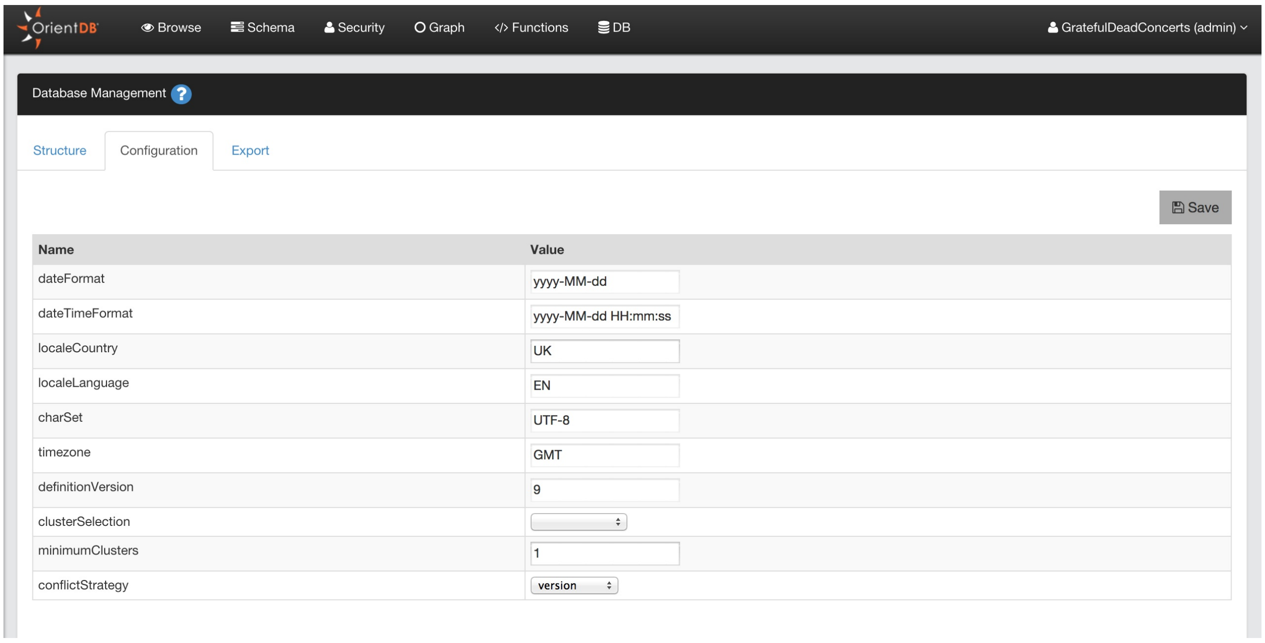
Clusters are sets of records grouped by a mean. Any mean you want to assign. You can use a Cluster like a Table of the Relational DBMS world, namely to group records of the same type. Or you could want to group records by different logics.
There are two kinds of clusters: **Physical** and **In-memory**.The 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:

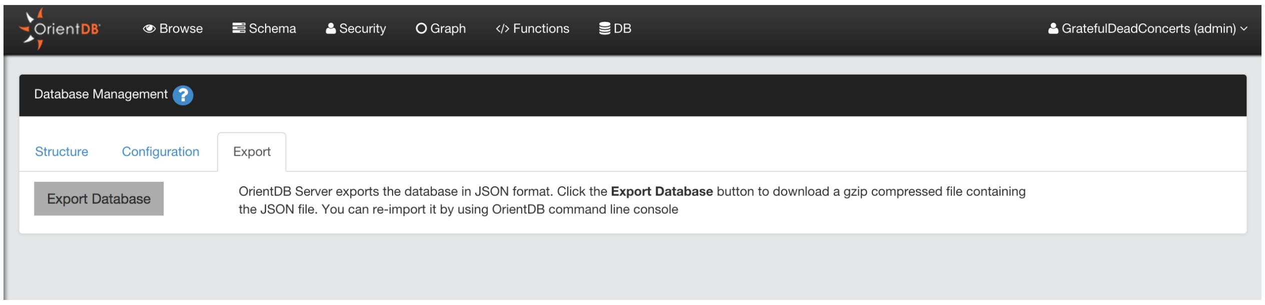
- `dateFormat` , is the date format used in the database by default. Example: yyyy-MM-dd
- `dateTimeFormat` is the datetime format used in the database by default. Example: yyyy-MM-dd HH:mm:ss
- `localeCountry` , is the country used. "NO" means no country set
- `localeLanguage` , is the language used. "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

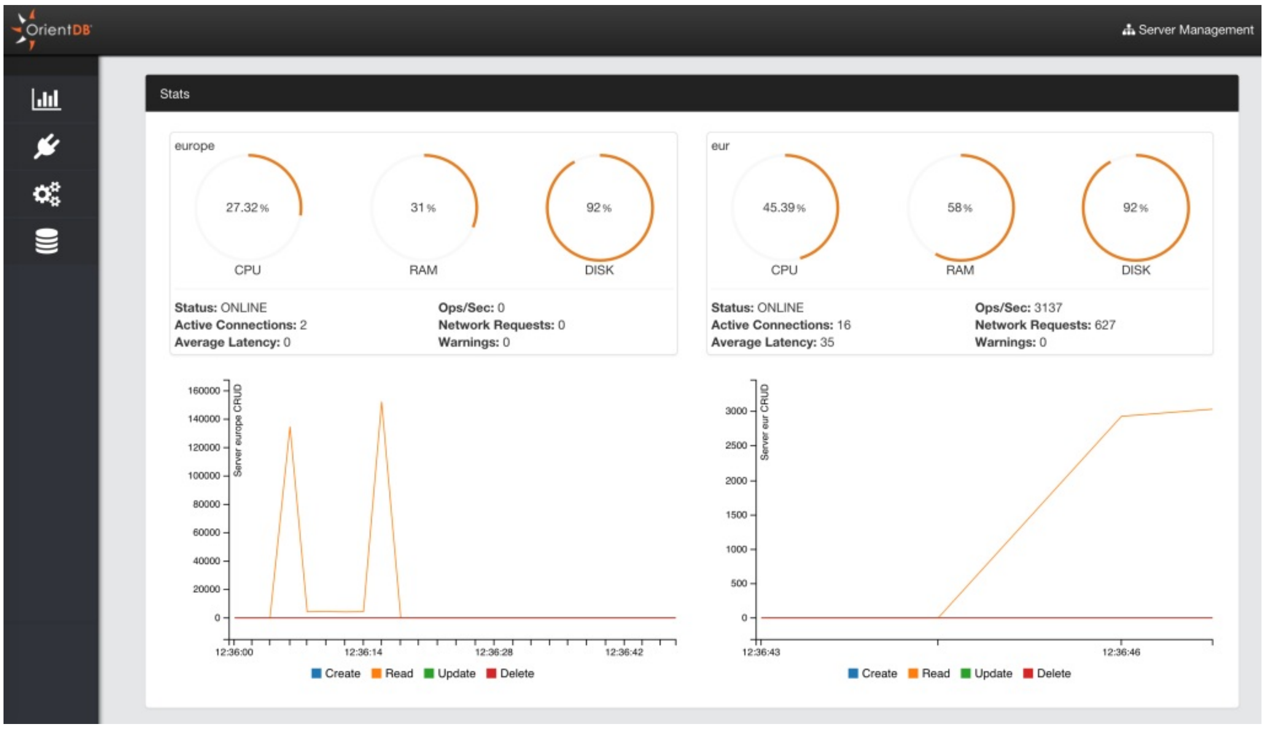
Server Management

This is the section to work with OrientDB Server as DBA/DevOps. Starting from OrientDB 2.1 Studio has been enriched of features taken from the Enterprise Edition.

Statistics

This page summarizes all the most important information about the current server and the other servers connected in cluster if any:

- `Server status`
- `Operations per second`
- `Active Connections`
- `Warnings`
- `CPU` , `RAM` and `DISK` used
- `Live chart` with CRUD operations in real-time



europe

Status: ONLINE
Active Connections: 2
Average Latency: 0
Ops/Sec: 0
Network Requests: 0
Warnings: 0

eur

Status: ONLINE
Active Connections: 16
Average Latency: 35
Ops/Sec: 3137
Network Requests: 627
Warnings: 0

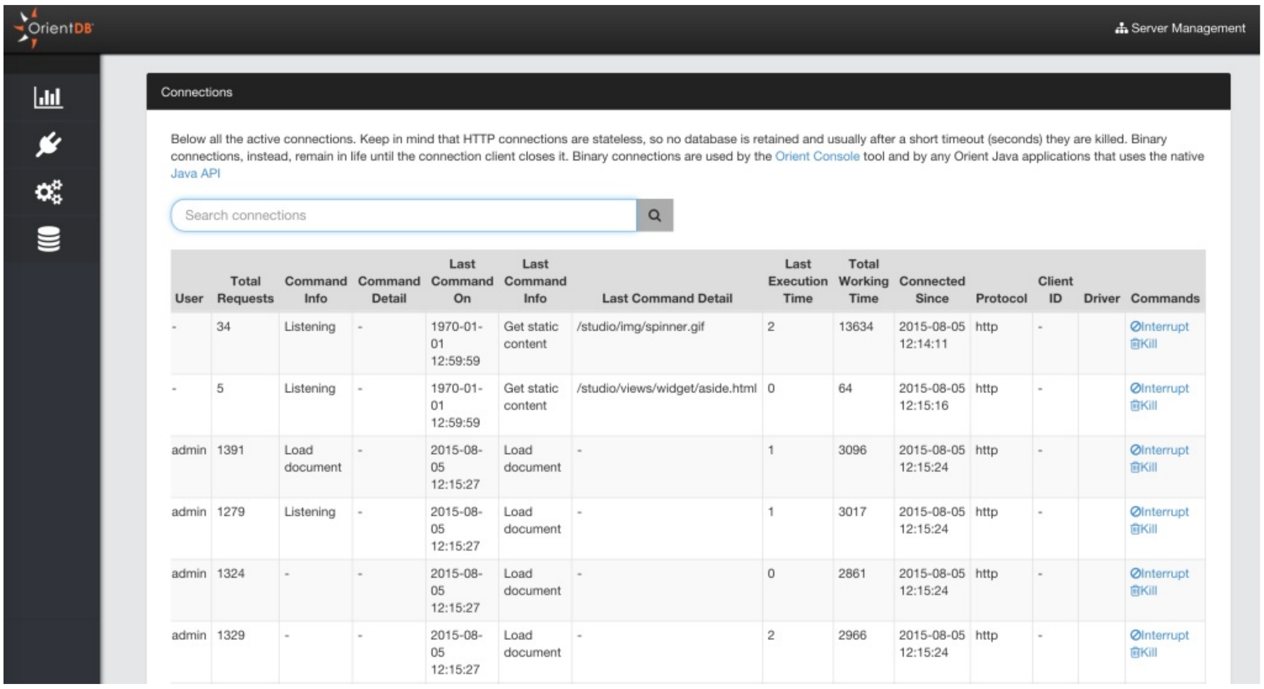
Connections

Displays all the active connections to the server. 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 between HTTP and Binary
- `Client ID` , a text representing the client connection
- `Driver` , the driver name

Each session can be interrupted or even killed.



Connections

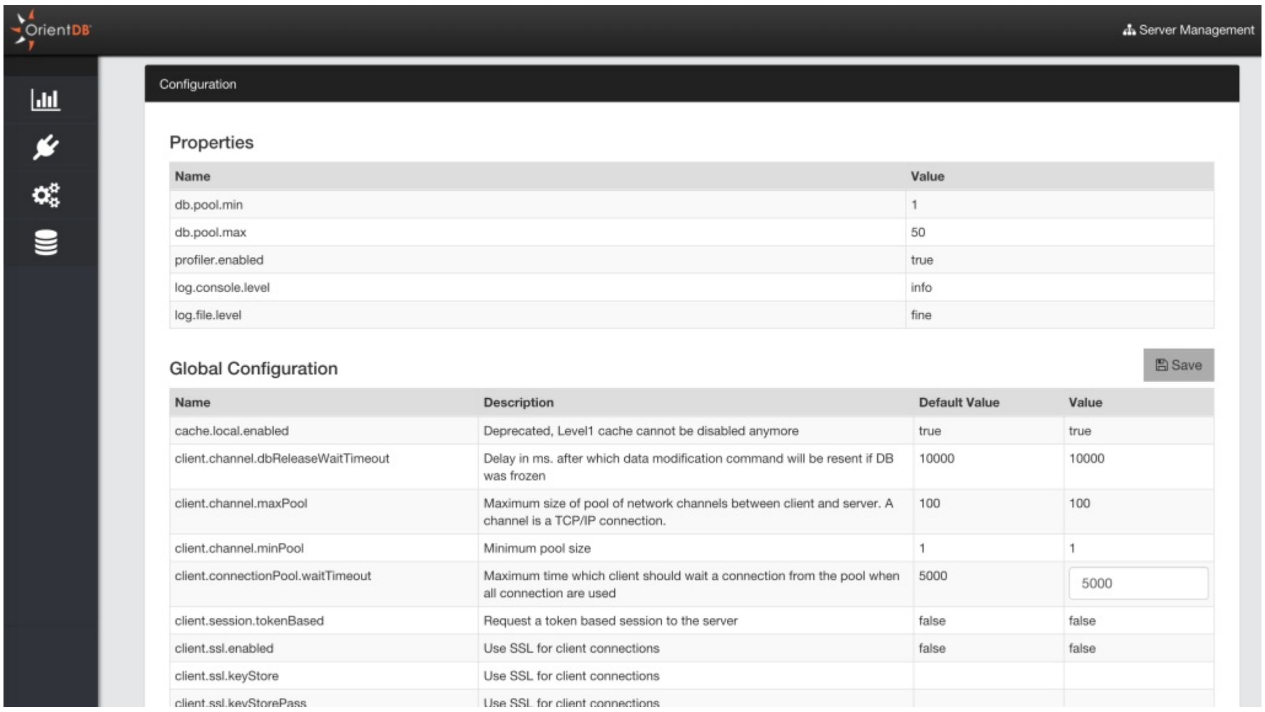
Below all the active connections. Keep in mind that HTTP connections are stateless, so no database is retained and usually after a short timeout (seconds) they are killed. Binary connections, instead, remain in life until the connection client closes it. Binary connections are used by the Orient Console tool and by any Orient Java applications that uses the native Java API

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
-	34	Listening	-	1970-01-01 12:59:59	Get static content	/studio/img/spinner.gif	2	13634	2015-08-05 12:14:11	http	-		Interrupt Kill
-	5	Listening	-	1970-01-01 12:59:59	Get static content	/studio/views/widget/aside.html	0	64	2015-08-05 12:15:16	http	-		Interrupt Kill
admin	1391	Load document	-	2015-08-05 12:15:27	Load document	-	1	3096	2015-08-05 12:15:24	http	-		Interrupt Kill
admin	1279	Listening	-	2015-08-05 12:15:27	Load document	-	1	3017	2015-08-05 12:15:24	http	-		Interrupt Kill
admin	1324	-	-	2015-08-05 12:15:27	Load document	-	0	2861	2015-08-05 12:15:24	http	-		Interrupt Kill
admin	1329	-	-	2015-08-05 12:15:27	Load document	-	2	2966	2015-08-05 12:15:24	http	-		Interrupt Kill

Configuration

This panel shows the Server settings divided in two boxes:

- `Properties` , as the custom settings in `config/orientdb-server-config.xml` file
- `Global Configuration` , as all the global configuration. Only few of them can be changed at run-time with the "Save" button



Configuration

Properties

Name	Value
db.pool.min	1
db.pool.max	50
profiler.enabled	true
log.console.level	info
log.file.level	fine

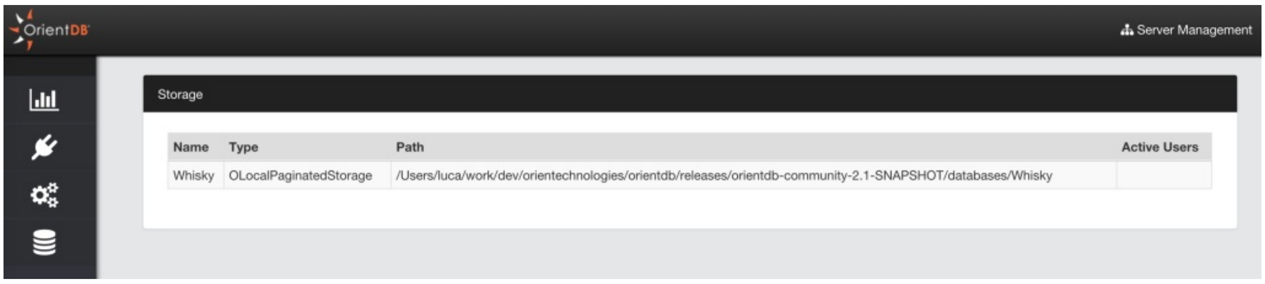
Global Configuration

Name	Description	Default Value	Value
cache.local.enabled	Deprecated, Level1 cache cannot be disabled anymore	true	true
client.channel.dbReleaseWaitTimeout	Delay in ms. after which data modification command will be resent if DB was frozen	10000	10000
client.channel.maxPool	Maximum size of pool of network channels between client and server. A channel is a TCP/IP connection.	100	100
client.channel.minPool	Minimum pool size	1	1
client.connectionPool.waitTimeout	Maximum time which client should wait a connection from the pool when all connection are used	5000	5000
client.session.tokenBased	Request a token based session to the server	false	false
client.ssl.enabled	Use SSL for client connections	false	false
client.ssl.keyStore	Use SSL for client connections		
client.ssl.keyStorePass	Use SSL for client connections		

Storages

This panel shows the storages used by the server. Below the information reported per storage:

- `Name` , is the storage name
- `Type` , where `OLocalPaginatedStorage` (plocal) means persstent and `ODirectMemoryStorage` (memory) is in memory only
- `Path` , as the path on server's file system where the storage is located
- `Active Users` . This infomation couldn't be updated with the real number of users that are using the database

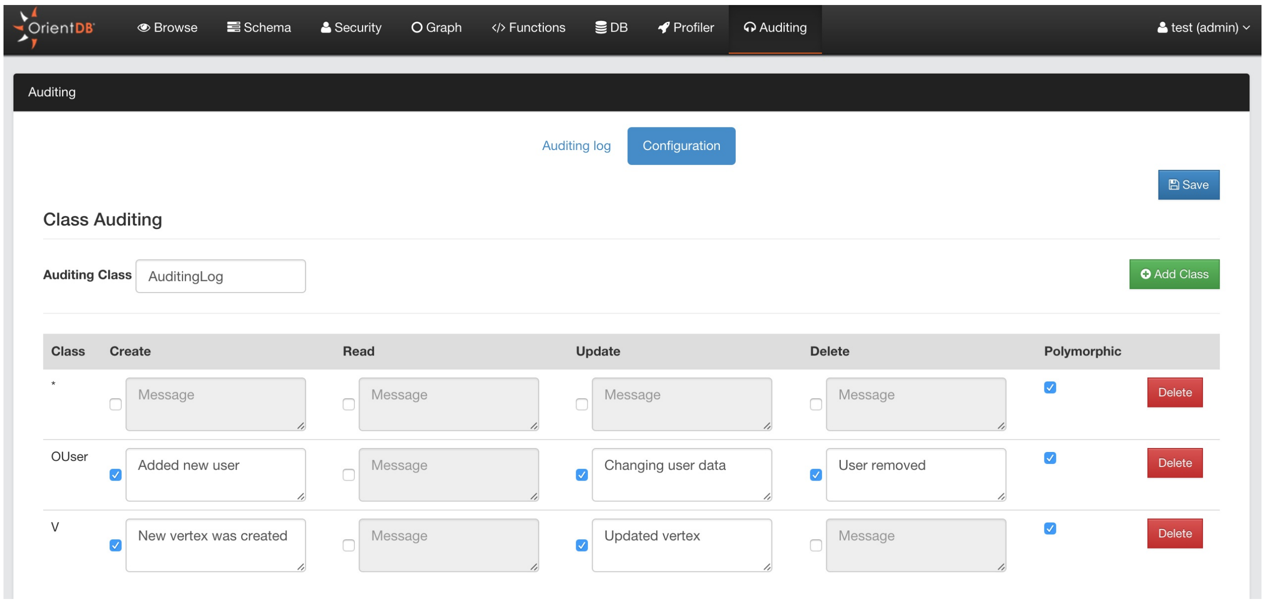


Name	Type	Path	Active Users
Whisky	OLocalPaginatedStorage	/Users/luca/work/dev/orientechnologies/orientdb/releases/orientdb-community-2.1-SNAPSHOT/databases/Whisky	

Auditing (Enterprise only)

Studio 2.1 includes a new functionality called Auditing. To understand how Auditing works, please read the Auditing page.

The Studio Auditing panel helps with configuring auditing by avoiding editing the `auditing-config.json` file manually.



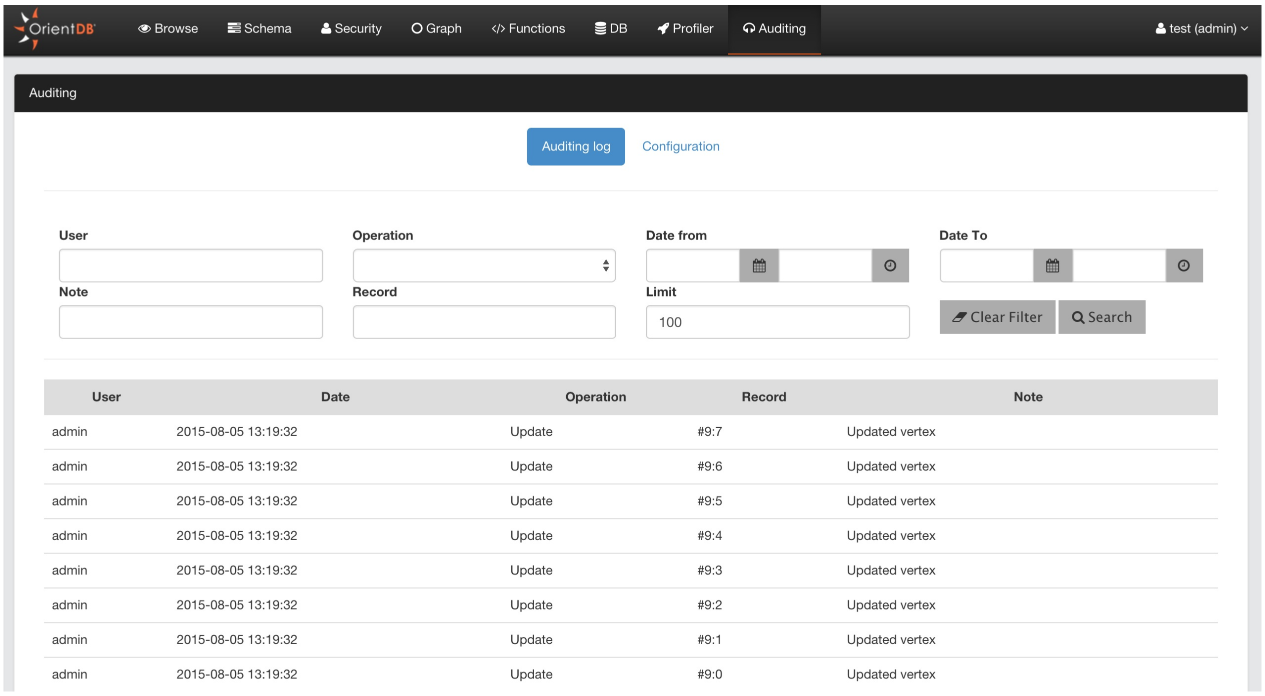
Class Auditing

Auditing Class: AuditingLog

Class		Create		Read		Update		Delete	Polymorphic
*	☐	Message	☐	Message	☐	Message	☐	Message	✓
OUser	✓	Added new user	☐	Message	✓	Changing user data	✓	User removed	✓
V	✓	New vertex was created	☐	Message	✓	Updated vertex	☐	Message	✓

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 last 20 logs: `select from AuditingLog order by @rid desc limit 20`.

However, Studio provides a panel to filter the Auditing Log messages without using SQL.



Limit: 100

User	Date	Operation	Record	Note
admin	2015-08-05 13:19:32	Update	#9:7	Updated vertex
admin	2015-08-05 13:19:32	Update	#9:6	Updated vertex
admin	2015-08-05 13:19:32	Update	#9:5	Updated vertex
admin	2015-08-05 13:19:32	Update	#9:4	Updated vertex
admin	2015-08-05 13:19:32	Update	#9:3	Updated vertex
admin	2015-08-05 13:19:32	Update	#9:2	Updated vertex
admin	2015-08-05 13:19:32	Update	#9:1	Updated vertex
admin	2015-08-05 13:19:32	Update	#9:0	Updated vertex

Troubleshooting

This page aims to link all the guides to Problems and Troubleshooting.

Sub sections

- Troubleshooting Java API

Topics

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 i 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:

```
select v.name, sum(count) as cnt from (
  select if(eval("in=#17:0"),out,in) as v,count from E where (
    in=#17:0 or out=#17:0)
  ) group by v order by cnt desc
```

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: http://code.google.com/p/orient/source/browse/trunk/tests/src/test/java/com/orientechnologies/orient/test/database/auto/SQLSelectIndexReuseTest.java

Avoid use of @rid in WHERE conditions (not actual from 1.3 version)

Using **@rid** in where conditions slow down queries. Much better to use the RecordID as target. Example:

Change this:

```
SELECT FROM Profile WHERE @rid = #10:44
```

With this:

```
SELECT FROM #10:44
```

Also

```
SELECT FROM Profile WHERE @rid IN [#10:44, #10:45]
```

With this:

```
SELECT FROM [#10:44, #10:45]
```

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` .

Environment

environment.dumpCfgAtStartup

Dumps the configuration during application startup..

```
Setting name...: environment.dumpCfgAtStartup
Default value..: false
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

environment.allowJVMShutdown

Allows the shutdown of the JVM, if needed/requested..

```
Setting name...: environment.allowJVMShutdown
Default value..: true
Set at run-time: true
Hidden.........: false
```

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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

memory.directMemory.trackMode

If 'track mode' is switched on, then the following steps are performed: 1. direct memory JMX bean is registered. 2. You may check amount of allocated direct memory as a property of the JMX bean. 3. If a memory leak is detected, then a JMX event will be fired. This mode causes a large overhead and should be used for testing purposes only..

```
Setting name...: memory.directMemory.trackMode
Default value..: false
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

Storage

storage.diskCache.bufferSize

Size of disk buffer in megabytes..

```
Setting name...: storage.diskCache.bufferSize
Default value..: 4096
Set at run-time: false
Hidden.........: false
```

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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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..: 100
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

storage.encryptionKey

Contains the storage encryption key. This setting is hidden..

```
Setting name...: storage.encryptionKey
Default value..: null
Set at run-time: false
Hidden.........: true
```

storage.makeFullCheckpointAfterCreate

Indicates whether a full checkpoint should be performed, if storage was created..

```
Setting name...: storage.makeFullCheckpointAfterCreate
Default value..: true
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

storage.useWAL

Whether WAL should be used in paginated storage..

```
Setting name...: storage.useWAL
Default value..: true
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

storage.wal.commitTimeout

Maximum interval between WAL commits (in ms.).

```
Setting name...: storage.wal.commitTimeout
Default value..: 1000
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

storage.wal.fuzzyCheckpointInterval

Interval between fuzzy checkpoints (in seconds).

```
Setting name...: storage.wal.fuzzyCheckpointInterval
Default value..: 300
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

storage.keepOpen

Deprecated.

```
Setting name...: storage.keepOpen
Default value..: true
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

Db

db.pool.min

Default database pool minimum size..

```
Setting name...: db.pool.min
Default value..: 1
Set at run-time: false
Hidden.........: false
```

db.pool.max

Default database pool maximum size..

```
Setting name...: db.pool.max
Default value..: 100
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

db.pool.idleCheckDelay

Delay time on checking for idle databases..

```
Setting name...: db.pool.idleCheckDelay
Default value..: 0
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

db.validation

Enables or disables validation of records..

```
Setting name...: db.validation
Default value..: true
Set at run-time: true
Hidden.........: 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
Hidden.........: false
```

db.makeFullCheckpointOnSchemaChange

When index schema is changed, a full checkpoint is performed..

```
Setting name...: db.makeFullCheckpointOnSchemaChange
Default value..: true
Set at run-time: true
Hidden.........: false
```

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
Hidden.........: false
```

db.mvcc

Deprecated, MVCC cannot be disabled anymore.

```
Setting name...: db.mvcc
Default value..: true
Set at run-time: false
Hidden.........: false
```

db.use.distributedVersion

Deprecated, distributed version is not used anymore.

```
Setting name...: db.use.distributedVersion
Default value..: false
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

tx.commit.synch

Synchronizes the storage after transaction commit.

```
Setting name...: tx.commit.synch
Default value..: false
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

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
Hidden.........: false
```

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
Hidden.........: 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
Hidden.........: false
```

index.flushAfterCreate

Flush storage buffer after index creation..

```
Setting name...: index.flushAfterCreate
Default value..: true
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

index.cursor.prefetchSize

Default prefetch size of index cursor..

```
Setting name...: index.cursor.prefetchSize
Default value..: 500000
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

RidBag

ridBag.embeddedDefaultSize

Size of embedded RidBag array, when created (empty).

```
Setting name...: ridBag.embeddedDefaultSize
Default value..: 4
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

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
Hidden.........: false
```

Collections

collections.preferSBTreeSet

This configuration setting is experimental..

```
Setting name...: collections.preferSBTreeSet
Default value..: false
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

file.lock

Locks files when used. Default is true.

```
Setting name...: file.lock
Default value..: true
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

file.deleteRetry

Number of retries to delete a locked file..

```
Setting name...: file.deleteRetry
Default value..: 50
Set at run-time: false
Hidden.........: false
```

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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

Network

network.maxConcurrentSessions

Maximum number of concurrent sessions..

```
Setting name...: network.maxConcurrentSessions
Default value..: 1000
Set at run-time: true
Hidden.........: false
```

network.socketBufferSize

TCP/IP Socket buffer size..

```
Setting name...: network.socketBufferSize
Default value..: 32768
Set at run-time: true
Hidden.........: false
```

network.lockTimeout

Timeout (in ms) to acquire a lock against a channel..

```
Setting name...: network.lockTimeout
Default value..: 15000
Set at run-time: true
Hidden.........: false
```

network.socketTimeout

TCP/IP Socket timeout (in ms)..

```
Setting name...: network.socketTimeout
Default value..: 15000
Set at run-time: true
Hidden.........: false
```

network.requestTimeout

Request completion timeout (in ms)..

```
Setting name...: network.requestTimeout
Default value..: 3600000
Set at run-time: true
Hidden.........: false
```

network.retry

Number of attempts to connect to the server on failure..

```
Setting name...: network.retry
Default value..: 5
Set at run-time: true
Hidden.........: false
```

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
Hidden.........: false
```

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
Hidden.........: false
```

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
Hidden.........: false
```

network.binary.maxLength

TCP/IP max content length (in bytes) of BINARY requests..

```
Setting name...: network.binary.maxLength
Default value..: 32736
Set at run-time: true
Hidden.........: false
```

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
Hidden.........: false
```

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
Hidden.........: false
```

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
Hidden.........: false
```

network.http.charset

Http response charset.

```
Setting name...: network.http.charset
Default value..: utf-8
Set at run-time: true
Hidden.........: false
```

network.http.jsonResponseError

Http response error in json..

```
Setting name...: network.http.jsonResponseError
Default value..: true
Set at run-time: true
Hidden.........: false
```

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
Hidden.........: false
```

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
Hidden.........: false
```

network.http.useToken

Enable Token based sessions for http..

```
Setting name...: network.http.useToken
Default value..: false
Set at run-time: false
Hidden.........: false
```

network.token.secretyKey

Network token sercret key..

```
Setting name...: network.token.secretyKey
Default value..:
Set at run-time: false
Hidden.........: false
```

network.token.encriptionAlgorithm

Network token algorithm..

```
Setting name...: network.token.encriptionAlgorithm
Default value..: HmacSHA256
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

Profiler

profiler.enabled

Enables the recording of statistics and counters..

```
Setting name...: profiler.enabled
Default value..: false
Set at run-time: true
Hidden.........: false
```

profiler.config

Configures the profiler as ,,.

```
Setting name...: profiler.config
Default value..: null
Set at run-time: true
Hidden.........: false
```

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
Hidden.........: false
```

Log

log.console.level

Console logging level..

```
Setting name...: log.console.level
Default value..: info
Set at run-time: true
Hidden.........: false
```

log.file.level

File logging level..

```
Setting name...: log.file.level
Default value..: fine
Set at run-time: true
Hidden.........: 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
Hidden.........: false
```

cache.local.enabled

Deprecated, Level1 cache cannot be disabled anymore.

```
Setting name...: cache.local.enabled
Default value..: true
Set at run-time: false
Hidden.........: false
```

Command

command.timeout

Default timeout for commands (in ms)..

```
Setting name...: command.timeout
Default value..: 0
Set at run-time: true
Hidden.........: false
```

command.cache.enabled

Enable command cache..

```
Setting name...: command.cache.enabled
Default value..: false
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

Query

query.parallelAuto

Auto enable parallel query, if requirements are met..

```
Setting name...: query.parallelAuto
Default value..: true
Set at run-time: false
Hidden.........: false
```

query.parallelMinimumRecords

Minimum number of records to activate parallel query automatically..

```
Setting name...: query.parallelMinimumRecords
Default value..: 300000
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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..: 100000
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

Statement

statement.cacheSize

Number of parsed SQL statements kept in cache..

```
Setting name...: statement.cacheSize
Default value..: 100
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

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
Hidden.........: false
```

client.ssl.enabled

Use SSL for client connections..

```
Setting name...: client.ssl.enabled
Default value..: false
Set at run-time: false
Hidden.........: false
```

client.ssl.keyStore

Use SSL for client connections..

```
Setting name...: client.ssl.keyStore
Default value..: null
Set at run-time: false
Hidden.........: false
```

client.ssl.keyStorePass

Use SSL for client connections..

```
Setting name...: client.ssl.keyStorePass
Default value..: null
Set at run-time: false
Hidden.........: false
```

client.ssl.trustStore

Use SSL for client connections..

```
Setting name...: client.ssl.trustStore
Default value..: null
Set at run-time: false
Hidden.........: false
```

client.ssl.trustStorePass

Use SSL for client connections..

```
Setting name...: client.ssl.trustStorePass
Default value..: null
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

client.channel.minPool

Minimum pool size.

```
Setting name...: client.channel.minPool
Default value..: 1
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

server.cache.staticFile

Cache static resources upon loading..

```
Setting name...: server.cache.staticFile
Default value..: false
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

distributed.commandTaskTimeout

Maximum timeout (in ms) to wait for Command remote tasks..

```
Setting name...: distributed.commandTaskTimeout
Default value..: 10000
Set at run-time: true
Hidden.........: false
```

distributed.commandLongTaskTimeout

Maximum timeout (in ms) to wait for Long-running remote tasks..

```
Setting name...: distributed.commandLongTaskTimeout
Default value..: 86400000
Set at run-time: true
Hidden.........: false
```

distributed.deployDbTaskTimeout

Maximum timeout (in ms) to wait for database deployment..

```
Setting name...: distributed.deployDbTaskTimeout
Default value..: 1200000
Set at run-time: true
Hidden.........: false
```

distributed.deployChunkTaskTimeout

Maximum timeout (in ms) to wait for database chunk deployment..

```
Setting name...: distributed.deployChunkTaskTimeout
Default value..: 15000
Set at run-time: true
Hidden.........: false
```

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
Hidden.........: false
```

distributed.queueTimeout

Maximum timeout (in ms) to wait for the response in replication..

```
Setting name...: distributed.queueTimeout
Default value..: 5000
Set at run-time: true
Hidden.........: false
```

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
Hidden.........: 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
Hidden.........: 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
Hidden.........: false
```

distributed.queueMaxSize

Maximum queue size to mark a node as stalled. If the numer 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..: 100
Set at run-time: false
Hidden.........: false
```

distributed.backupDirectory

Directory where the copy of an existent database is saved, before it is downloaded from the cluster..

```
Setting name...: distributed.backupDirectory
Default value..: ../backup/databases
Set at run-time: false
Hidden.........: 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
Hidden.........: false
```

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
Hidden.........: false
```

Oauth2

oauth2.secretkey

Http OAuth2 secret key..

```
Setting name...: oauth2.secretkey
Default value..:
Set at run-time: false
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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
Hidden.........: 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: Graph-Database-Tinkerpop#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: Graph-Database-Tinkerpop#create-element-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 son'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
        }
    }
}
```

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
- Scale up on writes
- Scale up on reads
- Replication vs Sharding

Generic advice

Load Balancing

Active 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 transfering 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.

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, 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. 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.

Asynchronous replication

To speed up things, you can setup Asynchronous Replication to remove the latency bottleneck. In this case the coordinator server node execute the operation locally and gives the answer to the client. The entire replication will be in background. In case the quorum is not reached, the changes will be rollbacked transparently.

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 Sever Security.
- Additionally, you can encrypt the database contents on disk. For more information on this process, see Database Encryption.

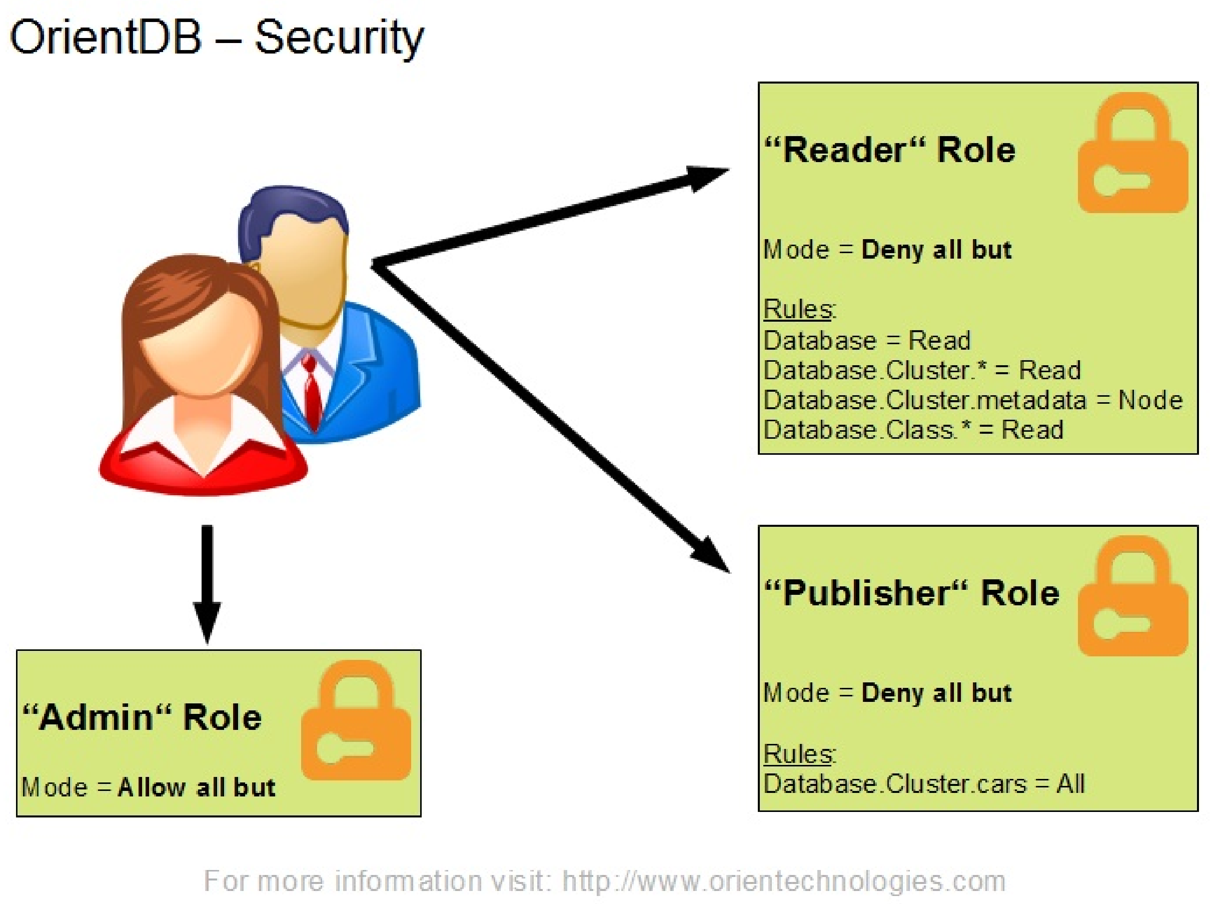
❗	While OrientDB Server can function as a regular Web Server, it is not recommended that you expose it directly to either the Internet or public networks. Instead, always hide OrientDB server in private networks.

See also:

- 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 flask.

```
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 - 1
UPDATE:             #0100 - 4
Permission to use:  #0110 - 5
```

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="{SHA-256}55F95B91628EF3E679628ACB23AE" resources="*" />
    <user name="guest" password="guest" 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 SHA-256 algorithm.

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.

Resources	Description
`server.info`	Retrieves server information and statistics.
`server.listDatabases`	Lists available databases on the server.
`database.create`	Creates a new database in the server
`database.drop`	Drops a database
`database.passthrough`	Allows access to all managed databases.

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.

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**: As of 2.2 this feature is in beta. It will be final with 2.2 GA.

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 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

Server Configuration

The server configuration file, `$ORIENTDB_HOME/config/orientdb-server-config.xml`, does not use SSL by default. To enable SSL 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`. 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 implementations. 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.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"/>
       <!-- 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/ore trust-stores other than that of the default JVN, 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 SSL 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>
```

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");
```

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 (native) API	Native	Native implementation.
	JDBC driver	Native	For legacy and reporting/Business Intelligence applications and JCA integration for J2EE containers
	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
	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**
	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	
	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.
	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

A **Function** is an executable unit of code that can take parameters and return a result. Using Functions you can perform Functional programming where logic and data are all together in a central place. Functions are similar to the Stored Procedures of RDBMS.

> **NOTE**: This guide refers to the last available release of OrientDB. For past revisions look at Compatibility.

OrientDB Functions features:

- are persistent
- can be written in SQL or Javascript (Ruby, Scala, Java and other languages are coming)
- can be executed via SQL, Java, REST and Studio
- can call each other
- supports recursion
- have automatic mapping of parameters by position and name
- plugins can inject new objects to being used by functions

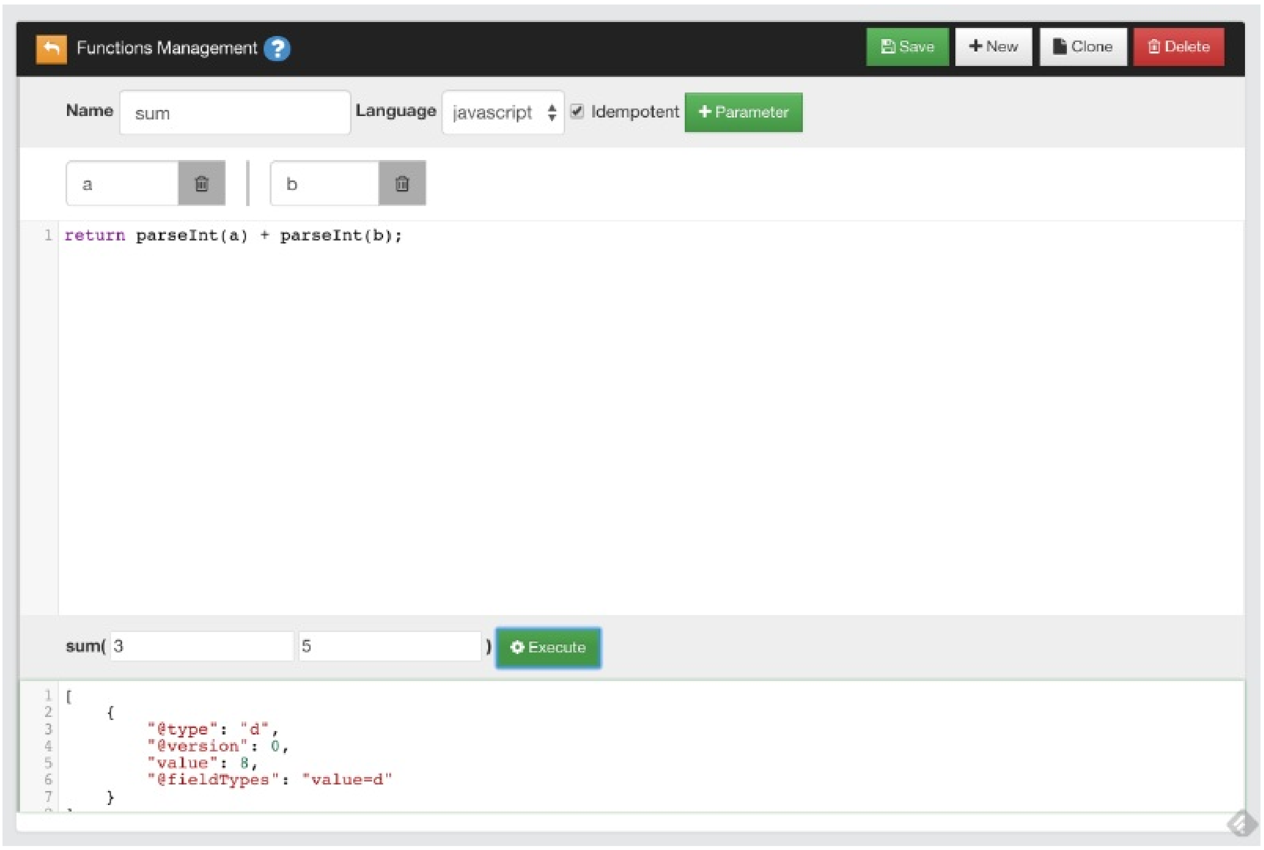
Create your first function

To start using Functions the simplest way is using the Studio. Open the database and go to the "Functions" panel. Then write as name "sum", add 2 parameters named "a" and "b" and now write the following code in the text area:

```
return parseInt(a) + parseInt(b);
```

Click on the "Save" button. Your function has been saved and will appear on the left between the available functions.

Now let's go to test it. On the bottom you will find 2 empty boxes. This is where you can insert the parameters when invoking the function. Write 3 and 5 as parameters and click "Execute" to see the result. "8.0" will appear in the output box below.



Name: sum
Language: javascript
[✓ Idempotent]
a
b

```
return parseInt(a) + parseInt(b);
```

sum(3 5)

```
[
    {
        "@type": "d",
        "@version": 0,
        "value": 8,
        "@fieldTypes": "value=d"
    }
```

Why using parseInt() and not just `a + b`*? because HTTP protocol passes parameters as strings.*

Where are my functions saved?

Functions are saved in the database using the `OFunction` class and the following properties:

- `name` , as the name of the function
- `code` , as the code to execute
- `parameters` , as an optional `EMBEDDEDLIST` of String containing the parameter names if any
- `idempotent` , tells if the function is `idempotent` , namely if it changes the database. Read-only functions are `idempotent` . This is needed to avoid calling non- `idempotent` functions using the HTTP GET method

Concurrent editing

Since OrientDB uses 1 record per function, the MVCC mechanism is used to protect against concurrent record updates.

Usage

Usage via Java API

Using OrientDB's functions from Java is straightforward. First get the reference to the Function Manager, get the right function and execute it passing the parameters (if any). In this example parameters are passed by position:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("local:/tmp/db");
db.open("admin", "admin");
OFunction sum = db.getMetadata().getFunctionLibrary().getFunction("sum");
Number result = sum.execute(3, 5);
```

If you're using the Blueprints Graph API get the reference to the Function in this way:

```
OFunction sum = graph.getRawGraph().getMetadata().getFunctionLibrary().getFunction("sum");
```

You can execute functions passing parameters by name:

```
Map<String,Object> params = new HashMap<String,Object>();
params.put("a", 3);
params.put("b", 5);
Number result = sum.execute(params);
```

Usage via HTTP REST

Each function is exposed as a REST service allowing the receiving of parameters. Parameters can be passed by position in the URL, or starting from 2.1 can be passed in the request payload as JSON. In this case the mapping is not positional, but by name.

Example to execute the `sum` function created before passing 3 and 5 as parameters in the URL, so positional:

```
http://localhost:2480/function/demo/sum/3/5
```

Since 2.1, parameters can be passed also in the request's payload in a JSON, so by name:

```
{ "a": 3, "b": 5 }
```

Both calls will return an HTTP 202 OK with an envelope containing the result of the calculation:

```
{"result":[{"@type":"d","@version":0,"value":2}]}
```

You can call with HTTP GET method only functions declared as "idempotent". Use HTTP POST to call any functions.

If you're executing the function using HTTP POST method, encode the content and set the HTTP request header to: `"Content-Type: application/json"` .

For more information, see HTTP REST protocol. To learn how to write server-side function for web applications, see Server-Side functions.

Function return values in HTTP calls

When calling a function as a REST service, OrientDB encapsulates the result in a JSON and sends it to the client via HTTP. The result can be slightly different depending on the return value of the function. Here are some details about different cases:

- a function that returns a number:

  ```
  return 31;
  ```

result:

```
{"result":[{"@type":"d","@version":0,"value":31}]}
```

- a function that returns a JS object

  ```
  return {"a":1, "b":"foo"}
  ```

result:

```
{"result":[{"@type":"d","@version":0,"value":{"a":1,"b":"foo"}}]}
```

- a function that returns an array

  ```
  return [1, 2, 3]
  ```

result:

```
{"result":[{"@type":"d","@version":0,"value":[1,2,3]}]}
```

- a function that returns a query result

  ```
  return db.query("select from OUser")
  ```

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"
        }
    ]
}
```

Access to the databases from Functions

OrientDB always binds a special variable `orient` to use OrientDB services from inside the functions. The most important methods are:

- `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

Execute a query

Query is an idempotent command. To execute a query use the `query()` method. Example:

```
return orient.getDatabase().query("select name from ouser");
```

Execute a query with external parameters

Create a new function with name `getyUserRoles` with the parameter `user` . Then write this code:

```
return orient.getDatabase().query("select roles from ouser where name = ?", name );
```

The name parameter is bound as variable in Javascript. You can use this variable to build your query.

Execute a command

Commands can be written in any language supported by JVM. By default OrientDB supports "SQL" and "Javascript".

SQL Command

```
var gdb = orient.getGraph();
var results = gdb.command( "sql", "select from Employee where company = ?", [ "Orient Technologies" ] );
```

The result of command is an array of objects, where objects can be:

- OrientVertex instances if vertices are returned
- OrientEdge instances if edges are returned
- OIdentifiable, or any subclasses of it, instances if records are returned

Write your own repository classes

Functions are the perfect place to write the logic for your application to access to the database. You could adopt a DDD approach allowing the function to work as a Repository or a DAO.

This mechanism provides a thin (or thick if you prefer) layer of encapsulation which may protect you from database changes.

Furthermore each function is published and reachable via HTTP REST protocol allowing the automatic creation of a RESTful service.

Example

Below an example 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(){

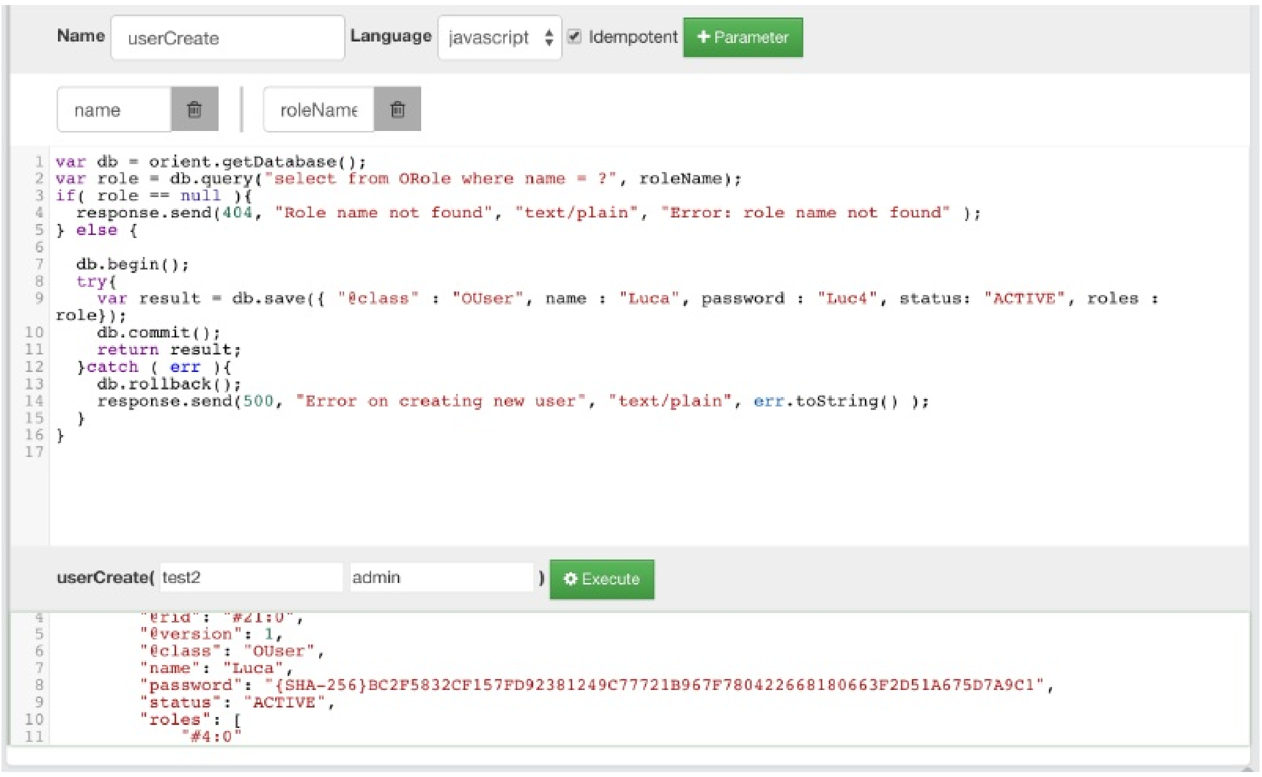
```
return user_getByName("admin");
```

}

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() );
  }
}
```

}



```
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() );
  }
}
```

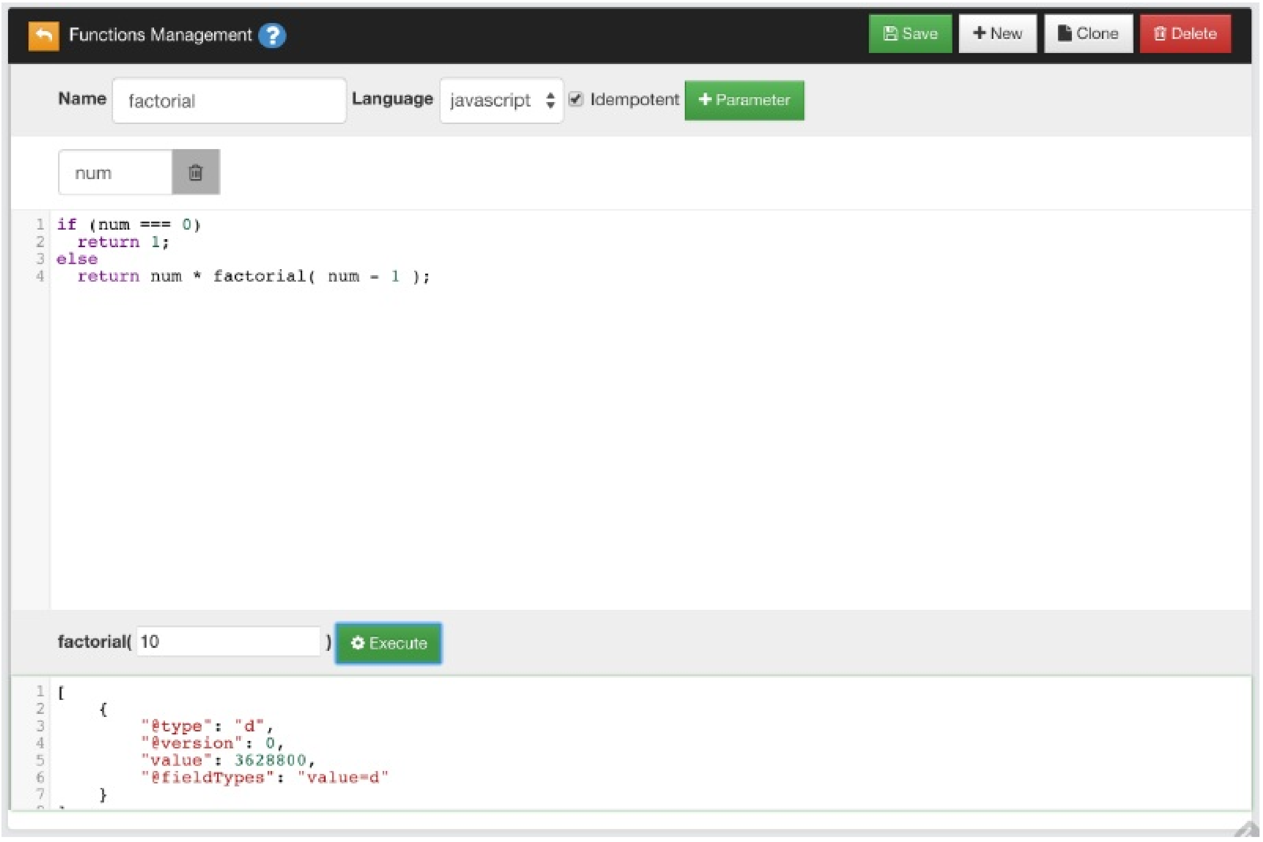
userCreate(test2 admin)

```
        "@rid": "#21:0",
        "@version": 1,
        "@class": "OUser",
        "name": "Luca",
        "password": "{SHA-256}BC2F5832CF157FD92381249C77721B967F780422668180663F2D51A675D7A9C1",
        "status": "ACTIVE",
        "roles": [
            "#4:0"
```

Recursive calls

Create the new function with name "factorial" with the parameter "n". Then write this code:

```
if (num === 0)
  return 1;
else
  return num * factorial( num - 1 );
```



```
if (num === 0)
  return 1;
else
  return num * factorial( num - 1 );
```

factorial(10)

```
[
    {
        "@type": "d",
        "@version": 0,
        "value": 3628800,
        "@fieldTypes": "value=d"
    }
```

This function calls itself to find the factorial number for `<num>` as parameter. The result is `3628800.0` .

Server-Side functions

Server-Side functions can be used as Servlet replacement. To know how to call a Server-Side function, see Usage via HTTP REST. When server-side functions are called via HTTP REST protocol, OrientDB embeds a few additional variables:

- **request**, as the HTTP request and implemented by `OHttpRequestWrapper` class
- **response**, as the HTTP request response implemented by `OHttpResponseWrapper` class
- **util**, as an utility class with helper functions to use inside the functions. It's implemented by `OFunctionUtilWrapper` class

Request object

Refer to this object as "request". Example:

```
var params = request.getParameters();
```

Method signature	Description	Return type
`getContent()`	Returns the request's content	String
`getUser()`	Gets the request's user name	String
`getContentType()`	Returns the request's content type	String
`getHttpVersion()`	Return the request's HTTP version	String
`getHttpMethod()`	Return the request's HTTP method called	String
`getIfMatch()`	Return the request's IF-MATCH header	String
`isMultipart()`	Returns if the requests has multipart	boolean
`getArguments()`	Returns the request's arguments passed in REST form. Example: /2012/10/26	String[]
`getArgument(<position>)`	Returns the request's argument by position, or null if not found	String
`getParameters()`	Returns the request's parameters	String
`getParameter(<name>`)	Returns the request's 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's URL	String

Response object

Refer to this object as "response". Example:

```
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's additional headers	String
`setHeader(String header)`	Sets the response's additional headers to send back. To specify multiple headers use the line breaks	Request object
`getContentType()`	Returns the response's content type. If null will be automatically detected	String
`setContentType(String contentType)`	Sets the response's content type. If null will be automatically detected	Request object
`getCharacterSet()`	Returns the response's character set used	String
`setCharacterSet(String characterSet)`	Sets the response's character set	Request object
`getHttpVersion()`		String
`writeStatus(int httpCode, String reason)`	Sets the response's status as HTTP code and reason	Request object
`writeStatus(int httpCode, String reason)`	Sets the response's status as HTTP code and reason	Request object
`writeHeaders(String contentType)`	Sets the response's headers using the keep-alive	Request object
`writeHeaders(String contentType, boolean keepAlive)`	Sets the response's 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

Refer to this object as `util`. Example:

```
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 equals to "undefined"	Boolean

Native functions

OrientDB's SQL dialect supports many functions written in native language. To obtain better performance you can write you own native functions in Java language and register them to the engine.

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

Plugins

If you're looking for drivers or JDBC connector go to Programming-Language-Bindings.

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

spring-orientdb is an attempt to provide a PlatformTransactionManager for OrientDB usable with the Spring Framework, in particular with @Transactional annotation. Apache 2 license

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.

Plugin for FuelPHP framework.

---

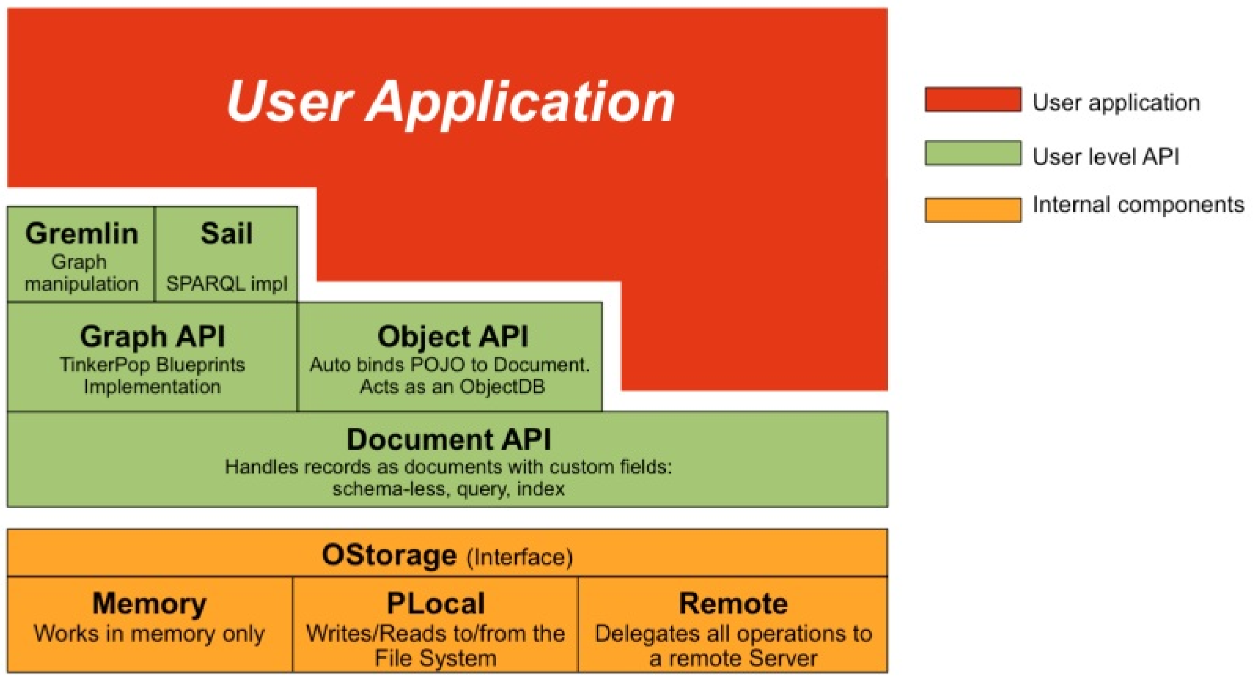
orientqb

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 100% in Java. You can use the native Java APIs without any driver or adapter. Here is the Javadocs.

Architecture of components


User Application
Gremlin
Graph manipulation
Sail
SPARQL impl
Graph API
TinkerPop Blueprints Implementation
Object API
Auto binds POJO to Document. Acts as an ObjectDB
Document API
Handles records as documents with custom fields: schema-less, query, index
OStorage (Interface)
Memory
Works in memory only
PLocal
Writes/Reads to/from the File System
Remote
Delegates all operations to a remote Server
User application
User level API
Internal components

OrientDB provides 3 different Java APIs to work with OrientDB. Each one has pros and cons.

Which API to choose between Graph and Document? Look also at Graph-or-Document-API?.

Graph API

Use OrientDB as a Graph Database working with Vertices and Edges. Graph API is 100% compliant with TinkerPop standard.

API: Graph API

Document API

Handles records as documents. Documents are comprised of fields. Fields can be any of the types supported. Does not need a Java domain POJO, as required for the Object Database. Can be used as schema-less or schema-base modes.

API: Document API

Object API

It's the JPA like interface where POJO are automatically bound to the database as documents. Can be used in schema-less or schema-based modes. This API hasn't been improved since OrientDB v1.5. Please consider using Document or Graph API by writing an additional layer of mapping with your POJO. While you can use both Graph and Document APIs at the same time, the Object API is compatible with Document API, but it doesn't work very well with the Graph API. The main reason is that you should create POJOs that mimic the Vertex and Edge classes with sub optimal performance in comparison with direct Graph API. For this reason we don't suggest to work with Object API with a Graph domain. You could evaluate using Object Mapping on top of OrientDB Blueprints Graph API, such as TinkerPop Frames, Ferma and Totorom.

API: Object Database

What to use? Feature Matrix

	Graph	Document	Object
API	Graph API	Document API	Object Database
Use this if	You work with **graphs** and want your code to be **portable** across **TinkerPop Blueprints** implementations	Your domain fits better the Document Database use case with **schema-less structures**	If you need a full **Object Oriented** abstraction that binds all the database entities to **POJO** (Plain Old Java Object)
Easy to switch from	Other GraphDBs like Neo4J or Titan. If you used TinkerPop standard OrientDB is a drop-in replacement	Other DocumentDB like MongoDB and CouchDB	JPA applications
Java class	OrientGraph	ODatabaseDocumentTx	OObjectDatabaseTx
Query	Yes	Yes	Yes
Schema Less	Yes	Yes	Yes
Schema full	Yes	Yes	Yes
Speed `*`	90%	100%	50%

`*` Speed comparison for generic CRUD operations such as query, insertion, update and deletion. Larger is better. 100% is fastest. In general the price of a high level of abstraction is a speed penalty, but remember that Orient is orders of magnitude faster than the classic RDBMS. So using the Object Database gives you a high level of abstraction with much less code to develop and maintain.

Which library do I use?

OrientDB comes with some jar files contained in the lib directory

JAR name	Description	When required	Depends on 3rd party jars
`orientdb-core-*.jar`	Core library	Always	`snappy-*.jar` as optional, performance pack: `orientdb-nativeos-*.jar` , `jna-*.jar` and `jna-platform-*.jar`
`orientdb-client-*.jar`	Remote client	When your application talks with a remote server	
`orientdb-enterprise-*.jar`	Deprecated since v2.2. Base package with the protocol and network classes shared by client and server	When your application talks with a remote server	
`orientdb-server-*.jar`	Server component	It's used by the server component. Include it only if you're embedding a server	
`orientdb-tools-*.jar`	Contain the console and console commands	Never, unless you want to execute console command directly by your application. Used by the console application	
`orientdb-object-*.jar`	Contain the Object Database interface	Include it if you're using this interface	`javassist.jar` , `persistence-api-1.0.jar`
`orientdb-graphdb-*.jar`	Contain the GraphDB interface	Include it if you're using this interface	`blueprints-core-*.jar`
`orientdb-distributed-*.jar`	Contain the distributed plugin	Include it if you're working with a server cluster	`hazelcast-*.jar`

Java Tutorial

In the event that you are used only to Relation database systems, you may find OrientDB a very unfamiliar system to work with. Given that it also supports Document, Graph and Object-Oriented modes, it requires different Java API's. But, there are some similarities between them too.

Similar to JDBC, a Blueprints API exists, made by Tinkerpop, which supports the basic operations on a graph database. There is an OrientDB driver, (or, to be more accurate, an adapter), which makes it possible to operate without having to deal with OrientDB classes. This means that the resulting code is more portable, given that Blueprints offers adapters to other graphing database systems.

If you need to tweak the database configuraiton, you need to use OrientDB API's directly. It is recommend that in these situations you use a mix: Blueprings when you can, the OrientDB API's where necessary.

OrientDB Java APIs

There are three different API's that OrientDB ships with. Choose one based on your mode.

- Graph API (suggested)
- Document API
- Object API

OrientDB comes with 3 different APIs. Pick your based on your model (for more information look at Java API):

For more information on the API's in general, see Java API

Graph API

Connecting to a Graph Database

The first object you need is a `OrientGraph`:

```
import com.tinkerpop.blueprints.impls.orient.OrientGraph;

OrientGraph graph = new OrientGraph("local:test", "username", "password");
```

Inserting Vertices and Edges

While OrientDB can work with the generic `V` class for verticies and `E` class for edges, you gain much more power by defining custom types for both vertices and edges.

```
odb.createVertexType("Person");
odb.createVertexType("Address");
```

The Blueprint adapter for OrientDB is thread-safe and automatically creates a transaction where necessary. That is, it creates a transaction at the first operation, in the event that a transaction has not yet explicitly been started. You have to specify where 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");
```

Bear in mind, the specific syntax with Blueprint is `class:<class name>`. You must use this syntax in creating an object to specify its class. This is not mandatory. It is also possible to specify a `null` value, (which means a vertex is created with the class `V`, as its the superclass for all vertices in OrientDB).

```
Vertex vPerson = graph.addVertex(null);
```

In consequence of this is that you cannot distinguish `null` vertices from other vertices in a query.

Use a similar API in adding an edge:

```
OrientEdge eLives = graph.addEdge(null, vPerson, vAddress, "lives");
```

In OrientDB, the Blueprints label concept is bound to an edge's class. 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, as you defined the vertex types, but not their properties. By default, OrientDB dynamically accepts everything working in a schema-less mode.

SQL queries

The Tinkerpop interfaces allow you to execute fluent queries or Germlin queries, but you can still use the power of OrientDB SQL through the `.command()` method.

```
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.

Along with queries, you can execute any SQL command like CREATE VERTEX, UPDATE, or DELETE VERTEX. For example,

```
int modified = graph.command(
          new OCommandSQL("UPDATE Customer SET local = true WHERE 'Rome' IN out('lives').name")).execute());
```

This sets a new property called `local` to `true` on all instances in the `Customer` class that live in Rome.

Graph API

OrientDB adheres to the TinkerPop Blueprints standard and uses it as default Graph Java API.

Requirements

To use the Graph API include the following jars in your classpath:

```
orientdb-core-*.jar
blueprints-core-*.jar
orientdb-graphdb-*.jar
```

Also include the following 3rd party jars:

```
jna-*.jar
jna-platform-*.jar
concurrentlinkedhashmap-lru-*.jar
```

If you're connected to a remote server (not local/plocal/memory modes) include also:

```
orientdb-client-*.jar
orientdb-enterprise-*.jar
```

To also use the TinkerPop Pipes tool include also:

```
pipes-*.jar
```

To also use the TinkerPop Gremlin language include also:

```
gremlin-java-*.jar
gremlin-groovy-*.jar
groovy-*.jar
```

NOTE: Starting from v2.0, Lightweight Edges are disabled by default when new database are created.

Introduction

Tinkerpop is a complete stack of projects to handle Graphs:

- **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 being tied to particular underlying data management systems.
- **Pipes** is a graph-based data flow framework for Java 1.6+. A process graph is 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** is a Turing-complete, graph-based programming language designed for key/value-pair multi-relational graphs. Gremlin makes use of an XPath-like syntax to support complex graph traversals. This language has application in the areas of graph query, analysis, and manipulation.
- **Rexster** is 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 its possible to run Rexster over various graph systems. To configure Rexster to work with OrientDB follow this guide: configuration.
- **Sail Ouplementation** to use OrientDB as a RDF Triple Store.

Get started with Blueprints

OrientDB supports different kind of storages and depends by the Database URL used:

- **Persistent embedded** GraphDB. OrientDB is linked to the application as JAR (No network transfer). Use **plocal** as prefix. Example "plocal:/tmp/graph/test"
- **In-Memory embedded** GraphDB. Keeps all the data only in memory. Use **memory** as prefix. Example "memory:test"
- **Persistent remote** GraphDB. Uses a binary protocol to send and receive data from a remote OrientDB server. Use **remote** as prefix. Example "remote:localhost/test". It requires a OrientDB Server instance is up and running at the specified address (localhost in this case). Remote database can be persistent or in-memory as well.

Working with the GraphDB

Before working with a graph you need an instance of OrientGraph class. The constructor gets a URL that is the location of the database. If the database already exists, it will be opened, otherwise it will be created. However a new database can only be created in **plocal** or **memory** mode, not in **remote** mode. In multi-threaded applications use one OrientGraph instance per thread. Also all the graph components (Vertices and Edges) are not thread-safe, so sharing them between threads could cause unpredictable errors.

Remember to always close the graph once done using the `.shutdown()` method.

Example:

```
OrientGraph graph = new OrientGraph("plocal:C:/temp/graph/db");
try {
  ...
} finally {
  graph.shutdown();
}
```

Use the factory

Starting from v1.7 the best way to get a Graph instance is through the OrientGraphFactory. To know more: Use the Graph Factory. Example:

```
// AT THE BEGINNING
OrientGraphFactory factory = new OrientGraphFactory("plocal:C:/temp/graph/db").setupPool(1,10);

// EVERY TIME YOU NEED A GRAPH INSTANCE
OrientGraph graph = factory.getTx();
try {
  ...

} finally {
   graph.shutdown();
}
```

Transactions

Before v2.1.7, every time the graph is modified an implicit transaction is started automatically if no previous transaction was running. Transactions are committed automatically when the graph is closed by calling the `shutdown()` method or by explicit `commit()`. To rollback changes call the `rollback()` method.

After v2.1.7, you can setup the consistency level.

Changes inside a transaction will be temporary until the commit or the close of the graph instance. Concurrent threads or external clients can see the changes only when the transaction has been fully committed.

Full example:

```
try{
  Vertex luca = graph.addVertex(null); // 1st OPERATION: IMPLICITLY BEGIN A 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();
}
```

Surrounding the transaction between a try/catch assures that any errors will rollback the transaction to the previous status for all the involved elements. For more information, look at Concurrency.

NOTE: Before v2.1.7, to work against a graph always use transactional OrientGraph instances and never non-transactional ones to avoid graph corruption from multi-threaded changes. A non-transactional graph instance created with `OrientGraphNoTx graph = factory.getNoTx();` is only useful if you don't work with data but want to define the database schema or for bulk inserts.

Optimistic approach

OrientDB supports optimistic transactions, so no lock is kept when a transaction is running, but at commit time each graph element version is checked to see if there has been an update by another client. This is the reason why you should write your code to be concurrency-proof by handling 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
    }
}
```

Working with Vertices and Edges

Create a vertex

To create a new Vertex in the current Graph call the Vertex OrientGraph.addVertex(Object id)) method. Note that the id parameter is ignored since OrientDB implementation assigns a unique-id once the vertex is created. To return it use Vertex.getId()). Example:

```
Vertex v = graph.addVertex(null);
System.out.println("Created vertex: " + v.getId());
```

Create an edge

An Edge links two vertices previously created. To create a new Edge in the current Graph call the Edge OrientGraph.addEdge(Object id, Vertex outVertex, Vertex inVertex, String label)) method. Note that the id parameter is ignored since OrientDB implementation assigns a unique-id once the Edge is created. To return it use Edge.getId()). `outVertex` is the Vertex instance where the Edge starts and `inVertex` is the Vertex instance where the Edge ends. `label` is the Edge's label. Specify null to not assign it. Example:

```
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());
```

If you're interested on optimizing creation of edges by concurrent threads/clients, look at Concurrency on adding edges.

Retrieve all the Vertices

To retrieve all the vertices use the `getVertices()` method:

```
for (Vertex v : graph.getVertices()) {
    System.out.println(v.getProperty("name"));
}
```

Retrieve all the Edges

To retrieve all the vertices use the getEdges()) method:

```
for (Edge e : graph.getEdges()) {
    System.out.println(e.getProperty("age"));
}
```

NOTE: When Lightweight Edges are enabled (starting from v2.0 are disabled by default), edges are stored as links not as records. This is to improve performance. As a consequence, `getEdges()` will only retrieve records of class E. With useLightweightEdges=true, records of class E are only created under certain circumstances (e.g. if the Edge has properties) otherwise they will be links on the in and out vertices. If you really want `getEdges()` to return all edges, disable the Lightweight Edges feature by executing this command once: `alter database custom useLightweightEdges=false`. This will only take effect for new edges so you'll have to convert the links to actual edges before getEdges will return all edges. For more information look at: Troubleshooting: Why can't I see all the edges.

Removing a Vertex

To remove a vertex from the current Graph call the OrientGraph.removeVertex(Vertex vertex)) method. The vertex will be disconnected from the graph and then removed. Disconnection means that all the vertex's edges will be deleted as well. Example:

```
graph.removeVertex(luca);
```

Removing an Edge

To remove an edge from the current Graph call the OrientGraph.removeEdge(Edge edge)) method. The edge will be removed and the two vertices will not be connected anymore. Example:

```
graph.removeEdge(lucaKnowsMarko);
```

Set and get properties

Vertices and Edges can have multiple properties where the key is a String and the value can be any supported OrientDB types.

- To set a property use the method setProperty(String key, Object value)).
- To get a property use the method Object getProperty(String key)).
- To get all the properties use the method Set<String> getPropertyKeys()).
- To remove a property use the method void removeProperty(String key)).

Example:

```
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

Blueprints Extension OrientDB Blueprints implementation supports setting of multiple properties in one shot against Vertices and Edges. This improves performance avoiding to save the graph element at every property set: setProperties(Object ...)). Example:

```
vertex.setProperties( "name", "Jill", "age", 33, "city", "Rome", "born", "Victoria, TX" );
```

You can also pass a Map of values as first argument. In this case all the map entries will be 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 Element and Properties all together

If you want to create a vertex or an edge while setting the initial properties, the OrientDB Blueprints implementation offers new methods to do it:

```
graph.addVertex("class:Customer", "name", "Jill", "age", 33, "city", "Rome", "born", "Victoria, TX");
```

This creates a new Vertex of class `Customer` with the properties: `name` , `age` , `city` , and `born` . The same is for Edges:

```
person1.addEdge("class:Friend", person2, null, null, "since", "2013-07-30");
```

This creates a new Edge of class `Friend` between vertices `person1` and `person2` with the property `since` .

Both methods accept a `Map<String, Object>` as a parameter to set one property per map entry (see above for the example).

These methods are especially useful if you've declared constraints in the schema. For example, a property cannot be null, and only using these methods will the validation checks succeed.

Using Indices

OrientDB allows execution of queries against any field of vertices and edges, indexed and not-indexed. The first rule to speed up queries is to setup indices on the key properties you use in the query. For example, if you have a query that is looking for all the vertices with the name 'OrientDB' you do this:

```
graph.getVertices("name", "OrientDB");
```

Without an index against the property "name" this execution could take a lot of time. So let's create a new index against the "name" property:

```
graph.createKeyIndex("name", Vertex.class);
```

If the name MUST be unique you can enforce this constraint by setting the index as "UNIQUE" (this is an OrientDB only feature):

```
graph.createKeyIndex("name", Vertex.class, new Parameter("type", "UNIQUE"));
```

This constraint will be applied to all the Vertex and sub-type instances. To specify an index against a custom type like the "Customer" vertices use the additional parameter "class":

```
graph.createKeyIndex("name", Vertex.class, new Parameter("class", "Customer"));
```

You can also have both UNIQUE index against custom types:

```
graph.createKeyIndex("name", Vertex.class, new Parameter("type", "UNIQUE"), new Parameter("class", "Customer"));
```

To create a case insensitive index use the additional parameter "collate":

```
graph.createKeyIndex("name", Vertex.class, new Parameter("type", "UNIQUE"), new Parameter("class", "Customer"),new Parameter("
collate", "ci"));
```

To get a vertex or an edge by key prefix use the class name before the property. For the example above use `Customer.name` in place of only `name` to use the index created against the field `name` of class `Customer` :

```
for (Vertex v : graph.getVertices("Customer.name", "Jay")) {
    System.out.println("Found vertex: " + v);
}
```

If the class name is not passed, then "V" is taken for vertices and "E" for edges:

```
graph.getVertices("name", "Jay");
graph.getEdges("age", 20);
```

For more information about indices look at Index guide.

Using Non-Transactional Graphs

To speed up operations like on massive insertions you can avoid transactions by using a different class than OrientGraph: **OrientGraphNoTx**. In this case each operation is *atomic* and data is updated at each operation. When the method returns, the underlying storage is updated. Use this for bulk inserts and massive operations or for schema definition.

NOTE: Using non-transactional graphs could create corruption in the graph if changes are made in multiple threads at the same time. So use non-transactional graph instances only for non multi-threaded operations.

Configure the Graph

Starting from v1.6 OrientDB supports configuration of the graph by setting all the properties during construction:

Name	Description	Default value
blueprints.orientdb.url	Database URL	-
blueprints.orientdb.username	User name	admin
blueprints.orientdb.password	User password	admin
blueprints.orientdb.saveOriginalIds	Saves the original element IDs by using the property *id*. This could be useful on import of a graph to preserve original ids.	false
blueprints.orientdb.keepInMemoryReferences	Avoids keeping records in memory by using only RIDs	false
blueprints.orientdb.useCustomClassesForEdges	Uses the Edge's label as OrientDB class. If it doesn't exist create it under the hood.	true
blueprints.orientdb.useCustomClassesForVertex	Uses Vertex's label as OrientDB class. If it doesn't exist create it under the hood.	true
blueprints.orientdb.useVertexFieldsForEdgeLabels	Stores the Edge's relationships in the Vertex by using the Edge's class. This allows using multiple fields and makes faster traversal by edge's label (class).	true
blueprints.orientdb.lightweightEdges	Uses Lightweight Edges. This avoids creating a physical document per edge. Documents are created only when the Edges have properties.	false
blueprints.orientdb.autoStartTx	Auto starts a transaction as soon as the graph is changed by adding/remote vertices and edges and properties.	true

Gremlin usage

If you use GREMLIN language with OrientDB remember to initialize it with:

```
OGremlinHelper.global().create()
```

Look at these pages about GREMLIN usage:

- 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

Multi-threaded applications must use one OrientGraph instance per thread. For more information about multi-threading look at Java Multi Threading. Also all the graph components (Vertices and Edges) are not thread-safe, so sharing them between threads could cause unpredictable errors.

Blueprints Extensions

OrientDB is a Graph Database on steroids because it merges the graph, document, and object-oriented worlds together. Below are some of the features exclusive to OrientDB.

Custom types

OrientDB supports custom types for vertices and edges in an Object Oriented manner. Even if this isn't supported directly by Blueprints there are some tricks to use them. Look at the Graph Schema page to know how to create a schema and work against types.

OrientDB added a few variants to the Blueprints methods to work 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 to allow OrientDB to write vertices and edges on multiple files. Furthermore working in Distributed Mode each cluster can be configured to be managed by a different server.

Example:

```
// SAVE THE VERTEX INTO THE CLUSTER 'PERSON_USA' ASSIGNED TO THE NODE 'USA'
graph.addVertex("class:Person,cluster:Person_usa");
```

Retrieve vertices and edges by type

To retrieve all the vertices of `Person` class use the special `getVerticesOfClass(String className)` method:

```
for (Vertex v : graph.getVerticesOfClass("Person")) {
    System.out.println(v.getProperty("name"));
}
```

All the vertices of class Person and all subclasses will be retrieved. This is because by default polymorphism is used. If you're interested ONLY into `Person` vertices (excluding any sub-types) use the `getVerticesOfClass(String className, boolean polymorphic)` method specifying `false` in the second argument `polymorphic` :

```
for (Vertex v : graph.getVerticesOfClass("Person", false)) {
    System.out.println(v.getProperty("name"));
}
```

The same variants also apply to the `getEdges()` method as:

- `getEdgesOfClass(String className)` and
- `getEdgesOfClass(String className, boolean polymorphic)`

Ordered Edges

OrientDB, by default, uses a set to handle the edge collection. Sometimes it's better having an ordered list to access the edge by an offset. Example:

```
person.createEdgeProperty(Direction.OUT, "Photos").setOrdered(true);
```

Every time you access the edge collection the edges are ordered. Below is an example to print all the photos 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 have to use the Document Database API. Here's an example to swap the 10th photo with the last.

```
// REPLACE EDGE Photos
List<ODocument> photos = loadedPerson.getRecord().field("out_Photos");
photos.add(photos.remove(9));
```

To have the same result by using SQL, execute the following commands:

```
create property out_Photos LINKLIST
alter property User.out_Photos custom ordered=true
```

Working on detached elements

When you work with web applications, it’s very common to query elements and render them to the user to let him apply some changes. Once the user updates some fields and presses the “save” button, what happens?

Before now the developer had to track the changes in a separate structure, load the vertex/edge from the database, and apply the changes to the element.

Starting with OrientDB v1.7 we added two new methods to the Graph API on the OrientElement and OrientBaseGraph classes:

- `OrientElement.detach()`
- `OrientElement.attach()`
- `OrientBaseGraph.detach(OrientElement)`
- `OrientBaseGraph.attach(OrientElement)`

Detach

Detach methods fetch all the record content in RAM and reset the connection to the Graph instance. This allows you to modify the element off-line and to re-attach it once finished.

Attach

Once the detached element has been modified, to save it back to the database you need to call the `attach()` method. It restores the connection between the Graph Element and the Graph Instance.

Example

The first step is load a vertex and detach 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();
}
```

After a while the element is updated (from GUI or by application)

```
v.setProperty("name", "super fast!");
```

On “save” 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

Does detach go recursively to detach all connected elements? No, it works only at the current element level.

Can I add an edge against detached elements? No, you can only get/set/remove a property while is detached. Any other operation that requires the database will throw an IllegalStateException.

Execute commands

The OrientDB Blueprints implementation allows you to execute commands using SQL, Javascript, and all the other supported languages.

SQL 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);
}
```

It is possible to have parameters in a query using prepared queries.

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

Along with queries, you can execute any SQL command like `CREATE VERTEX`, `UPDATE`, or `DELETE VERTEX`. In the example below it sets 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 (like `create/alter/drop class` and `create/alter/drop property` commands), remember to force updating of the schema of the database instance you're using by calling `reload()`:

```
graph.getRawGraph().getMetadata().getSchema().reload();
```

For more information look at the available SQL commands.

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 look at SQL Batch.

Database functions

To execute a database function it must be written in Javascript or any other supported languages. In the example below we imagine having written the function `updateAllTheCustomersInCity(cityName)` that executes the same update like above. Note the 'Rome' attribute passed in the `execute()` method:

```
graph.command(
          new OCommandFunction("updateAllTheCustomersInCity")).execute("Rome"));
```

Code

To execute code on the server side you can select between the supported language (by default 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 console or in the local console if the database has been opened in "plocal" mode.

Access to the underlying Graph

Since the TinkerPop Blueprints API is quite raw and doesn't provide ad-hoc methods for very common use cases, you might need to access the underlying ODatabaseGraphTx object to better use the graph-engine under the hood. Commons operations are:

- Count incoming and outgoing edges without browsing them all
- Get incoming and outgoing vertices without browsing the edges
- Execute a query using SQL-like language integrated in the engine

The OrientGraph class provides the method `.getRawGraph()` to return the underlying database: [Document Database].

Example:

```
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 constructor that retrieves the URL, user and password. To know more about OrientDB security visit Security. By default the "admin" user is used.

Tuning

Look at the Performance Tuning Blueprints page.

Graph Factory

TinkerPop Blueprints standard doesn't define a proper "Factory" to get graph instances. For this reason OrientDB users that wanted to use a pool of instances had to mix 2 different API: Graph and Document one. Example:

```
ODatabaseDocumentPool pool = new ODatabaseDocumentPool("plocal:/temp/mydb");
OrientGraph g = new OrientGraph(pool.acquire());
```

Now everything is simpler, thanks to the new OrientGraphFactory class to manage graphs in easy way. These are the main features:

- by default acts as a factory by creating new database instances every time
- can be configured to work as a pool, by recycling database instances
- if the database doesn't exist, it's created automatically (but in "remote" mode)
- returns transactional and non-transactional instances
- on `graph.shutdown()` the pooled instance is returned to the pool to be reused

This is the basic way to create the factory, by using the default "admin" user (with "admin" password by default):

```
OrientGraphFactory factory = new OrientGraphFactory("plocal:/temp/mydb");
```

But you can also pass user and password:

```
OrientGraphFactory factory = new OrientGraphFactory("plocal:/temp/mydb", "jayminer", "amigarocks");
```

To work with a recyclable pool of instances with minimum 1, maximum 10 instances:

```
OrientGraphFactory factory = new OrientGraphFactory("plocal:/temp/mydb").setupPool(1, 10);
```

Once the factory is configured you can get a Graph instance to start working. OrientGraphFactory has 2 methods to retrieve a Transactional and Non-Transactional instance:

```
OrientGraph txGraph = factory.getTx();
OrientGraphNoTx noTxGraph = factory.getNoTx();
```

Or again you can configure in the factory the instances you want and use the get() method every time:

```
factory.setTransactional(false);
OrientGraphNoTx noTxGraph = (OrientGraphNoTx) factory.get();
```

To return the Graph instance to the pool, call the shutdown method on graph instance. `shutdown()` will not close the graph instance, but will keep open and available for the next requester:

```
graph.shutdown();
```

To release all the instances and free all the resources (in case of pool usage), call the close():

```
factory.close();
```

Graph Schema

Although OrientDB can work in schema-less mode, sometimes you need to enforce your data model using a schema. OrientDB supports schema-full or schema-hybrid solutions where the second one means to set such constraints only for certain fields and leave the user to add custom fields to the records. This mode is at class level, so you can have the "Employee" class as schema-full and "EmployeeInformation" class as schema-less.

- **Schema-Full**: enable the strict-mode at class level and set all the fields as mandatory
- **Schema-Less**: create classes with no properties. Default mode is non strict-mode so records can have arbitrary fields
- **Schema-Hybrid**, called also *Schema-Mixed* is the most used: create classes and define some fields but leave the record to define own custom fields

NOTE: *Changes to the schema are not transactional, so execute them outside a transaction.*

To access to the schema, you can use SQL or API. Will follow examples using Java API.

For a tutorial look at the following links:

- Orient Technologies's Blog post about Using Schema with Graphs

Class

A Class, or type, is a concept taken from the Object Oriented paradigm. In OrientDB defines a type of record. It's the closest concept to a Relational DBMS Table. Class can be schema-less, schema-full or mixed. A class can inherit from another shaping a tree of classes. Inheritance means that the sub-class extends the parent one inheriting all the attributes as they was own.

A class must have at least one cluster defined (as its default cluster), but can support multiple ones. In this case By default OrientDB will write new records in the default cluster, but reads will always involve all the defined clusters. When you create a new class by default a new physical cluster is created with the same name of the class in lower-case.

The Graph structure is based on two classes: "V" for Vertices and "E" for Edges. These class are automatically built once a database is built using the mode "graph". If you don't have these classes just create them (see below).

You can build a graph using V and E instances but it's strongly suggested to use custom types for vertices and edges.

Working with custom vertex and edge types

To create a custom Vertex class (or type) use the `createVertexType(<name>)` :

```
OrientGraph graph = new OrientGraph("local:/temp/db");
OrientVertexType account = graph.createVertexType("Account");
```

To create a vertex of type "Account" pass a string with the format `"class:<name>"` as vertex id:

```
Vertex v = graph.addVertex("class:Account");
```

Since classes are polymorphic if you look for generic Vertices also "Account" instances are returned:

```
Iterable<Vertex> allVertices = graph.getVertices();
```

To retrieve only the vertices of "Account" class:

```
Iterable<Vertex> accountVertices = graph.getVerticesOfClass("Account");
```

In Blueprints Edges has the concept of "label" to distinguish between edge types. In OrientDB we binds the concept of Edge label to Edge class. To create an Edge custom type use the similar method `createEdgeType(<name>)` :

```
OrientGraph graph = new OrientGraph("local:/temp/db");
OrientVertexType accountVertex = graph.createVertexType("Account");
OrientVertexType addressVertex = graph.createVertexType("Address");
// CREATE THE EDGE TYPE
OrientEdgeType livesEdge = graph.createEdgeType("Lives");

Vertex account = graph.addVertex("class:Account");
Vertex address = graph.addVertex("class:Address");

// CREATE THE EDGE
Edge e = account.addEdge("Lives", address);
```

Inheritance tree

Classes can extends other classes. Starting from 2.1 OrientDB supports also multiple inheritance. To create a class that extends a class different by "V" (Vertex) and E (Edge) types, pass the class name on construction:

```
graph.createVertexType(<class-name>, <super-class>); // VERTEX TYPE
graph.createEdgeType(<class-name>, <super-class>);  // EDGE TYPE
```

Example to create a base class "Account" and two sub-classes "Provider" and "Customer":

```
graph.createVertexType("Account");
graph.createVertexType("Customer", "Account");
graph.createVertexType("Provider", "Account");
```

Get custom types

To retrieve such custom classes use the methods `graph.getVertexType(<name>)` and `graph.getEdgeType(<name>)` . Example:

```
OrientVertexType accountVertex = graph.getVertexType("Account");
OrientEdgeType livesEdge = graph.getEdgeType("Lives");
```

Drop persistent types

To drop a persistent class use the `dropVertexType(<name>)` and `dropVertexType(<name>)` methods.

```
graph.dropVertexType("Address");
graph.dropEdgeType("Lives");
```

Property

Properties are the fields of the class. In this guide Property is synonym of Field.

Create a property

Once the class has been created, you can define fields (properties). Below an example:

```
OrientVertexType accountVertex = graph.getVertexType("Account");
accountVertex.createProperty("id", OType.INTEGER);
accountVertex.createProperty("birthDate", OType.DATE);
```

Please note that each field must belong to one of [Types supported types].

Drop the Class property

To drop a persistent class property use the `OClass.dropProperty(String)` method.

```
accountVertex.dropProperty("name");
```

The dropped property will not be removed from records unless you explicitly delete them using the [SQLUpdate SQL UPDATE + REMOVE statement]. Example:

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

Constraints

ⓘ	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.

OrientDB supports a number of constrains for each field:

- **Minimum value**, accepts a string because works also for date ranges `setMin()`
- **Maximum value**, accepts a string because works also for date ranges `setMax()`
- **Mandatory**, it must be specified `setMandatory()`
- **Readonly**, it may not be updated after record is created `setReadonly()`
- **Not Null**, can't be NULL `setNotNull()`
- **Unique**, doesn't allow duplicates and speedup searches.
- **Regexp**, it must satisfy the Regular expression.
- **Ordered**, specify if edge list must be ordered, so a List will be used in place of Set. The method is `setOrdered()`

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");
```

Indexes as constrains

To let to a property value to be UNIQUE use the UNIQUE index as constraint by passing a Parameter object with key "type":

```
graph.createKeyIndex("id", Vertex.class, new Parameter("type", "UNIQUE"));
```

This constraint will be applied to all the Vertex and sub-types instances. To specify an index against a custom type use the additional parameter "class":

```
graph.createKeyIndex("name", Vertex.class, new Parameter("class", "Member"));
```

You can also have both UNIQUE index against custom types:

```
graph.createKeyIndex("id", Vertex.class, new Parameter("type", "UNIQUE"), new Parameter("class", "Member"));
```

To get a vertex or an edge by key prefix the class name to the field. For the example above use "Member.name" in place of only "name" to use the index created against the field "name" of class "Member":

```
for( Vertex v : graph.getVertices("Member.name", "Jay") ) {
  System.out.println("Found vertex: " + v );
}
```

If the class name is not passed, then "V" is taken for vertices and "E" for edges:

```
graph.getVertices("name", "Jay");
graph.getEdges("age", 20);
```

For more information about indexes look at Index guide.

(Go back to Graph-Database-Tinkerpop)

Partitioned graphs

This tutorial explains step-by-step how to create partitioned graphs using the Record Level Security feature introduced in OrientDB 1.2.0. This feature is so powerful we can totally separate database's records as sand-boxes where each "Restricted" records can't be accessed by non authorized users. This tutorial demonstrates this sand-boxes works well also with the GraphDB API and the TinkerPop stack. Partitioning graphs allows to build real Multi-tenant applications in a breeze.

Requirements:

- OrientDB 1.2.0-SNAPSHOT or higher
- TinkerPop Blueprints 2.2.0 or higher.

Create a new empty graph database

First open the console of the GraphDB Edition and create the new database "blog" of type "graph" against 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 local graph
Creating database [local:../databases/blog] using the storage type [local]...
Database created successfully.

Current database is: local:../databases/blog
```

Enable graph partitioning

Now turn on partitioning against graph by letting classes V (Vertex) and E (Edge) to extend the éORestricted* class. In this way any access to Vertex and Edge instances can be restricted:

```
ALTER CLASS V superclass orestricted

Class updated successfully
```

```
ALTER CLASS E superclass orestricted

Class updated successfully
```

Create 2 users

Now let's go creating 2 users: "luca" and "steve". First ask the current roles in database to know the "writer" role's rid:

```
SELECT FROM orole

 ---+------+--------+------+---------------------------------------------------------------+----------------
  # | RID  | name   | mode | rules                                                         | inheritedRole
 ---+------+--------+------+---------------------------------------------------------------+-----------------
  0 | #4:0 | admin  | 1    | {}                                                            | null
  1 | #4:1 | reader | 0    | {database=2, database.schema=2, database.cluster.internal=2,  | null
    |      |        |      | {database.cluster.orole=2, database.cluster.ouser=2,          |
    |         |        |      | database.class.*=2, database.cluster.*=2, database.command=2, |
    |       |        |      | database.hook.record=2                                          |
  2 | #4:2 |writer  | 0    | {database=2, database.schema=7, database.cluster.internal=2,  | null
    |      |        |      |  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).
```

Found it, it's the #4:2. Not create 2 users with as first role #4:2 (writer):

```
INSERT INTO ouser SET name = 'luca', status = 'ACTIVE', password = 'luca', roles = [#4:2]

Inserted record 'OUser#5:4{name:luca,password:{SHA-256}D70F47790F689414789EEFF231703429C7F88A10210775906460EDBF38589D90,roles:
[1]} v1' in 0,001000 sec(s).

INSERT INTO ouser SET name = 'steve', status = 'ACTIVE', password = 'steve', roles = [#4:2]

Inserted record 'OUser#5:3{name:steve,password:{SHA-256}F148389D080CFE85952998A8A367E2F7EAF35F2D72D2599A5B0412FE4094D65C,roles
:[1]} v1' in 0,001000 sec(s).
```

Create a simple graph as user 'Luca'

Now it's time to disconnect and reconnect to the blog database using the new "luca" user:

```
DISCONNECT

Disconnecting from the database [blog]...OK
```

```
CONNECT local:../databases/blog luca luca
Connecting to database [local:../databases/blog] with user 'luca'...OK
```

Now create 2 vertices: a Restaurant and a Pizza:

```
CREATE VERTEX SET label = 'food', name = 'Pizza'

Created vertex 'V#9:0{label:food,name:Pizza,_allow:[1]} v0' in 0,001000 sec(s).
```

```
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).
```

Now connect these 2 vertices with an edge labelled "menu":

```
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).
```

To check if everything is ok execute a select against vertices:

```
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).
```

Create a simple graph as user 'Steve'

Now let's connect to the database using the 'Steve' user and check if there are vertices:

```
DISCONNECT

Disconnecting from the database [blog]...OK

CONNECT local:../databases/blog steve steve
Connecting to database [local:../databases/blog] with user 'steve'...OK

SELECT FROM V

0 item(s) found. Query executed in 0.0 sec(s).
```

Ok, no vertices found. Try to create something:

```
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).

CREATE VERTEX SET label = 'driver', name = 'steve'

Created vertex 'V#9:3{label:driver,name:steve,_allow:[1]} v0' in 0,000000 sec(s).

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).
```

Now check the graph just created:

```
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).
```

The "Steve" user doesn't see the vertices and edges creates by other users!

What happen if we try to connect 2 vertices of different users?

```
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 is totally isolated and OrientDB thinks the vertex doesn't exist while it's present, but invisible to the current user.

TinkerPop Stack

Record Level Security feature is very powerful because acts at low level inside the OrientDB engine. This is why everything works like a charm, even the TinkerPop stack.

Now try to display all the 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).
```

The same is using other technologies that use the !TinkerPop Blueprints: TinkerPop Rexter, TinkerPop Pipes, TinkerPop Furnace, TinkerPop Frames and 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 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

OrientDB supports **Lightweight Edges** as regular edges, but without an identity on database. Lightweight edges can be used only when no properties are defined on edge.

By avoiding the creation of the underlying Document, **Lightweight Edges** have the same impact on speed and space as with Document LINKs, but with the additional bonus to have bidirectional connections. This means you can use the MOVE VERTEX command to refactor your graph with no broken LINKs.

Regular Edge representation

Look at the figure below. With Regular Edges both vertices (#10:33 and #10:12) are connected through an Edge Document (#17:11). The outgoing `out_Friend` property in #10:33 document is a set of LINKs with #17:11 as item. Instead, in document #10:12 the relationship is as incoming, so the property `in_Friend` is used with the LINK to the same Edge #17:11.

When you cross this relationship, OrientDB loads the Edge document #17:11 to resolve the other part of the relationship.

```
+---------------------+    +---------------------+    +---------------------+
|   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]  |
                           +---------------------+    +---------------------+
```

Lightweight Edge representation

With Lightweight Edge, instead, there is no Edge document, but both vertices (#10:33 and #10:12) are connected directly to each other. The outgoing `out_Friend` property in #10:33 document contains the direct LINK to the vertex #10:12. The same happens on Vertex document #10:12, where the relationship is as incoming and the property `in_Friend` contains the direct LINK to vertex #10:33.

When you cross this relationship, OrientDB doesn't need to load any edge to resolve the other part of the relationship. Furthermore no edge document is created.

```
+---------------------+    +---------------------+
|   Account Vertex    |    |    Account Vertex   |
|       #10:33        |    |       #10:12        |
+---------------------+    +---------------------+
|out_Friend: [#10:12] |<-->|in_Friend: [#10:33]  |
+---------------------+    +---------------------+
```

Starting from OrientDB v2.0, **Lightweight Edges** are disabled by default with new databases. This is because having regular edges makes easier to act on edges from SQL. Many issues from beginner users were on **Lightweight Edges**. If you want to use **Lightweight Edges**, enable it via API:

```
OrientGraph g = new OrientGraph("mygraph");
g.setUseLightweightEdges(true);
```

Or via SQL:

```
ALTER DATABASE custom useLightweightEdges=true
```

Changing `useLightweightEdges` setting to `true`, will not transform previous edges, but all new edges could be **Lightweight Edges** if they meet the requirements.

When use Lightweight Edges?

These are the PROS and CONS of Lightweight Edges vs Regular Edges:

PROS:

- faster in creation and traversing, because don't need an additional document to keep the relationships between 2 vertices

CONS:

- cannot store properties
- harder working with Lightweight edges from SQL, because there is no a regular document under the edge

Document API

To use the Document API include the following jars in your classpath:

```
orientdb-core-*.jar
```

If you're using the Document Database interface connected to a remote server (not local/embedded mode) include also:

```
orientdb-client-*.jar
orientdb-enterprise-*.jar
```

Introduction

The Orient Document DB is the base of higher-level implementation like Object-Database and Graph-Database. The Document Database API has the following features:

- supports Multi threads access
- supports Transactions
- supports Queries
- supports Traverse
- very flexible: can be used in schema-full, schema-less or schema-hybrid mode.

This is an example to store 2 linked documents in the database:

```
// OPEN THE DATABASE
ODatabaseDocumentTx db = new ODatabaseDocumentTx("remote:localhost/petshop").open("admin", "admin");

// 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();
```

This is the very first example. While the code is pretty clear and easy to understand please note that we haven't declared the type "Person" before now. When an ODocument instance is saved, the declared type "Person" will be created without constraints. To declare persistent classes look at the Schema management.

Use the database

Before to execute any operation you need an opened database instance. You can open an existent database or create a new one. Databases instances aren't thread safe, so use one database per thread.

Before to open or create a database instance you need a valid URL. URL is where the database is available. URL says what kind of database will be used. For example *memory:* means in-memory only database, *plocal:* is for embedded ones and *remote:* to use a remote database hosted on an up & running DBServer OrientDB Server instance. For more information look at Database URL.

Database instances must be closed once finished to release precious resources. To assure it the most common usage is to enclose all the database operations inside a try/finally block:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx("plocal:/temp/test");
db.open("admin", "admin");

try {
  // YOUR CODE
} finally {
  db.close();
}
```

If you are using a remote storage (url starts with **"remote:"**) assure the server is up & running and include the **orientdb-client.jar** file in your classpath.

Multi-threading

The ODatabaseDocumentTx class is non thread-safe. For this reason use different ODatabaseDocumentTx instances by multiple threads. They will share the same Storage instance (with the same URL) and the same level-2 cache. For more information look at Multi-Threading with Java.

Create a new database

In local filesystem

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx ("plocal:/tmp/databases/petshop").create();
```

On a remote server

To create a database in a remote server you need the user/password of the remote OrientDB Server instance. By default the "root" user is created on first startup of the server. Check this in the file config/orientdb-server-config.xml, where you will also find the password.

To create a new document database called dbname on dbhost using filesystem storage (as opposed to in-memory storage):

```
new OServerAdmin("remote:dbhost")
    .connect("root", "kjhsdjfsdh128438ejhj")
    .createDatabase("dbname","document","local").close();
```

To create a graph database replace "document" with "graph".

To store the database in memory replace "local" with "memory".

Open a database

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx ("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.

Use the connection Pool

One of most common use cases is to reuse the database, avoiding to create it every time. It's also the typical scenario of the Web applications. Instead of creating a new ODatabaseDocumentTx instance all the times, get an available instance from the pool:

```
// OPEN THE DATABASE
ODatabaseDocumentTx db = ODatabaseDocumentPool.global().acquire("remote:localhost/petshop", "admin", "admin");
try {
  // YOUR CODE
  ...
} finally {
  db.close();
}
```

Remember to always close the database instance using the `close()` database method like a classic non-pooled database. In this case the database will be not closed for real, but the instance will be released to the pool, ready to be reused by future requests. The best is to use a try/finally block to avoid cases where the database instance remains open, just like the example above.

Global pool

By default OrientDB provide a global pool declared with maximum 20 instances. Use it with: `ODatabaseDocumentPool.global()` .

Use your pool

To create your own pool build it and call the `setup(min, max)` method to define minimum and maximum managed instances. Remember to close it when the pool is not more used. Example:

```
// CREATE A NEW POOL WITH 1-10 INSTANCES
ODatabaseDocumentPool pool = new ODatabaseDocumentPool();
pool.setup(1,10);
...
pool.close();
```

Schema

OrientDB can work in schema-full (like RDBMS), schema-less (like many NoSQL Document databases) and in schema-hybrid mode. For more information about the Schema look at the Schema page.

To use the schema with documents create the ODocument instance using the `ODocument(String className)` constructor passing the class name. If the class hasn't been declared, it's created automatically with no fields. This can't work during transaction because schema changes can't be applied in transactional context.

Security

Few NoSQL solutions supports security. OrientDB does it. To know more about it look at Security.

To manage the security get the Security Manager and use it to work with users and roles. Example:

```
OSecurity sm = db.getMetadata().getSecurity();
OUser user = sm.createUser("god", "god", new String[] { "admin" } );
```

To get the reference to the current user use:

```
OUser user = db.getUser();
```

Create a new document

ODocument instances can be saved by calling the save() method against the object itself. Note that the behaviour depends on the running transaction, if any. See Transactions.

```
ODocument animal = new ODocument("Animal");
animal.field( "name", "Gaudi" );
animal.field( "location", "Madrid" );
animal.save();
```

Retrieve documents

Browse all the documents in a cluster

```
for (ODocument doc : database.browseCluster("CityCars")) {
  System.out.println( doc.field("model") );
```

Browse all the records of a class

```
for (ODocument animal : database.browseClass("Animal")) {
  System.out.println( animal.field( "name" ) );
```

Count records of a class

```
long cars = database.countClass("Car");
```

v= Count records of a cluster ==

```
long cityCars = database.countCluster("CityCar");
```

Execute a query

Although OrientDB is part of the NoSQL database community, it supports a subset of SQL that allows it to process links to documents and graphs.

To know more about the SQL syntax supported go to: SQL-Query.

Example of a SQL query:

```
List<ODocument> result = db.query(
  new OSQLSynchQuery<ODocument>("select * from Animal where ID = 10 and name like 'G%'"));
```

Asynchronous query

OrientDB supports asynchronous queries. The result is not collected and returned like synchronous ones (see above) but a callback is called every time a record satisfy 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;
        // DO SOMETHING WITH THE DOCUMENT

        return resultCount > 20 ? false : true;
      }

      @Override
      public void end() {
      }
    })).execute();
```

Asynchronous queries are useful to manage big result sets because don't allocate memory to collect results.

Non-Blocking query (since v2.1)

Both Synchronous and Asynchronous queries are blocking, that means that the first instruction you have after db.query() or db.command().execute() will be executed only after you received all the result-set or last callback was invoked. OrientDB also supports non-blocking queries. The API is very similar to asynchronous queries (you have a callback that is invoked for every record in the result-set), but the behavior is completely different: the execution of your current thread continues without blocking on the db.query() or db.command().execute(), and the callback is invoked by a different thread. That means that in the meantime you can close your db instance and keep on receiving callbacks from the query result.

```
Future future = database.command(new OSQLNonBlockingQuery<Object>("select * from animal where name = 'Gipsy'",
    new OCommandResultListener() {
      resultCount = 0;
      @Override
      public boolean result(Object iRecord) {
        resultCount++;
        ODocument doc = (ODocument) iRecord;
        // DO SOMETHING WITH THE DOCUMENT

        System.out.println("callback "+resultCount+" invoked");
        return resultCount > 20 ? false : true;
      }

      @Override
      public void end() {
      }
    })).execute();

System.out.println("query executed");

future.get();
```

the result of this snippet of code will be something like

```
query executed
callback 0 invoked
callback 1 invoked
callback 2 invoked
callback 3 invoked
callback 4 invoked
```

but it could also be

```
callback 0 invoked
callback 1 invoked
query executed
callback 2 invoked
callback 3 invoked
callback 4 invoked
```

depending on race conditions on the two parallel threads (the one that fires query execution and then continues with "query executed", and the other one that invokes callbacks).

`future.get();` is a blocking call that returns only after last callback invocation (you can avoid this if you don't need to know when the query terminates).

Prepared query

Prepared query are quite similar to the Prepared Statement of JDBC. Prepared queries are pre-parsed so on multiple execution of the same query are faster than classic SQL queries. Furthermore the pre-parsing doesn't allow SQL Injection. Note: prepared queries (parameter substition) only works with select statements (but not select statements within other types of queries such as "create vertex").

Prepared query uses two kinds of markers to substitute parameters on execution:

```
? is positional parameter
:<par> is named parameter
```

Example of positional parameters:

```
OSQLSynchQuery<ODocument> query = new OSQLSynchQuery<ODocument>("select from Profile where name = ? and surname = ?");
List<ODocument> result = database.command(query).execute("Barack", "Obama");
```

Example of named parameters:

```
OSQLSynchQuery<ODocument> query = new OSQLSynchQuery<ODocument>("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);
```

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();
```

If the command modifies the schema (like `create/alter/drop class` and `create/alter/drop property` commands), remember to force updating of the schema of the database instance you're using:

```
db.getMetadata().getSchema().reload();
```

For more information look at the available SQL commands.

Traverse records

Traversing is the operation to cross documents by links (relationships). OrientDB is a graph database so this operation is much much more efficient than executing a JOIN in the relational databases. To know more about traversing look at the Java traverse API.

The example below traverses, for each movie, all the connected records up to the 5th 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);
}
```

Update a document

Any persistent document can be updated by using the Java API and then by calling the db.save() method. Alternatively, you can call the document's save() method to synchronize the changes to the database. The behaviour depends on the transaction begun, if any. See Transactions.

```
animal.field( "location", "Nairobi" );
animal.save();
```

OrientDB will update only the fields really changed.

Example of how to increase the price of all the animals by 5%:

```
for (ODocument animal : database.browseClass("Animal")) {
  animal.field( "price", animal.field( "price" ) * 105 / 100 );
  animal.save();
}
```

Delete a document

To delete a document call the delete() method on the document instance that's loaded. The behaviour depends on the transaction begun, if any. See Transactions.

```
animal.delete();
```

Example of deletion of all the documents of class "Animal".

```
for (ODocument animal : database.browseClass("Animal"))
  animal.delete();
```

Transactions

Transactions are a practical way to group a set of operations together. OrientDB supports ACID transactions so that all or none of the operations succeed. The database always remains consistent. For more information look at Transactions.

Transactions are managed at the database level. Nested transactions are currently not supported. A database instance can only have one transaction running. The database's methods to handle transactions are:

- **`begin()`** to start a new transaction. If a transaction was already running, it's rolled back and a new one is begun.
- **`commit()`** makes changes persistent. If an error occurs during commit the transaction is rolled back and an OTransactionException exception is raised.
- **`rollback()`** aborts a transaction. All the changes will be lost.

Optimistic approach

The current release of OrientDB only supports OPTIMISTIC transactions where no lock is kept and all operations are checked at commit time. This improves concurrency but can throw an `OConcurrentModificationException` exception in the case where records are modified by concurrent clients or threads. In this scenario, the client code can reload the updated records and repeat the transaction.

Optimistic transactions keep all the changes in memory in the client. If you're using remote storage no changes are sent to the server until `commit()` is called. All the changes will be transferred in a block. This reduces network latency, speeds-up the execution, and increases concurrency. This is a big difference compared to most Relational DBMS where, during a transaction, changes are sent immediately to the server.

Usage

Transactions are committed only when the `commit()` method is called and no errors occur. The most common usage of transactions is to enclose all the database operations inside a `try/finally` block. On closing of the database ("finally" block) if a pending transaction is running it will be rolled back automatically. Look at this example:

```
ODatabaseDocumentTx db = new ODatabaseDocumentTx(url);
db.open("admin", "admin");

try {
  db.begin();
  // YOUR CODE
  db.commit();
} finally {
  db.close();
}
```

Index API

Even though you can use Indices via SQL, the best and most efficient way is to use the Java API.

The main class to use to work with indices is the IndexManager. To get the implementation of the IndexManager use:

```
OIndexManager idxManager = database.getMetadata().getIndexManager();
```

The Index Manager allows you to manage the index life-cycle for creating, deleting, and retrieving an index instance. The most common usage is with a single index. You can get the reference to an index by using:

```
OIndex<?> idx = database.getMetadata().getIndexManager().getIndex("Profile.name");
```

Where "Profile.name" is the index name. Note that by default OrientDB assigns the name as `<class>.<property>` for automatic indices created against a class's property.

The OIndex interface is similar to a Java Map and provides methods to get, put, remove, and count items. The following are examples of retrieving records using a UNIQUE index against a name field and a NOTUNIQUE index against a 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() );
```

While automatic indices are managed automatically by OrientDB hooks, the manual indices can be used to store any value. To create a new entry use the `put()` :

```
OIndex<?> addressbook = database.getMetadata().getIndexManager().getIndex("addressbook");

addressbook.put( "Luke", new ODocument("Contact").field( "name", "Luke" );
```

Resources

- Javadoc: JavaDoc
- OrientDB Studio Web tool.

Schema

Although OrientDB can work in schema-less mode, sometimes you need to enforce your data model using a schema. OrientDB supports schema-full or schema-hybrid solutions where the latter means to set such constraints only for certain fields and to leave the user to add custom fields on the records. This mode is at a class level, so you can have an "Employee" class as schema-full and an "EmployeeInformation" class as schema-less.

- **Schema-Full**: enables the strict-mode at class level and sets all the fields as mandatory.
- **Schema-Less**: creates classes with no properties. Default mode is non strict-mode so records can have arbitrary fields.
- **Schema-Hybrid**, also called *Schema-Mixed* is the most used: creates classes and define some fields but allows the user to define custom fields.

NOTE: *Changes to the schema are not transactional, so execute them outside a transaction.*

To access to the schema, you can use SQL or API. Will follow examples using Java API.

To gain access to the schema APIs you get the OMetadata object from database instance you're using and then call its `getSchema()` method.

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

Class

A Class is a concept taken from the Object Oriented paradigm. In OrientDB a class defines a type of record. It's the closest concept to a relational database table. A Class can be schema-less, schema-full, or mixed.

A Class can inherit from another class. This [#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 that can be logical (by default) or physical. A class must have at least one cluster defined (as its default cluster), but can support multiple ones. In this case By default OrientDB will write new records in the default cluster, but reads will always involve all the defined clusters.

When you create a new class, by default, a new physical cluster is created with the same name as the class (in lowercase).

Create a persistent class

Each class contains one or more properties (also called fields). This mode is similar to the classic relational DBMS approach where you define tables before storing records.

Here's an example of creating an Account class. By default a new [Concepts#Physical_Cluster Physical Cluster] will be created to keep the class instances:

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

To create a new Vertex or Edge type you have to extend the "V" and "E" classes, respectively. Example:

```
OClass person = database.getMetadata().getSchema().createClass("Account",
       database.getMetadata().getSchema().getClass("V"));
```

Look at Graph Schema for more information.

Get a persistent class

To retrieve a persistent class use the `getClass(String)` method. If the class does not exist then null is returned.

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

Drop a persistent class

To drop a persistent class use the `OSchema.dropClass(String)` method.

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

The records of the removed class will not be deleted unless you explicitly delete them before dropping the class. Example:

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

Constraints

To work in schema-full mode set the strict mode at the class level by calling the `setStrictMode(true)` method. In this case, all the properties of the record must be predefined.

Property

Properties are the fields of the class. In this guide a property is synonymous with a field.

Create the Class property

Once the class has been created, you can define fields (properties). Below is an example:

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

Please note that each field must belong to one of these Types.

Drop the Class property

To drop a persistent class property use the `OClass.dropProperty(String)` method.

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

The dropped property will not be removed from records unless you explicitly delete them using the [SQLUpdate SQL UPDATE + REMOVE statement]. Example:

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

Define relationships

OrientDB supports two types of relationships: *referenced* and *embedded*.

Referenced relationships

OrientDB uses a direct **link** to the referenced record(s) without the need of a costly JOIN as does the relational world. Example:

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

Record A will contain the *reference* to the *Record B* in the property called "customer". Note that both records are reachable by any other records since they have a [Concepts#RecordID RecordID].

1-1 and N-1 referenced relationships

1-1 and N-1 referenced relationships are expressed using the *LINK* type.

```
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);
```

In this case records of class "Invoice" will link to a record of class "Customer" using the field "customer".

1-N and N-M referenced relationships

1-N and N-M referenced relationships are expressed using the collection of links such as:

- **LINKLIST** as an ordered list of links
- **LINKSET** as an unordered set of links. It doesn't accept duplicates
- **LINKMAP** as an ordered map of links with *String* key. It doesn't accept duplicated keys

Example of a 1-N 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("items", OType.LINKLIST, orderItem);

db.getMetadata().getSchema().save();
```

Embedded relationships

Embedded records, instead, are contained inside the record that embeds them. It's a kind of relationship stronger than the [#Referenced_relationships reference]. The embedded record will not have its own [Concepts#RecordID RecordID] since it can't be directly referenced by other records. It's only accessible via the container record. If the container record is deleted, then the embedded record will be deleted too. Example:

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

Record A will contain the entire *Record B* in the property called "address". *Record B* can be reached only by traversing the container record.

Example:

```
SELECT FROM account WHERE address.city = 'Rome'
```

1-1 and N-1 embedded relationships

1-1 and N-1 embedded relationships are expressed using the *EMBEDDED* type.

```
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);
```

In this case, records of class "Account" will embed a record of class "Address".

1-N and N-M embedded relationships

1-N and N-M embedded relationships are expressed using the collection of links such as:

- **EMBEDDEDLIST**, as an ordered list of records.
- **EMBEDDEDSET**, as an unordered set of records. It doesn't accepts duplicates.
- **EMBEDDEDMAP**, as an ordered map with records as the value and *String* as the key. It doesn't accept duplicate keys.

Example of a 1-N relationship between the class 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);
```

Constraints

OrientDB supports a number of constraints for each field:

- **Minimum value**, accepts a string because it also works for date ranges `setMin()`
- **Maximum value**, accepts a string because it also works for date ranges `setMax()`
- **Mandatory**, must be specified `setMandatory()`
- **Readonly**, may not be updated after record is created `setReadonly()`
- **Not Null**, cannot be NULL `setNotNull()`
- **Unique**, doesn't allow duplicates and speeds up searches.
- **Regexp**, must satisfy the Regular expression.

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");
```

Indexes as constraints

To ensure that a property value is UNIQUE use the UNIQUE index as a constraint:

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

To ensure that a group of properties is UNIQUE create a composite index made of multiple fields: Example of creating a composite index:

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

For more information about indexes look at 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. Today only the equals condition is supported.

Example:

```
employees[label = 'Ferrari']
```

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.

Future directions

In the future you will be able to use the full expression of the OrientDB SQL language inside the square brackets [], like:

```
SELECT out[in.label.trim() = 'Ferrari' AND in.@class='Vehicle'] FROM v WHERE name = 'Tom'
```

But for this you have to wait yet :-) Monitor the issue: https://github.com/nuvolabase/orientdb/issues/513

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.

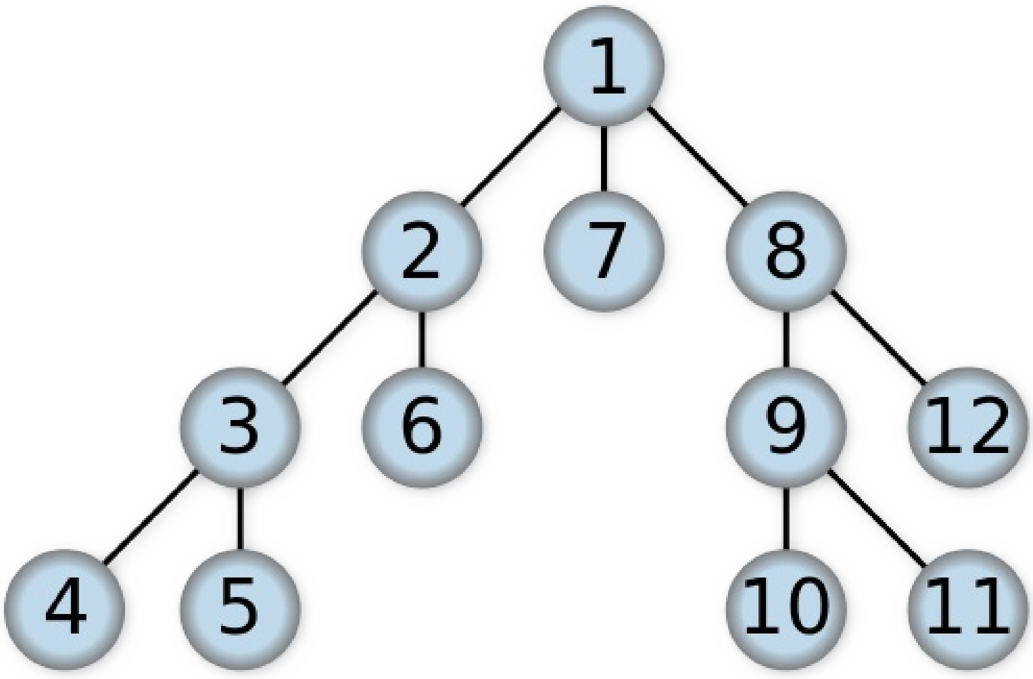
The main concepts of Traversal are:

- **target**, as the starting point where to traverse records. Can be:
  - **class**
  - **cluster**
  - **set of records**, specifying its RecordID
  - **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

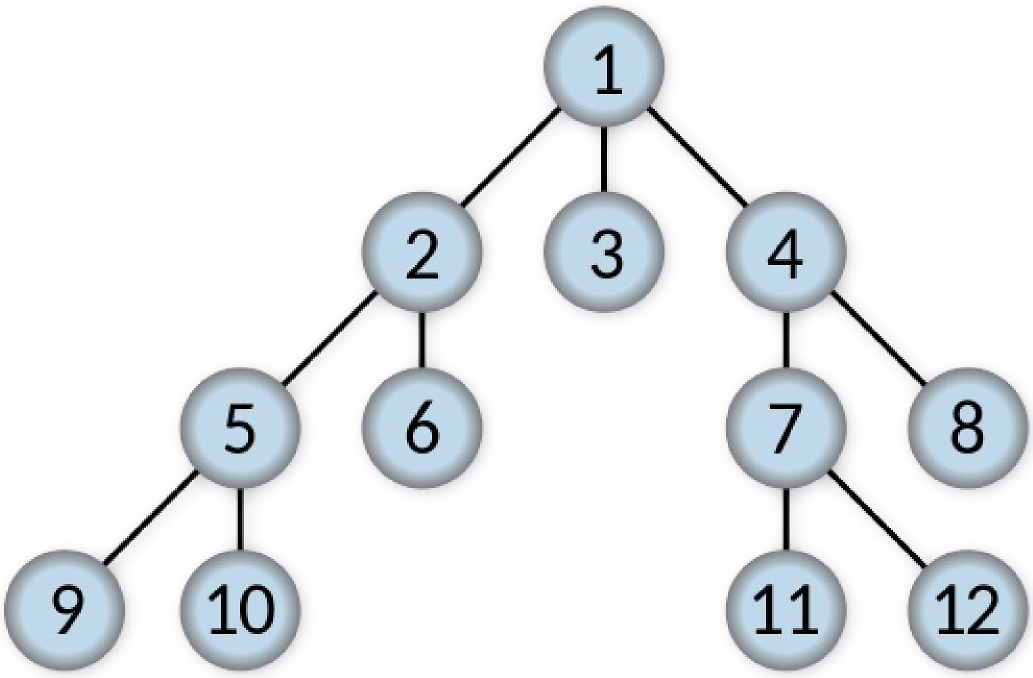
DEPTH_FIRST strategy

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", "int")
              .limit(100)
              .predicate( new OSQLPredicate("$depth <= 10"))) {

  System.out.println( id);
}
```

Live Query

(Since 2.1)

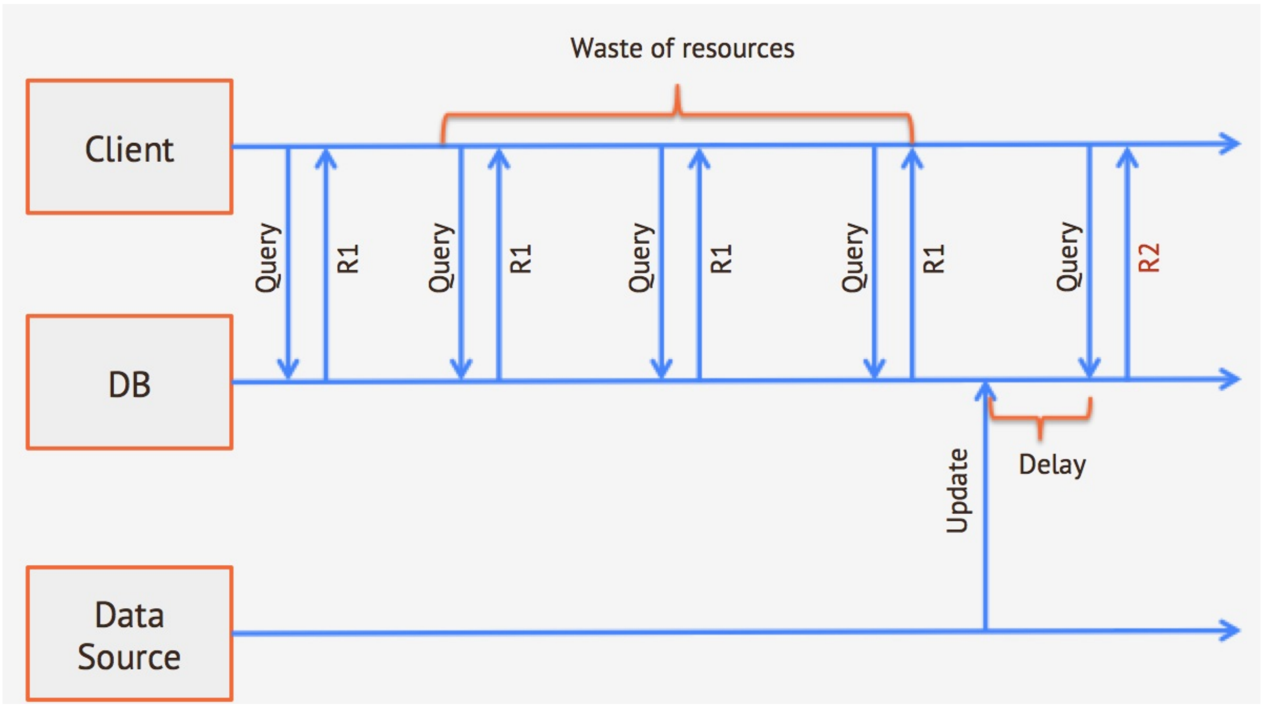
Writing **realtime, reactive applications** is hard task with traditional query paradigm. Think about a simple use case like updating a web page with fresh data coming from the database and keeping it updated over time; also consider that updates can be made by different data sources (multiple applications, 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

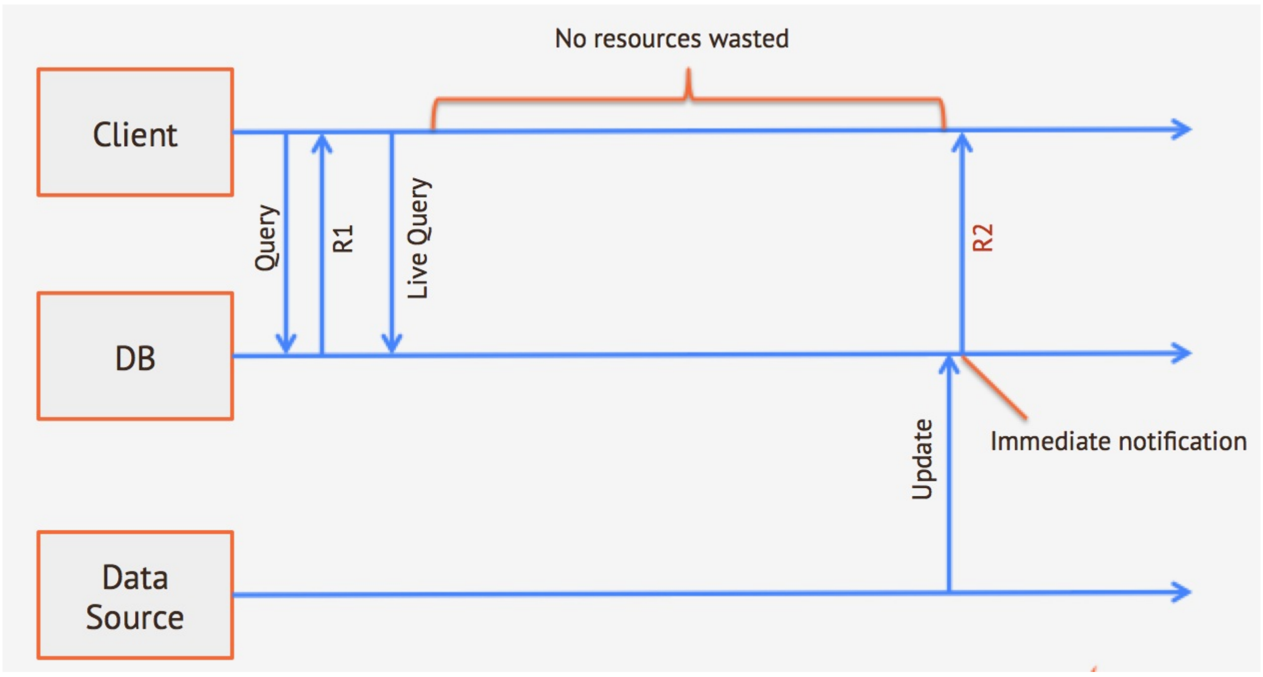


You have to make a choice here

- 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, from 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});
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.

Multi-Threading

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*. 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.

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

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

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.

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 2.1.21)

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();
```

JPA

There are two ways to configure OrientDB JPA

Configuration

The first - do it through /META-INF/persistence.xml Folowing 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 not 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 RecordId 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 RecordId 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');
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.

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; exi
t
```

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:

```
<!-- 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" />
    </parameters>
  </handler>
```

NOTE: this will allow clients to execute any code inside the server. Enable it only if clients are trusted.

Javascript API

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
}
```

Scala API

OrientDB is a NoSQL database writen in Java, we can use it in scala easily. Look also at Scala utilities and tests project for Scala high level classes built on top of OrientDB.

using SBT

Use the following configuration:

```
fork := true
```

Java method invocation problems

Usually the main problems are related to calling conventions between Scala and Java.

Parameters

Be careful to pass parameters to methods with varargs like `db.query(...)`. You need to convert it to java's repeated args correctly.

Look at these links:

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

Collections

You can only use java collections when define pojos. If you use scala collections, they can be persisted, but can't be queried.

This's not a problem, if you imported:

```
import scala.collection.JavaConverters._
import scala.collection.JavaConversions._
```

You don't need to convert Java and Scala collections manually (even don't need to invoke `.asJava` or `.asScala`) You can treat these java collections as scala's.

models.scala

```scala
package models

import javax.persistence.{Version, Id}

class User {
    @Id var id: String = _
    var name: String = _
    var addresses: java.util.List[Address] = new java.util.ArrayList()
    @Version var version: String = _

    override def toString = "User: " + this.id + ", name: " + this.name + ", addresses: " + this.addresses
}

class Address {
    var city: String = _
    var street: String = _

    override def toString = "Address: " + this.city + ", " + this.street
}

class Question {
    @Id var id: String = _
    var title: String = _
    var user: User = _
    @Version var version: String = _

    override def toString = "Question: " + this.id + ", title: " + this.title + ", belongs: " + user.name
}
```

test.scala

```scala
package models

import com.orientechnologies.orient.core.id.ORecordId
import com.orientechnologies.orient.core.sql.query.OSQLSynchQuery
import scala.collection.JavaConverters._
import scala.collection.JavaConversions._
import com.orientechnologies.orient.`object`.db.{OObjectDatabaseTx,OObjectDatabasePool}
import com.orientechnologies.orient.core.db.`object`.ODatabaseObject

object Test {
    implicit def dbWrapper(db: OObjectDatabaseTx) = new {
        def queryBySql[T](sql: String, params: AnyRef*): List[T] = {
            val params4java = params.toArray
            val results: java.util.List[T] = db.query(new OSQLSynchQuery[T](sql), params4java: _*)
            results.asScala.toList
        }
    }

    def main(args: Array[String]) = {
        // ~~~~~~~~~~~~~ create db ~~~~~~~~~~~~~~~~~~~
        var uri: String = "plocal:test/orientdb"
        var db: OObjectDatabaseTx = new OObjectDatabaseTx(uri)
        if (!db.exists) {
            db.create()
        } else {
            db.open("admin", "admin")
        }

        // ~~~~~~~~~~~~ register models ~~~~~~~~~~~~~~~~
        db.getEntityManager.registerEntityClasses("models")

        // ~~~~~~~~~~~~~ create some data ~~~~~~~~~~~~~~~~
        var user: User = new User
        user.name = "aaa"
        db.save(user)

        var address1 = new Address
        address1.city = "NY"
        address1.street = "road1"
        var address2 = new Address
        address2.city = "ST"
```

```
        address2.street = "road2"

        user.addresses += address1
        user.addresses += address2
        db.save(user)

        var q1 = new Question
        q1.title = "How to use orientdb in scala?"
        q1.user = user
        db.save(q1)

        var q2 = new Question
        q2.title = "Show me a demo"
        q2.user = user
        db.save(q2)

        // ~~~~~~~~~~~~~~~~ count them ~~~~~~~~~~~~~~~~
        val userCount = db.countClass(classOf[User])
        println("User count: " + userCount)

        val questionCount = db.countClass(classOf[Question])
        println("Question count: " + questionCount)

        // ~~~~~~~~~~~~~~~~~ get all users ~~~~~~~~~~~~
        val users = db.queryBySql[User]("select from User")
        for (user <- users) {
            println(" - user: " + user)
        }

        // ~~~~~~~~~~~~~~~~~~ get the first user ~~~~~~~~
        val firstUser = db.queryBySql[User]("select from User limit 1").head
        println("First user: " + firstUser)

        // query by id
        val userById = db.queryBySql[User]("select from User where @rid = ?", new ORecordId(user.id))
        println("User by id: " + userById)

        // query by field
        val userByField = db.queryBySql[User]("select from User where name = ?", user.name)
        println("User by field: " + userByField)

        // query by city
        val userByCity = db.queryBySql[User]("select from User where addresses contains ( city = ? )", "NY")
        println("User by city: " + userByCity)

        // query questions of the user
        val questions = db.queryBySql[Question]("select from Question where user = ?", user)
        for (q <- questions) {
            println(" - question: " + q)
        }

        db.drop()
        db.close()
    }
}
```

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)

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>`

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: RecordID
- `<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: RecordID

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/<index-name>/<key>`

Example

HTTP GET request: `http://localhost:2480/dictionary/test` HTTP response:

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

PUT - Index

Create or modify an index entry.

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

Example

HTTP PUT request: `http://localhost:2480/dictionary/test` content: `{ "name" : "Jay", "surname" : "Miner" }`

HTTP response: No response.

DELETE - Index

Remove an index entry.

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

Example

HTTP DELETE request: `http://localhost:2480/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
```

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.

Table of content

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	Async
Server *(CONNECT Operations)*			
REQUEST_SHUTDOWN	1	Shut down server.	no
REQUEST_CONNECT	2	Required initial operation to access to server commands.	no
REQUEST_DB_OPEN	3	**Required initial operation** to access to the database.	no
REQUEST_DB_CREATE	4	Add a new database.	no
REQUEST_DB_EXIST	6	Check if database exists.	no
REQUEST_DB_DROP	7	Delete database.	no
REQUEST_CONFIG_GET	70	Get a configuration property.	no
REQUEST_CONFIG_SET	71	Set a configuration property.	no
REQUEST_CONFIG_LIST	72	Get a list of configuration properties.	no
REQUEST_DB_LIST	74	Get a list of databases.	no
Database *(DB_OPEN Operations)*			
REQUEST_DB_CLOSE	5	Close a database.	no
REQUEST_DB_SIZE	8	Get the size of a database (in bytes).	no
REQUEST_DB_COUNTRECORDS	9	Get total number of records in a database.	no
REQUEST_DATACLUSTER_ADD (deprecated)	10	Add a data cluster.	no
REQUEST_DATACLUSTER_DROP (deprecated)	11	Delete a data cluster.	no
REQUEST_DATACLUSTER_COUNT (deprecated)	12	Get the total number of data clusters.	no
REQUEST_DATACLUSTER_DATARANGE (deprecated)	13	Get the data range of data clusters.	no

REQUEST_DATACLUSTER_COPY	14	Copy a data cluster.	no
REQUEST_DATACLUSTER_LH_CLUSTER_IS_USED	16		no
REQUEST_RECORD_METADATA	29	Get metadata from a record.	no
REQUEST_RECORD_LOAD	30	Load a record.	no
REQUEST_RECORD_LOAD_IF_VERSION_NOT_LATEST	44	Load a record.	no
REQUEST_RECORD_CREATE	31	Add a record.	yes
REQUEST_RECORD_UPDATE	32		yes
REQUEST_RECORD_DELETE	33	Delete a record.	yes
REQUEST_RECORD_COPY	34	Copy a record.	yes
REQUEST_RECORD_CLEAN_OUT	38	Clean out record.	yes
REQUEST_POSITIONS_FLOOR	39	Get the last record.	yes
REQUEST_COUNT *(DEPRECATED)*	40	See REQUEST_DATACLUSTER_COUNT	no
REQUEST_COMMAND	41	Execute a command.	no
REQUEST_POSITIONS_CEILING	42	Get the first record.	no
REQUEST_TX_COMMIT	60	Commit transaction.	no
REQUEST_DB_RELOAD	73	Reload database.	no
REQUEST_PUSH_RECORD	79		no
REQUEST_PUSH_DISTRIB_CONFIG	80		no
REQUEST_PUSH_LIVE_QUERY	81		no
REQUEST_DB_COPY	90		no
REQUEST_REPLICATION	91		no
REQUEST_CLUSTER	92		no
REQUEST_DB_TRANSFER	93		no
REQUEST_DB_FREEZE	94		no
REQUEST_DB_RELEASE	95		no
REQUEST_DATACLUSTER_FREEZE (deprecated)	96		no
REQUEST_DATACLUSTER_RELEASE (deprecated)	97		no
REQUEST_CREATE_SBTREE_BONSAI	110	Creates an sb-tree bonsai on the remote server	no
REQUEST_SBTREE_BONSAI_GET	111	Get value by key from sb-tree bonsai	no
REQUEST_SBTREE_BONSAI_FIRST_KEY	112	Get first key from sb-tree bonsai	no
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	no
REQUEST_RIDBAG_GET_SIZE	114	Rid-bag specific operation. Send but does not save changes of rid bag. Retrieves computed size of rid bag.	no
REQUEST_INDEX_GET	120	Lookup in an index by key	no
REQUEST_INDEX_PUT	121	Create or update an entry in an index	no
REQUEST_INDEX_REMOVE	122	Remove an entry in an index by key	no

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 ass 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)(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.
- **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)(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.
- **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)
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

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. The TCP socket is also closed.

```
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 existance. 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 RecordID, 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 RecordID 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 RecordID, 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 RecordID 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 RecordID of the newly created record.. New records can have version > 0 (since `1.0`) in case the RecordID 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 RecordID 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 RecordID. 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 RecordID 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 RecordID. 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 RecordID 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 RecordID 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
    - 'a', serialized result, a byte[] is sent
    - '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 RecordID 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 RecordID 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 RecordID 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 RecordIDs 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_LIVE_QUERY

```
(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 happend, 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 postion
record-content record content

History

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 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)
- **RecordID** (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")>` . 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
- SHORT
- INTEGER
- LONG
- BYTE
- DATE
- DATETIME
- DECIMAL
- FLOAT
- DOUBLE

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:int64)(record-position:int64)
```

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
- SHORT
- INTEGER
- LONG
- BYTE
- DATE
- DATETIME
- DECIMAL
- FLOAT
- DOUBLE

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

No documentation yet. :(

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 it's 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.

Many SQL commands share the WHERE condition. Keywords and class names in OrientDB SQL are case insensitive. Field names 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 you to get the course on 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

OrientDB SQL dialect

OrientDB supports SQL as a query language with some differences compared with SQL. Orient Technologies 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

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

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.
- **`FROM`** Designates the object to query. This can be a class, cluster, single Recprd 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>`. 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`:

  DEPRECIATED 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
```

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.

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. You can use any valid SQL expression. The most common use-cases,
  - `@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.

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.

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 - `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>]
  }*
RETURN <alias> [, <alias>]*
LIMIT <number>
```

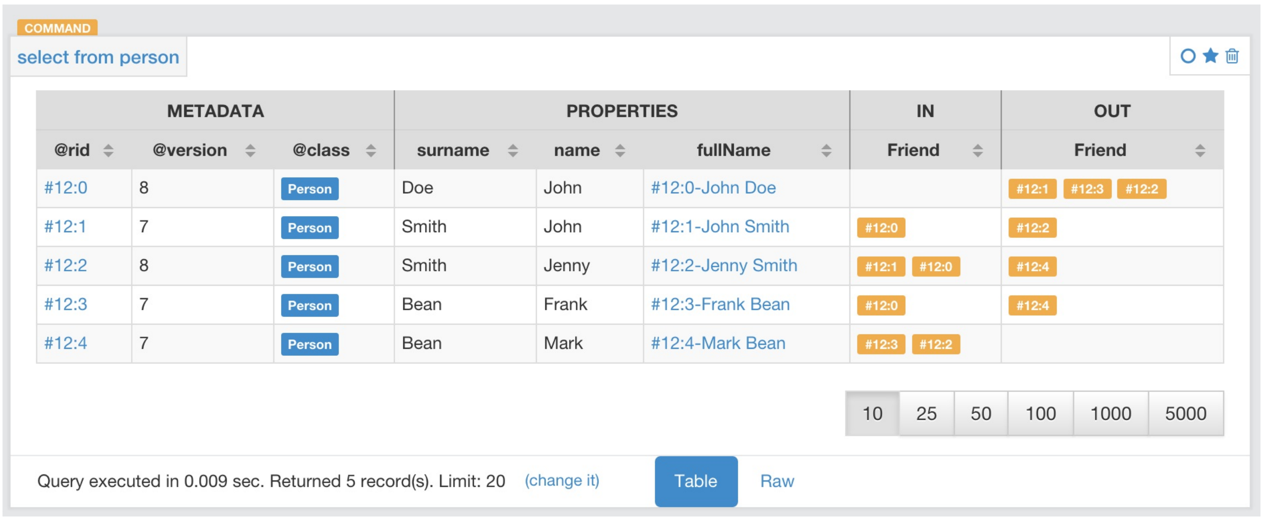
- **`<class>`** Defines a valid target class.
- **`as: <alias>`** Defines an alias for a node in the pattern.
- **`<whereCondition>`** Defines a filter condition to match a node in the pattern. It supports the normal SQL `WHERE` clause. You can also use the `$currentMatch` and `$matched` context variables.
- **`<functionName>`** Defines a graph function to represent the connection between two nodes. For instance, `out()` , `in()` , `outE()` , `inE()` , etc.
- **`<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 `$depth` context variables. For more information, see Deep Traversal While Condition, below.
- **`<maxDepth>`** Defines the maximum depth for this single path.
- **`RETURN <alias>`** Defines elements in the pattern that you want returned. It can use one of the following:
  - Aliases defined in the `as:` block.
  - `$matches` Indicating all defined aliases.
  - `$paths` Indicating the full traversed paths.

BNF Syntax

```
MatchStatement     := ( <MATCH> MatchExpression ( <COMMA> MatchExpression )* <RETURN> Identifier ( <COMMA> Identifier )* ( Lim
it )? )

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` :

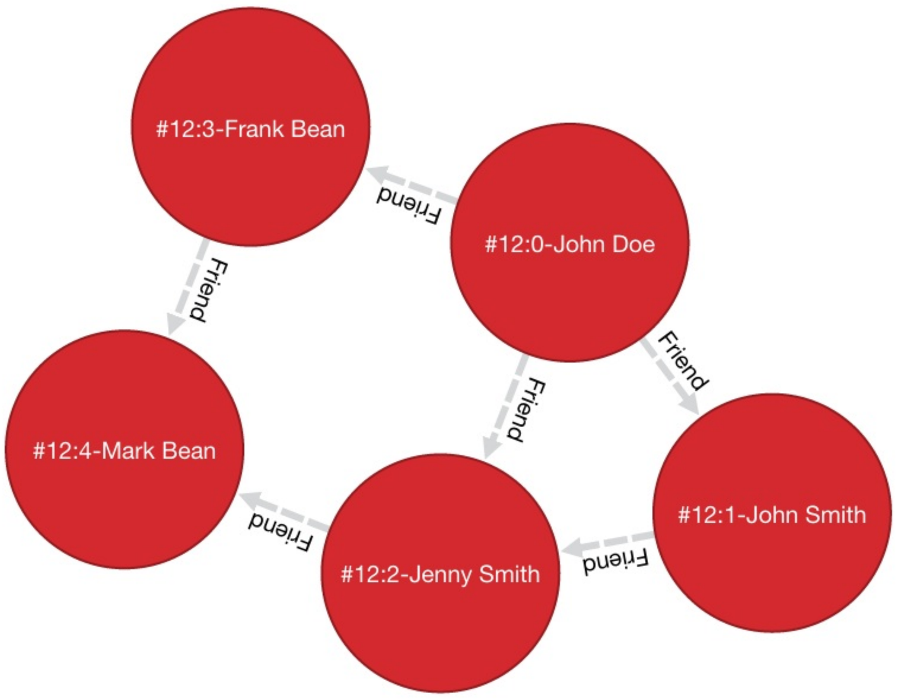


COMMAND

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	

Query executed in 0.009 sec. Returned 5 record(s). Limit: 20 (change it)



- 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

--------+---------
 people | 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

--------+----------------
 people | 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

--------+----------------
 people | 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

--------+---------
 people | 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 aslo 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

--------+--------
 people | 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
--------+----------+------------+---------------
```

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

---------
 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  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)
```

SQL Commands

CRUD	Graph	Schema	Indexes	Database	Utility
SELECT	CREATE VERTEX	CREATE CLASS	CREATE INDEX	CREATE CLUSTER	CREATE LINK
INSERT	CREATE EDGE	ALTER CLASS	REBUILD INDEX	ALTER CLUSTER	FIND REFERENCES
UPDATE	DELETE VERTEX	DROP CLASS	DROP INDEX	DROP CLUSTER	EXPLAIN
DELETE	DELETE EDGE	CREATE PROPERTY		ALTER DATABASE	GRANT
TRAVERSE	MATCH	ALTER PROPERTY		CREATE DATABASE (console only)	REVOKE
TRUNCATE CLASS		DROP PROPERTY		DROP DATABASE (console only)	CREATE FUNCTION
TRUNCATE CLUSTER				OPTIMIZE DATABASE	
TRUNCATE RECORD					

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.
`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 unqiue 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` .
`DEFAULT`			Defines the default value or function. Feature introduced in version 2.1, (see the section above for examples).

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.

> 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> [EXTENDS <super-class>] [CLUSTER <cluster-id>*] [CLUSTERS <total-cluster-number>] [ABSTRACT]
```

- **`<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.
- **`<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>]* TO <rid>|(<query>)|[<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 `OCommandExecutionException` 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 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, edge creation can sometimes be done in two steps, (that is, create and update). This can break some constraints defined in the Edge at the class-level. To avoid such problems, disable the constraints in the Edge class. Bear in mind that in distributed mode SB Tree index algorithm is not supported. You must set `ridBag.embeddedToSbtreeBonsaiThreashold=Integer.Max\_VALUE` to avoid replication errrors**

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.

  By default, indexes ignore null values. Queries against null values that use an index returns no entries. To index null values, see `{ ignoreNullValues: false }` as metadata.

```
orientdb> CREATE INDEX addresses ON Employee (address) NOTUNIQUE
           METADATA { ignoreNullValues : false }
```

> 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> <type> [<link-type>|<link-class>] [UNSAFE]
```

- **`<class>`** Defines the class for the new property.
- **`<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.
- **`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
  ```

> For more information, see
>
> - `DROP PROPERTY`
> - SQL Commands
> - Console Commands

Supported Types

OrientDB supports the following data types for standard properties:

`BOOLEAN`	`SHORT`	`DATE`	`BYTE`
`INTEGER`	`LONG`	`STRING`	`LINK`
`DOUBLE`	`FLOAT`	`BINARY`	`EMBEDDED`

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 many 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.

|----|----| | | **NOTE**: When using a distributed database, you can create vertexes through two steps (creation and update). Doing so can break constraints defined at the class-level for vertices. To avoid these issues, disable constraints in the vertex class.|

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 - `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 - `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 - `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 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.

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.

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> [ UNSAFE ]
```

- **`<class>`** Defines the class you want to remove.
- **`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.

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> [FORCE]
```

- **`<class>`** Defines the class where the property exists.
- **`<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 - `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.* TO 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 - `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.

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 htird level, using the `BREADTH_FIRST` strategy:

```
orientdb> TRAVERSE out("Friend") FROM #10:1234 WHILE $depth <= 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 WHILE $depth <= 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 WHILE $depth <= 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") WHILE $depth <= 3) WHERE
            @class = 'Movie'
  ```

- Display the current path in the traversal:

  ```
  orientdb> SELECT $path FROM ( TRAVERSE out() FROM V WHILE $depth <= 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 tot 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 WHILE
          $depth <= 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') WHILE $depth
```

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
            WHILE $depth <= 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 WHILE $depth <=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 $depth <=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 that `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 - 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 RecordID 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 RecordID 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
any	=	Equals to	name = 'Luke'
string	like	Similar to equals, but allow the wildcard '%' that means 'any'	name **like** 'Luk%'
any	<	Less than	age < 40
any	<=	Less than or equal to	age <= 40
any	>	Greater than	age > 40
any	>=	Greater than or equal to	age >= 40
any	<>	Not equals (same of !=)	age <> 40
any	BETWEEN	The value is between a range. It's equivalent to <field> >= <from-value> AND <field> <= <to-value>	price BETWEEN 10 and 30
any	IS	Used to test if a value is NULL	children **is** null
record, string (as class name)	INSTANCEOF	Used to check if the record extends a class	@this **instanceof** 'Customer' or @class **instanceof** 'Provider'
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')
collection	CONTAINSALL	true if all the elements of the collection satisfy the next condition	children *containsAll* (name = 'Luke')
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'

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
string	CONTAINSTEXT	used with 89cd72a14eb5493801e99a43c5034685. Current limitation is that it must be the unique condition of a query. 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'
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'
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')

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
$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()
outV()	avg()	distinct()	uuid()
inV()	count()	expand()	if()
traversedElement()	mode()	unionall()	
traversedVertex()	median()	flatten()	
traversedEdge()	percentile()	last()	
shortestPath()	variance()	symmetricDifference()	
dijkstra()	stddev()		

Functions by name

abs()	avg()	both()	bothE()	
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'
```

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(&lt;field&#124;value&gt;, &lt;return_value_if_null&gt; [,&lt;return_value_if_not_null&gt;](,&lt;field&.md#124;value&gt;
]*)
```

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.
```

on RIDs

```
SELECT EXPAND( addresses ) FROM Account.
```

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 parameters, 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 parameters, 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. 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
```

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
- `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, 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 also 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 than 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 than aggregates, otherwise 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 than 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()

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.

Syntax: `map(<key>, <value>)`

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>]*)`

Available since: 2.0-M1

Examples

```
SELECT percentile(salary, 95) FROM Account
SELECT percentile(salary, 25, 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
```

.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
```

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 'lenght' 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' and getting 'length' characters. 'begin' starts from 0 to string length - 1.

Syntax: `<value>.subString(<begin> [,<length>] )`

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
```

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. Example: `return [ $a, $b ]`
  - maps. Example: `return { 'first' : $a, 'second' : $b }`

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 edge = create edge Lives from $account to $city
commit retry 100
return $edge
```

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 edge = CREATE EDGE Lives FROM $account TO $city
COMMIT
return $edge
```

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.

OrientDB supports pagination natively. Pagination doesn't consume server side resources because no cursors are used. Only RecordIDs 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 RecordID
- **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));
```

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 from release 2.2, OrientDB supports caching of commands results. Caching command results has been used by other DBMSs and proven to dramatically improve the following use cases:

- database is mostly read than write
- there are a few heavy queries that result a small result set
- you have available RAM to use or caching results

By default, the command cache is disabled. To enable it, set `command.timeout=true`.

Settings

There are some settings to tune the command cache. Below find the table 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 could 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 then **INVALIDATE_ALL**

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.
- **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.

- **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

By default, indexes do not support null values. Queries made against `NULL` values that use indexes fail with the `Null keys are not supported` exception.

In the event that you want to index null values, you must set `{ignoreNullValues: false}` in the metadata. For instance,

```
orientdb> CREATE INDEX addresses ON Employee (address) NOTUNIQUE METADATA {ignoreNullValues: false}
```

Indexes and Composite Keys

Operations that work with indexes also work with indexes formed from composite keys. By its nature, a composite key is acollection 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 a public up through 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 equivalent,

```
orientdb> SELECT FROM INDEX:books WHERE key = "Donald Knuth"
```

Insertion for Composite Indexes

❗	Direct insertion for composite indexes is not yet supported in OrientDB.

Range Queries

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. Thisclass 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 teh 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.

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:

```
CREATE INDEX <name> ON <class-name> (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
```

Analyzer

This creates a basic FullText Index with the Lucene Engine on the specified properties. In the even that you do not specify the analyzer, OrientDB defaults to StandardAnalyzer.

In addition to the StandardAnalyzer, you can also create indexes that use different analyzer, using the `METADATA` operator through `CREATE INDEX`.

```
orientdb> CREATE INDEX City.name ON City(name) FULLTEXT ENGINE LUCENE METADATA
          {"analyzer": "org.apache.lucene.analysis.en.EnglishAnalyzer"}
```

(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_analyzer": "org.apache.lucene.analysis.en.EnglishAnalyzer",
          "query_analyzer": "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.

It is posssbile to configure analyzers at field level

```
orientdb> CREATE INDEX City.name_description ON City(name, description) FULLTEXT ENGINE LUCENE METADATA
          {
          "index_analyzer": "org.apache.lucene.analysis.en.EnglishAnalyzer",
          "query_analyzer": "org.apache.lucene.analysis.standard.StandardAnalyzer",
          "name_index_analyzer": "org.apache.lucene.analysis.standard.StandardAnalyzer",
          "name_query_analyzer": "org.apache.lucene.analysis.core.KeywordAnalyzer"
          }
```

With this configuration **name** will be indexed with StandardAnalyzer and query will be analyzed with the KeywordAnalyzer: **description** hasn't a custom configuration, so default analyzers for indexing an querying will be used.

You can also use the FullText Index with the Lucene Engine 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"});
```

Querying Lucene FullText Indexes

You can query the Lucene FullText Index using the custom operator `LUCENE` with the [Query Parser Synta]x(http://lucene.apache.org/core/5_4_1/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description) 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`.

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 Class WHERE [prop1, prop2] LUCENE "(prop1:foo AND prop2:bar)"
```

Here, hte engine parses the query using the QueryParser

Creating a Manual Lucene Index

Beginning with version 2.1, 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"
```

Lucene Spatial

For now the Index Engine can only index Points. Other Shapes like rectangles and polygons will be added in the future.

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.

Future Plans

In OrientDB 2.2 a new Spatial-Module will replace this implementation with:

- GeoSpatial standard (ST_*) functions
- Index All types of shape

Spatial Module

(Versions 2.2 and after only)Replacement for Spatial-Index

OrientDB offers a brand new module to handle geospatial information provided as external plugin.

Install

Download the plugin jar from maven central

```
http://central.maven.org/maven2/com/orientechnologies/orientdb-spatial/VERSION/orientdb-spatial-VERSION-dist.jar
```

where **VERSION** must be the same of the OrientDB installation. After download, copy the jar to orient lib directory. On *nix system it could be done this way:

```
wget  http://central.maven.org/maven2/com/orientechnologies/orientdb-spatial/VERSION/orientdb-spatial-VERSION-dist.jar
cp orientdb-spatial-VERSION-dist.jar /PATH/orientdb-community-VERSION/lib/
```

Orient db will load the spatial plugin on startup.

Geometry Data

OrientDB supports the following Geometry objects :

- Point
- Line
- Polygon
- MultiPoint
- MultiLine
- MultiPolygon
- 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();
```

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)

Functions

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_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
```

Install

2.2.0-SNAPSHOT

The module has been merged into the main repository branch develop

- Take the latest OrientDB 2.2.0-Snapshot here

Or

build the develop branch from scratch

2.2 GA

This module is part of orientdb-lucene plugin and will be included in OrientDB 2.2 GA

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 to manage the clustering. Many of the references in this page are linked to the Hazelcast official documentation to get more information about such topic.

Presentation

Main topics

- Distributed Architecture Lifecycle
- Configure the Cluster of servers
- Replication of databases
- Sharding
- Distributed Cache
- Tutorial to setup a distributed database
- Tuning

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.

Creation of records (documents, vertices and edges)

In distributed mode the RID is assigned with cluster locality. If you have class `Customer` and 3 nodes (node1, node2, node3), you'll have these clusters:

- `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.

Distributed transactions

Starting from v1.6, 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.

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. I 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.

Limitations

OrientDB v2.1.x has some limitations you should notice when you work in Distributed Mode:

- in memory database is not supported
- `hotAlignment:true` could bring the database status as inconsistent. Please set it always to 'false', the default
- 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
- If an error happen during CREATE RECORD, the operation is fixed across the entire cluster, but some node could have a wrong RID upper bound (the created record, then deleted as fix operation). In this case a new database deploy operation must be executed
- 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
- 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
- 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)
- Not complete merging of results for all the projections when running on sharder configuration. Some functions like AVG() doesn't work on map/reduce

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,t he 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`.

```
{
   "members":[
   { "name":"node1384015873680",
    "listeners": [
      { "protocol": "ONetworkProtocolBinary",
        "listen": "192.168.1.179:2425" },
      { "protocol": "ONetworkProtocolHttpDb",
        "listen": "192.168.1.179:2481"}
    ],
    "id": "3bba4280-b285-40ab-b4a0-38788691c4e7",
    "startedOn": "2013-11-09 17:51:13",
    "databases": []
   },
   { "name":"node1383734730415",
     "listeners": [
       { "protocol":"ONetworkProtocolBinary",
         "listen":"192.168.1.179:2424" },
       { "protocol":"ONetworkProtocolHttpDb",
         "listen":"192.168.1.179:2480"}
     ],
     "id": "5cb7972e-ccb1-4ede-bfda-c835b0c2e5da",
     "startedOn": "2013-11-09 17:30:56",
     "databases": []
    }
  ],
  "localName": "_hzInstance_1_orientdb",
  "localId": "5cb7972e-ccb1-4ede-bfda-c835b0c2e5da"
}
```

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 [192.168.1.179]:2435 [orientdb] Removing Member [192.168.1.179]:2434
     [ClusterService]
INFO [192.168.1.179]:2435 [orientdb]
Members [1] {
    Member [192.168.1.179]:2435 this
}
 [ClusterService]
WARN [node1384015873680] node removed id=Member [192.168.1.179]:2434
     name=node1384014656983 [OHazelcastPlugin]
INFO [192.168.1.179]: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 /192.168.1.179:2425 (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.

Distributed Architecture Lifecycle

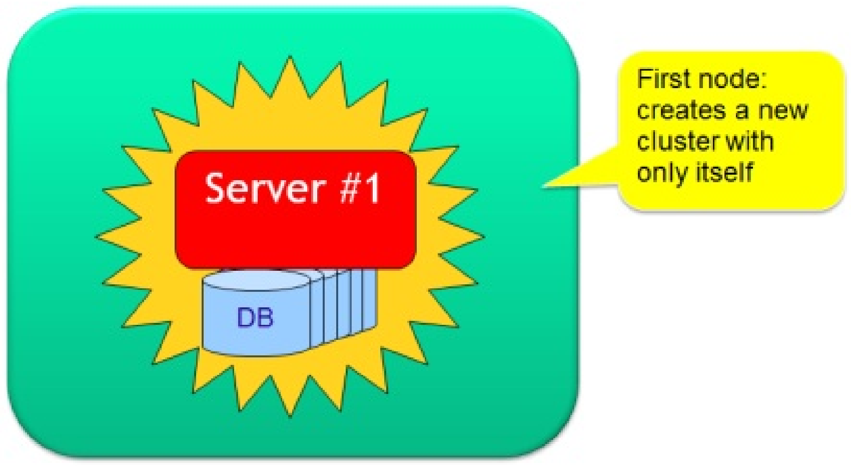
In OrientDB Distributed Architecture all the nodes are masters (Multi-Master), while in most DBMS the replication works in Master-Slave mode where there is only one Master node and N Slaves that are use only for reads or when the Master is down. Starting from OrientDB v2.1, you can also assign the role 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

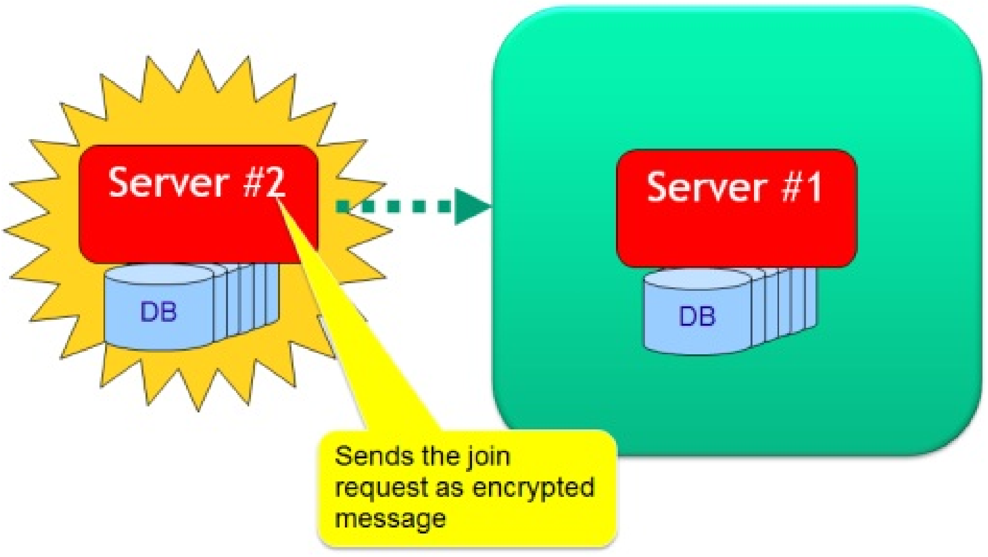
For more information look at Hazelcast documentation about configuring network.

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.



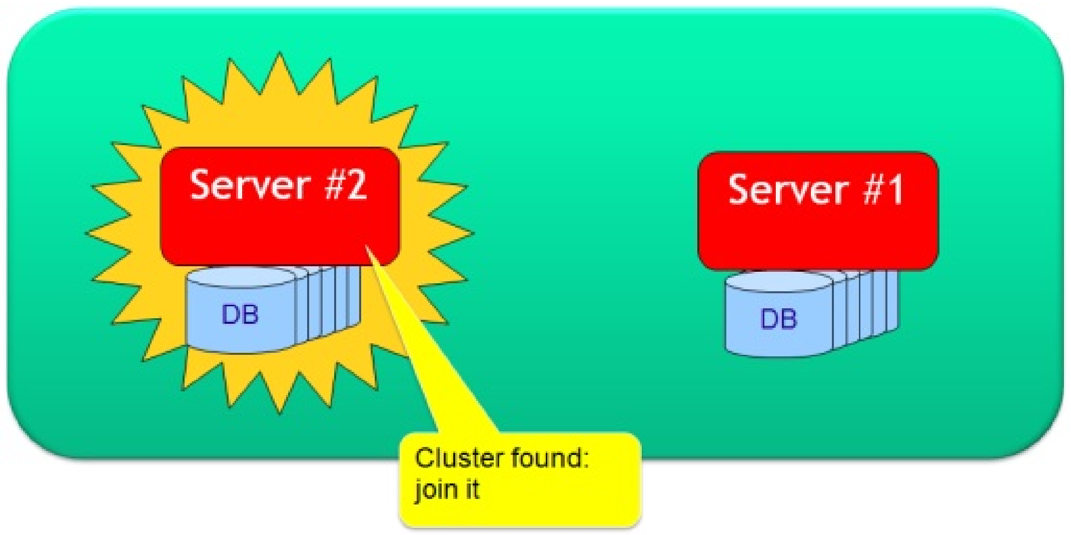
For more information look at: Hazelcast Encryption.

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>
```

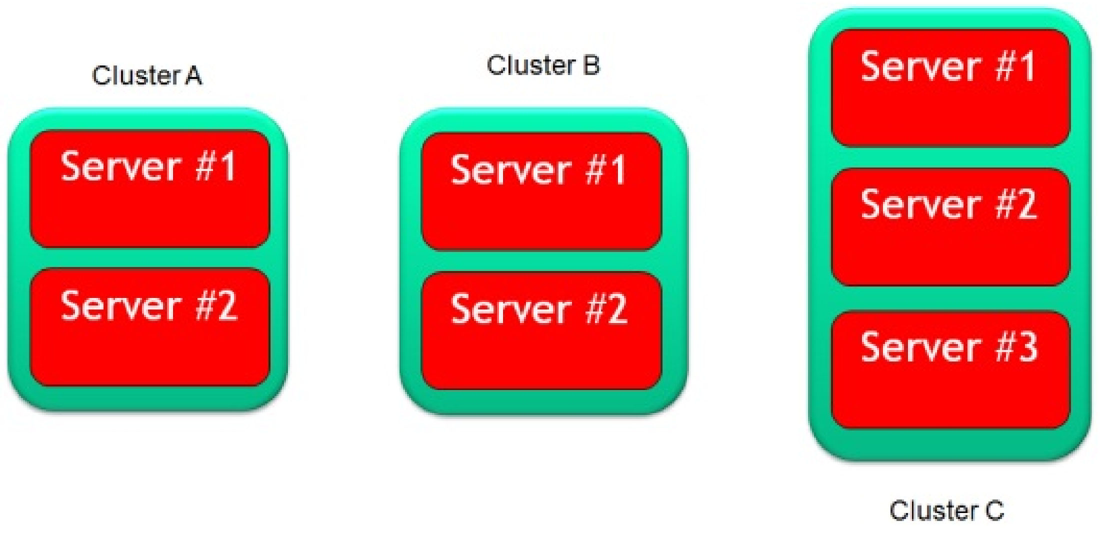
In this case Server #2 joins the existent cluster.



When a node joins an existent cluster, the most updated copy of the database is downloaded to the joining node. If any copy of database was already present, that is moved under `backup/databases` folder.

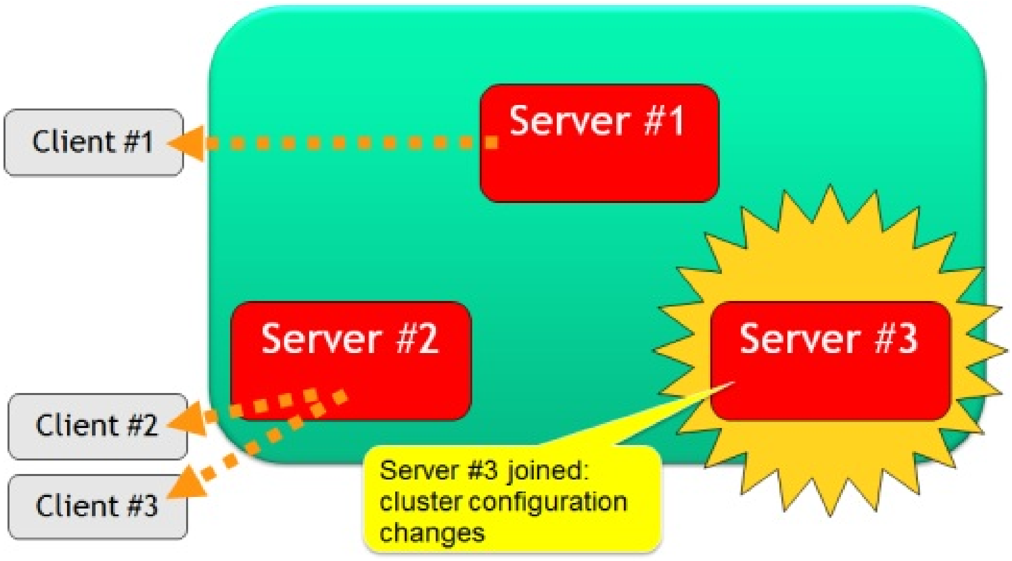
Multiple clusters

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

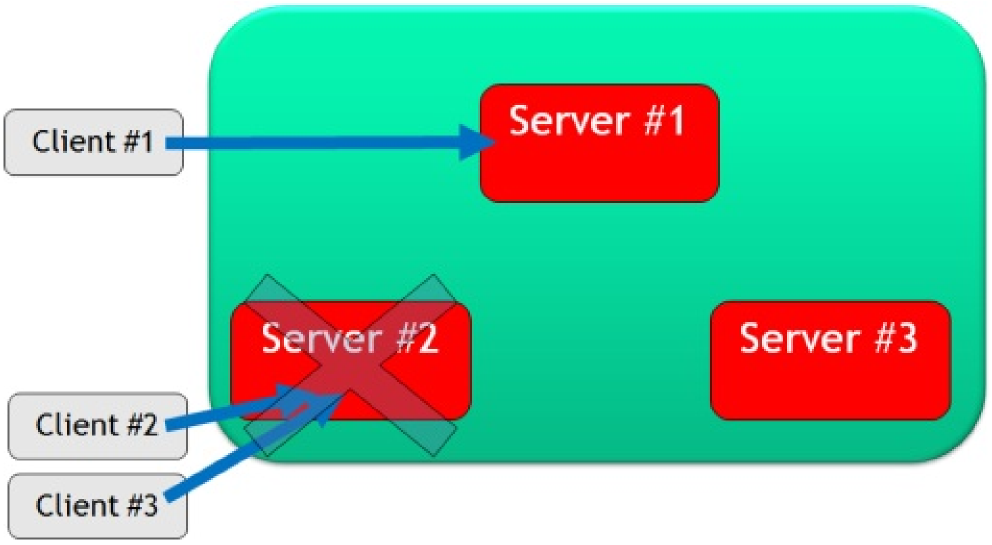
Every time a new Server Node joins or leaves the Cluster, the new Cluster configuration is broadcasted to all the connected clients. Everybody knows the cluster layout and who has a database!



Fail over management

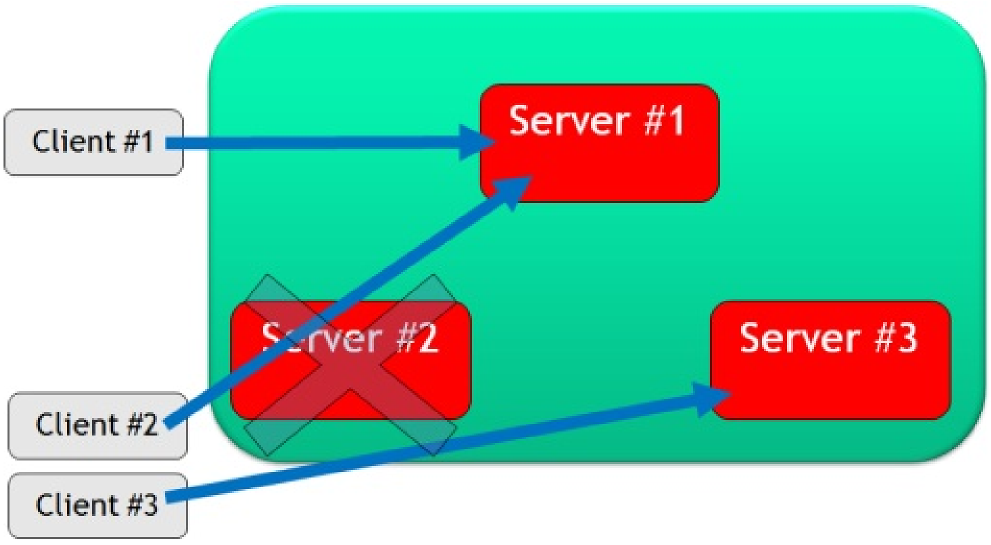
When a node is unreachable

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.



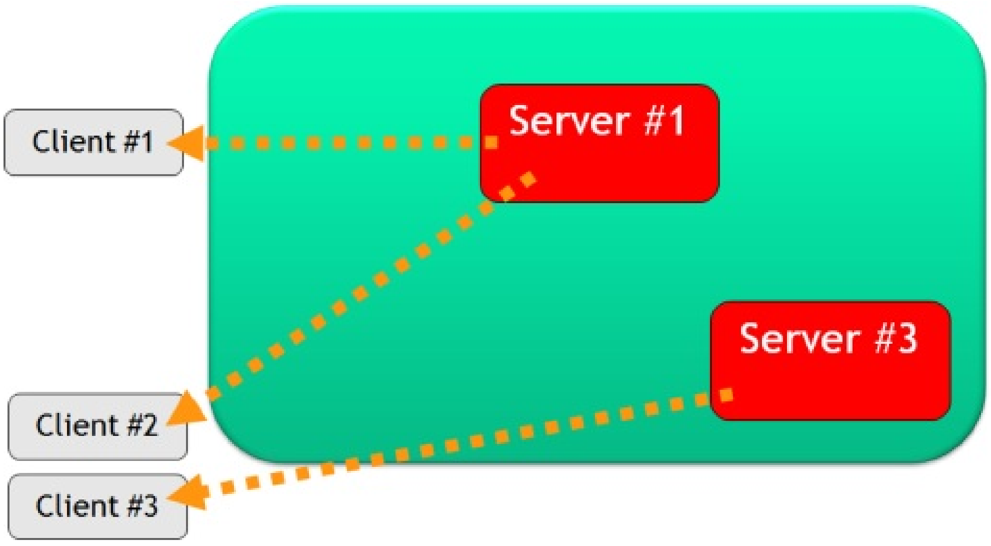
Automatic switch of servers

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
    - Cloud support

Main topics:

- Replication
- Asynchronous replication mode
- Return distributed configuration at run-time
- Load Balancing

orientdb-server-config.xml

To enable and configure the clustering between nodes, add and enable the **OHazelcastPlugin** plugin. It is configured as a Server handler. 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,
    "hotAlignment": false,
    "executionMode": "undefined",
    "readQuorum": 1,
    "writeQuorum": 2,
    "failureAvailableNodesLessQuorum": false,
    "readYourWrites": true,
    "servers": {
        "*": "master"
    },
    "clusters": {
        "internal": {
        },
        "index": {
        },
        "*": {
            "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`
hotAlignment	Whether the synchronization queue is left or not for a node leaving the cluster for hot alignment when the node joins the cluster again. It can be `true` or `false`	`false`
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 N/2+1 where N is the number of replicas. In this way the quorum is reached only if the majority of nodes are coherent	`2`
failureAvailableNodesLessQuorum	Whether to return error when the available nodes are less then quorum. Can be `true` or `false`	`false`
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`
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 N/2+1 where N is the number of replicas. In this way the quorum is reached only if the majority of nodes are coherent	`2`
failureAvailableNodesLessQuorum	Decide to return error when the available nodes are less then quorum. Can be `true` or `false`	`false`
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`
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`, `index`, because all the changes remain locally to each node (indexing is per node). 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 2 means that all the changes are in at least 2 nodes. If available nodes are less then 2, no error is given because "failureAvailableNodesLessQuorum" is false.

100% asynchronous writes

By default writeQuorum is 2. This means that it waits and checks the answer from at least 2 nodes before to send the ACK to the client. If you've more then 2 nodes configured, then starting from the 3rd node the response will be managed asynchronously. You could also set this to 1 to have all the writes asynchronous.

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 external 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-cloud.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 Config Amazon EC2 Auto Discovery.

Other Cloud providers

Uses manual IP like explained in Manual IP.

Asynchronous replication mode

In order to reduce the latency in WAN, 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,
    "hotAlignment": false,
    "executionMode": "asynchronous",
    "readQuorum": 1,
    "writeQuorum": 2,
    "failureAvailableNodesLessQuorum": false,
    "readYourWrites": true,
    "servers": {
        "*": "master"
    },
    "clusters": {
        "internal": {
        },
        "index": {
        },
        "*": {
            "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;
  }
})
 .onAsyncReplicationError(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);
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);
OrientGraphNoTx graph = factory.getNoTx();
```

Use multiple addresses

If the server addresses are known, it's 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

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`

History

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.

v2.2

Introduced Load balancing at client level. For more information look at load balancing.

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
        }
    ]
}
```

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-dserver-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>
```

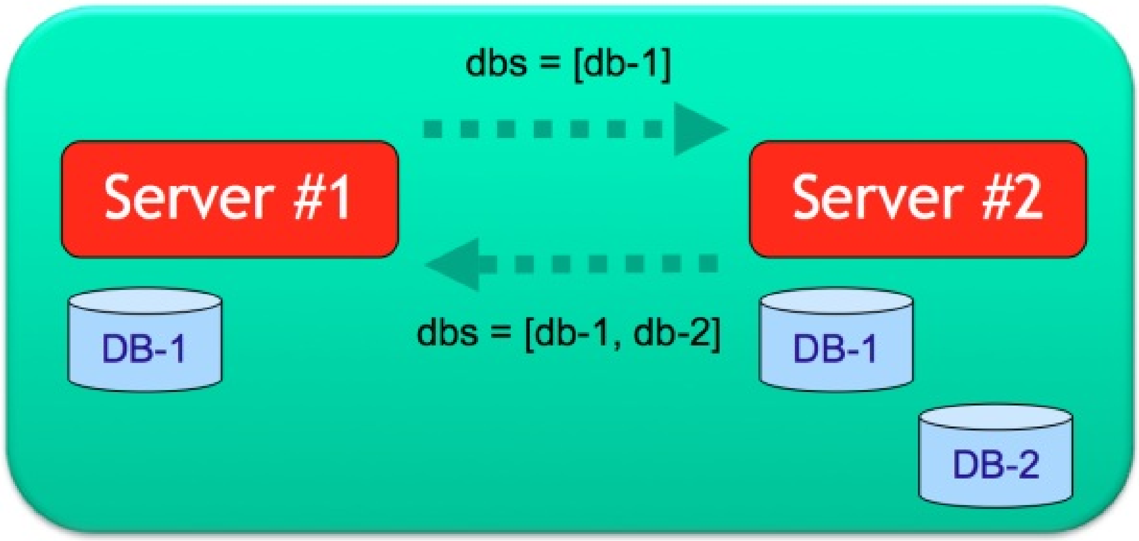
Replication

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.

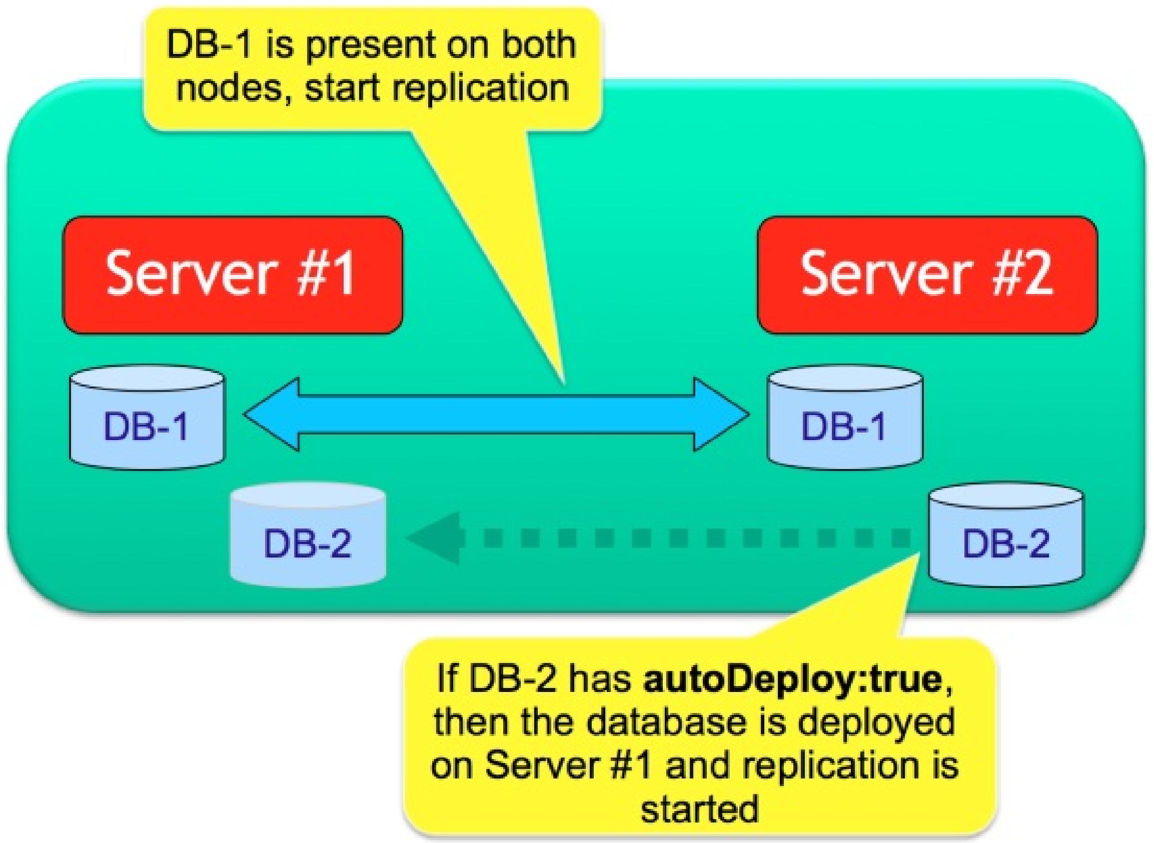
Sharing of database

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.



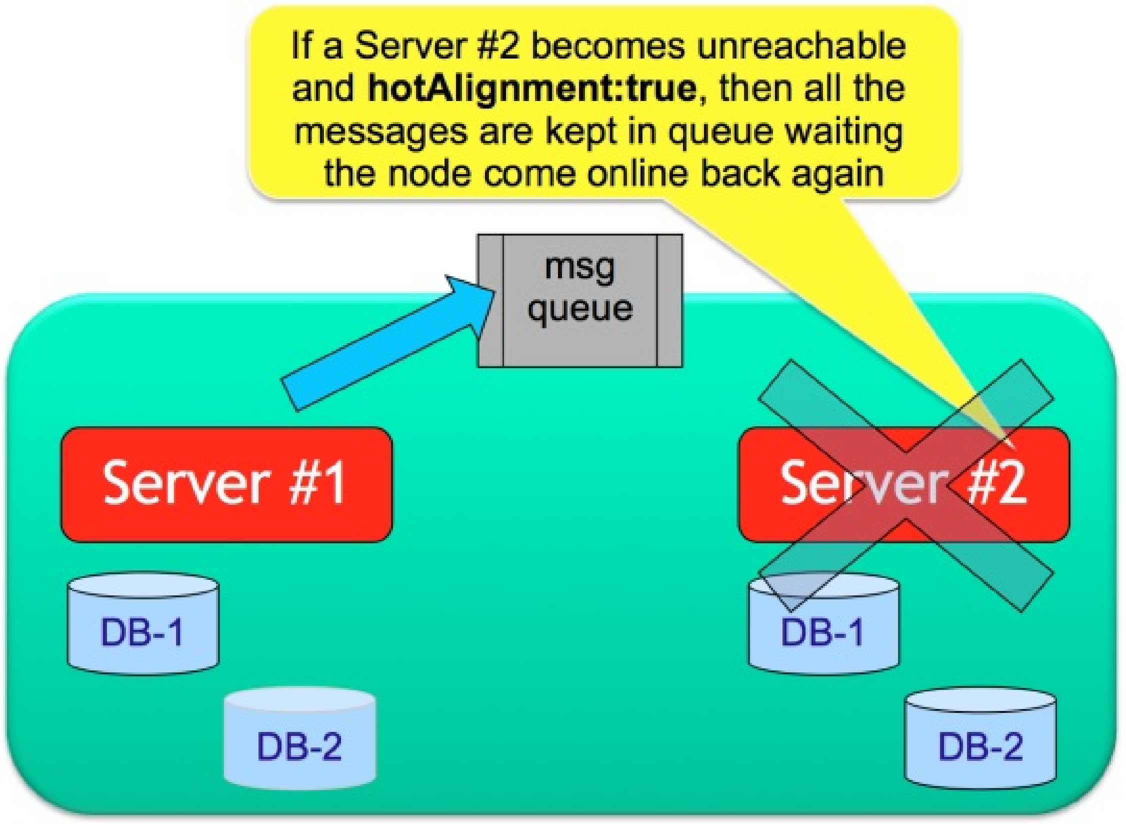
NOTE: In Distributed Architecture assure to avoid conflict with database names, otherwise 2 different databases could start replication with the chance to get corrupted.

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.

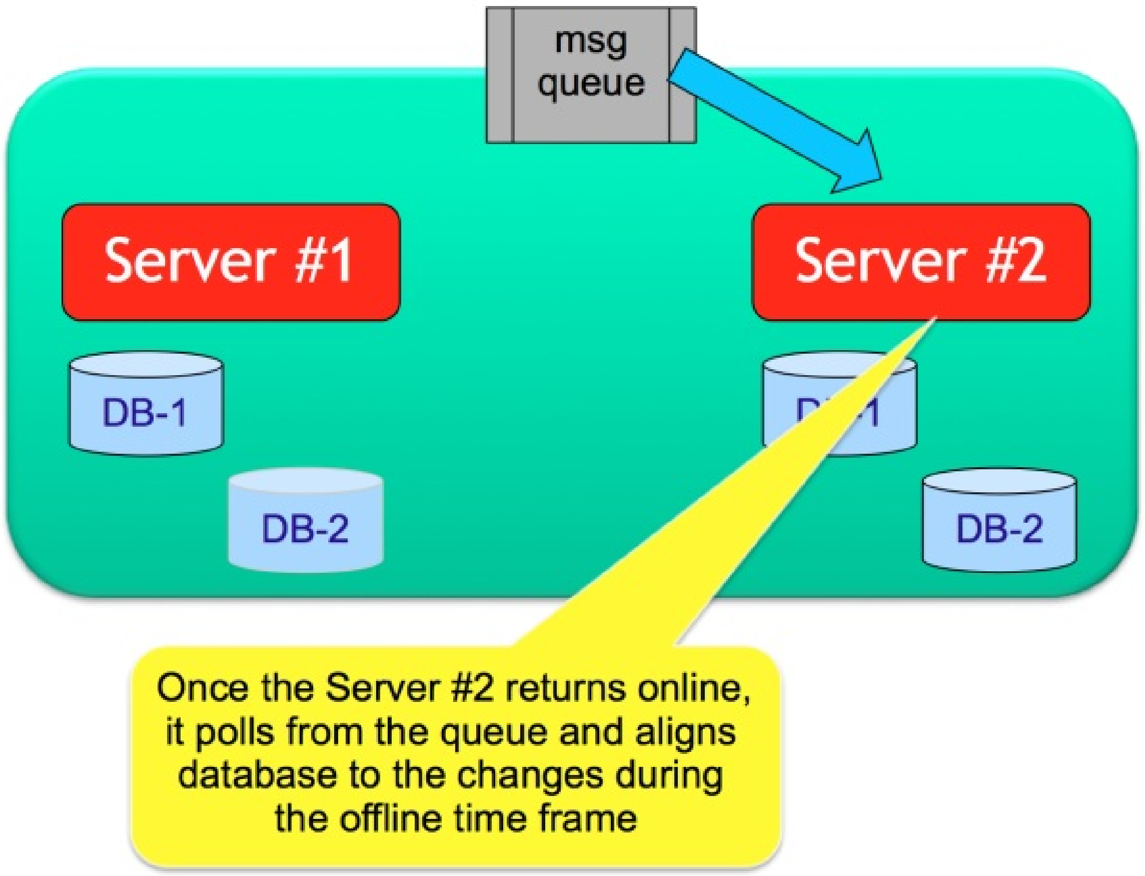

DB-1 is present on both nodes, start replication
Server #1
Server #2
DB-1
DB-1
DB-2
DB-2
If DB-2 has **autoDeploy:true**, then the database is deployed on Server #1 and replication is started

Server unreachable

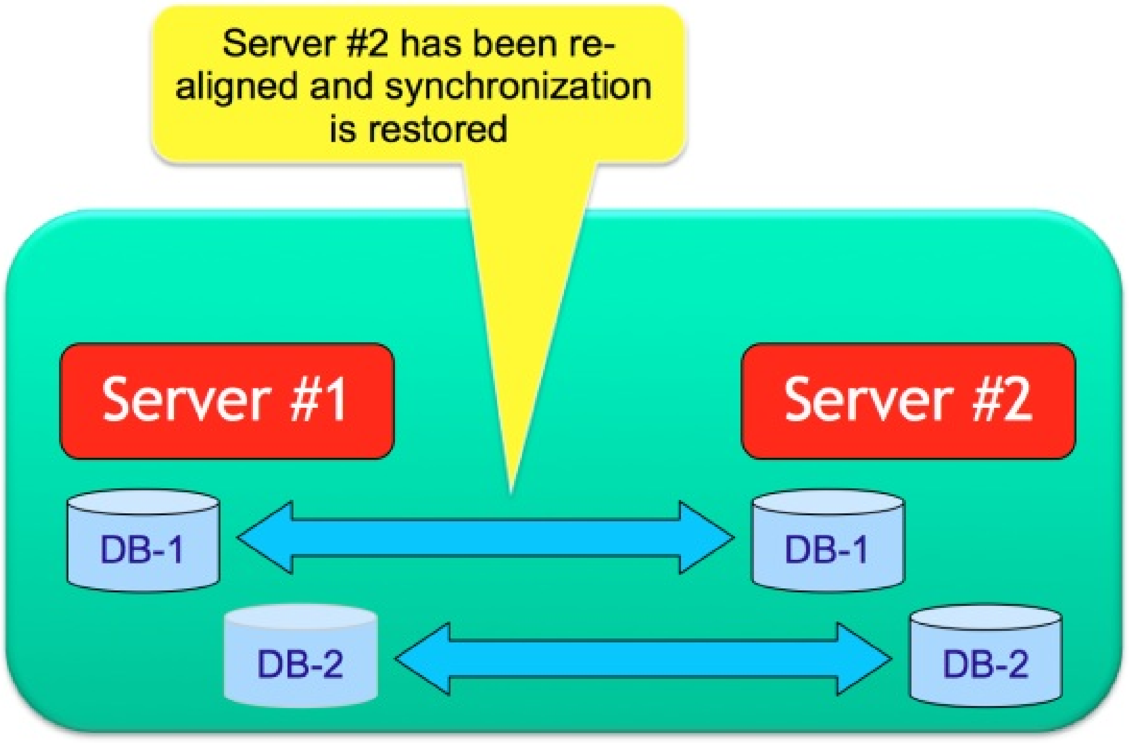
In case a server becomes unreachable, the node is removed by database configuration unless the setting `"hotAlignment" : true`. In this case all the new synchronization messages are kept in a distributed queue.


If a Server #2 becomes unreachable and **hotAlignment:true**, then all the messages are kept in queue waiting the node come online back again
msg queue
Server #1
Server #2
DB-1
DB-1
DB-2
DB-2

As soon as the Server becomes online again, it starts the synchronization phase (status=SYNCHRONIZING) by polling all the synchronization messages in the queue.


msg
queue
Server #1
Server #2
DB-1
DB-2
DB-1
DB-2
Once the Server #2 returns online, it polls from the queue and aligns database to the changes during the offline time frame

Once the alignment is finished, the node becomes online (status=ONLINE) and the replication continues like at the beginning.


Server #2 has been re-aligned and synchronization is restored
Server #1
Server #2
DB-1
DB-1
DB-2
DB-2

Further readings

Continue with:

- Distributed Architecture
- Distributed Sharding
- Distributed database configuration

Sharding

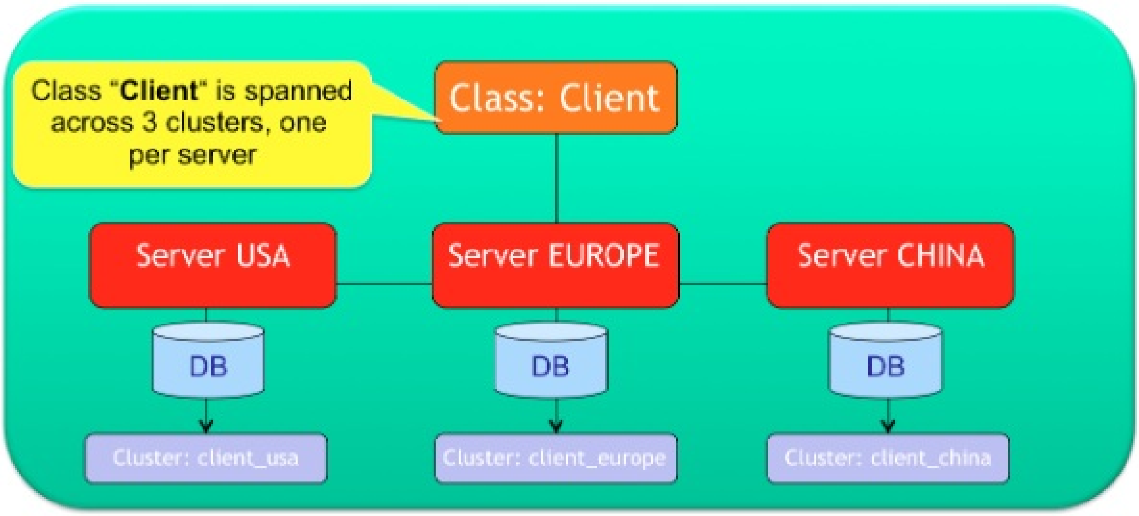
NOTE: Sharding is a new feature with some limitations. Please read them before using it.

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.



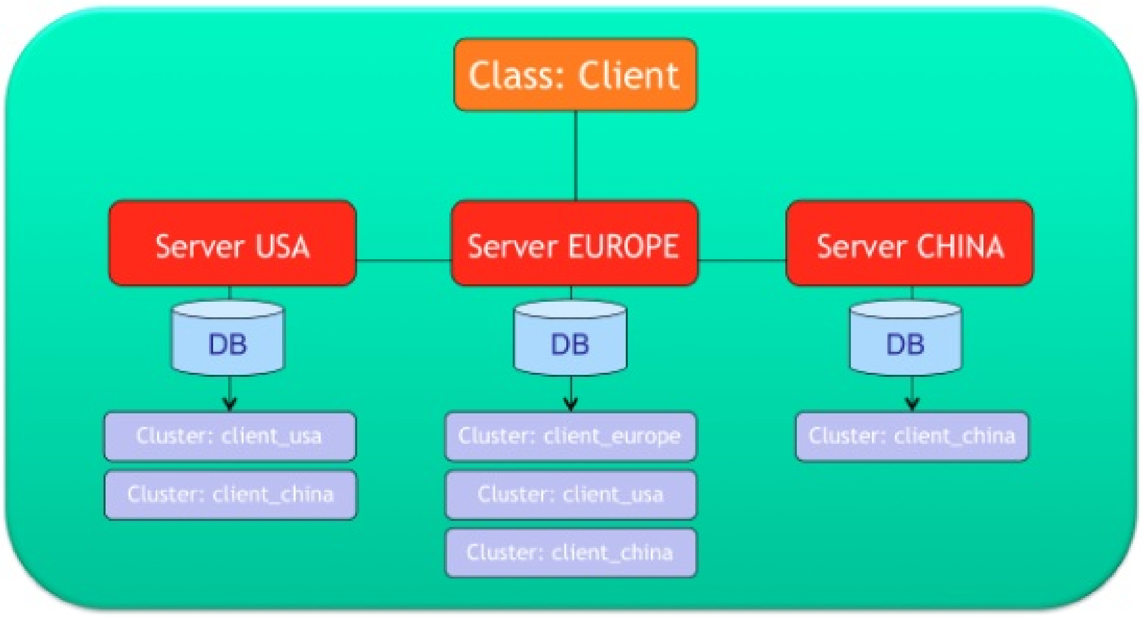
Shards, based on clusters, work against indexed and non-indexed class/clusters.

Multiple servers per cluster

You can assign each cluster to one or more servers. If more servers are enlisted the records will be copied in all the servers. This is similar to what RAID stands for Disks. The first server in the list will be the **master server** for that cluster.

This is an example of configuration where the Client class has been split in the 3 clusters client_usa, client_europe and client_china, each one with different configuration:

- `client_usa` , will be managed by "usa" and "europe" nodes
- `client_europe` , will be managed only by "europe" node
- `client_china` , will be managed by all 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,
  "hotAlignment": false,
  "readQuorum": 1,
  "writeQuorum": 2,
  "failureAvailableNodesLessQuorum": false,
  "readYourWrites": true,
  "clusters": {
    "internal": {
    },
    "index": {
    },
    "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 as node joins the distributed cluster. These cluster names have the node name as suffix: `<class>_<node>` . Example: `client_usa` . 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 previous clusters where it was master to the same node again following the convention `<class>_<node>` .

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 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".

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:

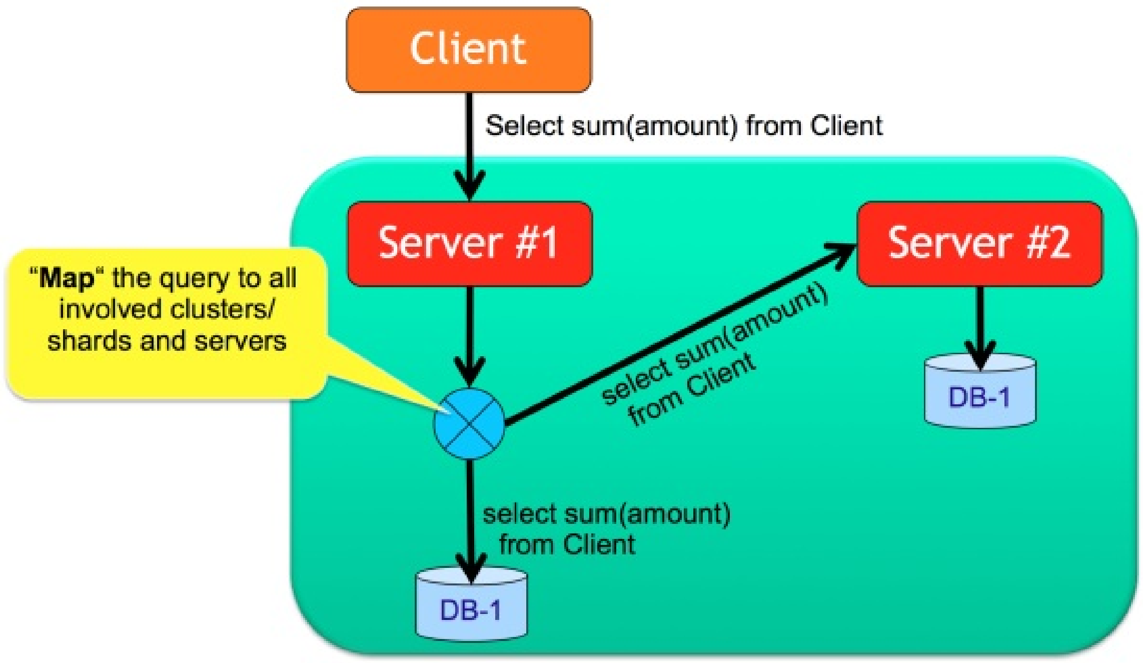
```
SELECT FROM CLUSTER:client_china
```

In this case the local node (USA) is used, because `client_china` is hosted on local node.

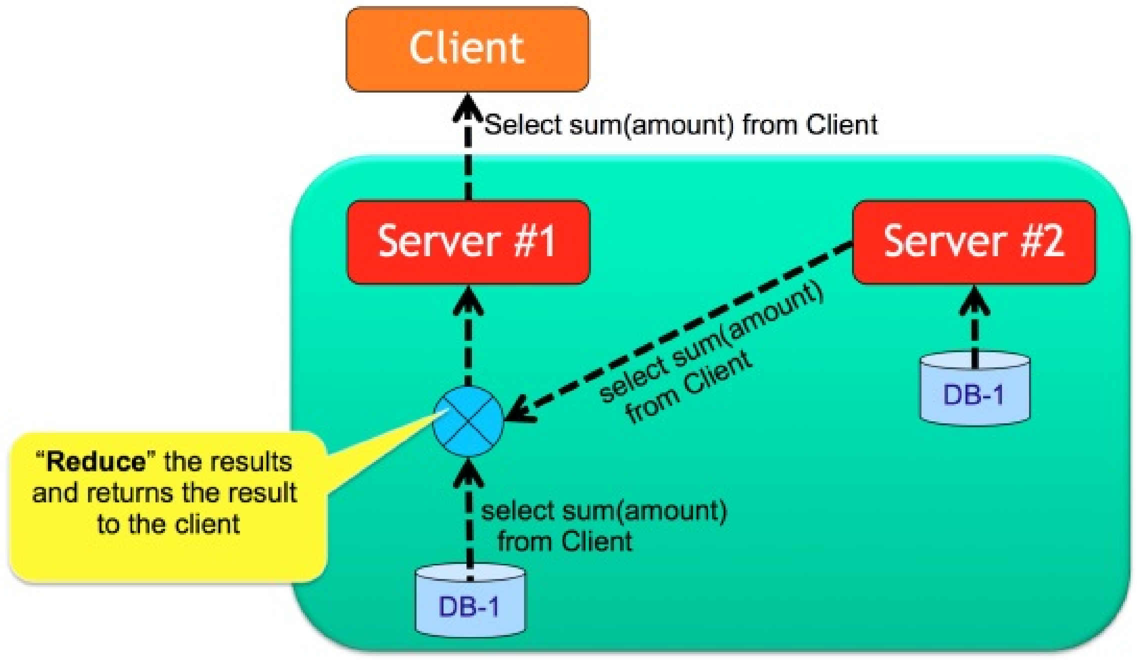
Map-Reduce

OrientDB supports Map/Reduce by using the OrientDB SQL. The Map/Reduce 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
```


Client
Select sum(amount) from Client
Server #1
Server #2
"Map" the query to all involved clusters/ shards and servers
select sum(amount) from Client
DB-1
select sum(amount) from Client
DB-1

In this case the query is executed across all the 3 nodes and then filtered again on starting node.


Client
Select sum(amount) from Client
Server #1
Server #2
select sum(amount) from Client
DB-1
"Reduce" the results and returns the result to the client
select sum(amount) from Client
DB-1

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

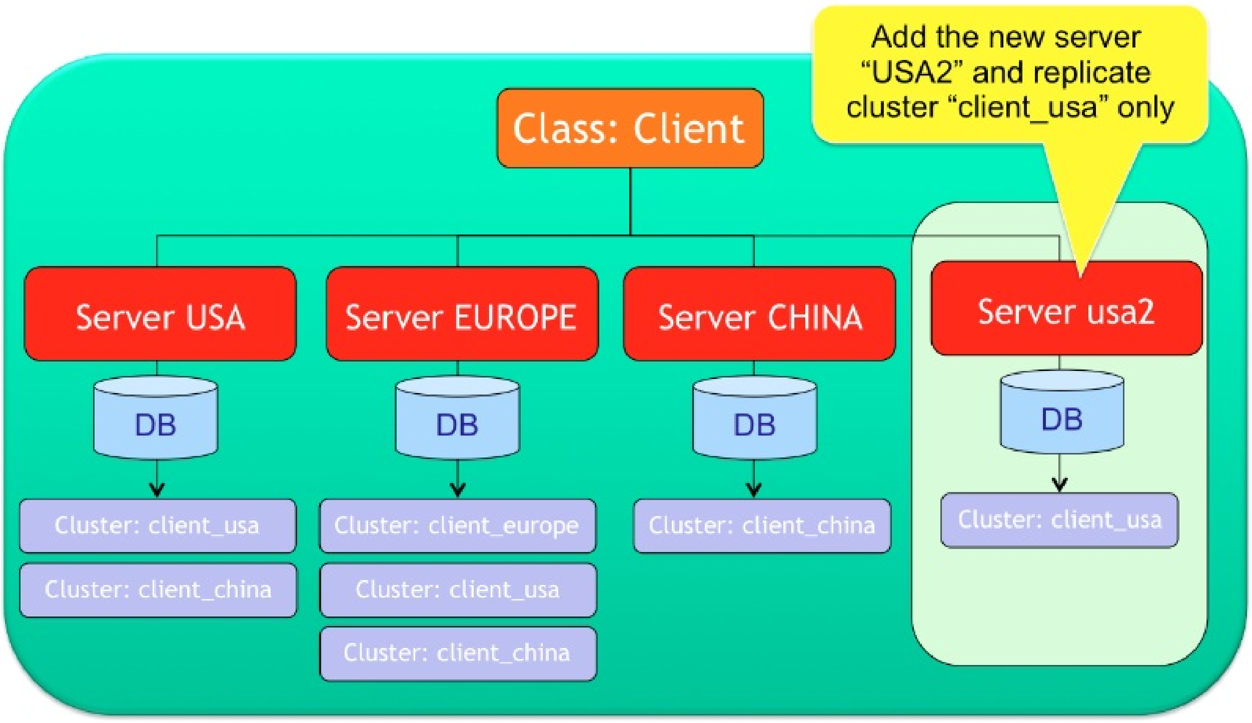
Indexes

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".


Add the new server "USA2" and replicate cluster "client_usa" only
Class: Client
Server USA
DB
Cluster: client_usa
Cluster: client_china
Server EUROPE
DB
Cluster: client_europe
Cluster: client_usa
Cluster: client_china
Server CHINA
DB
Cluster: client_china
Server usa2
DB
Cluster: client_usa

Distributed Cache

OrientDB has own more Cache levels. When OrientDB runs in Distributed-Architecture, each server has own cache. All the caches in each server are independent.

Distributed 2nd Level cache

You can also have a shared cache among servers, by enabling the Hazelcast's 2nd level cache. To enable it set the **cache.level2.impl** property in orientdb-dserver-config.xml file with value **com.orientechnologies.orient.server.hazelcast.OHazelcastCache**:

Note that this will slow down massive insertion but will improve query and lookup operations.

Example in **orientdb-dserver-config.xml** file:

```
...
<properties>
  <!-- Uses the Hazelcast distributed cache as 2nd level cache -->
  <entry name="cache.level2.impl" value="com.orientechnologies.orient.server.hazelcast.OHazelcastCache" />
</properties>
```

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 Architecutre. 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.

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]
INFO updated distributed configuration for database: GratefulDeadConcerts:
----------
{
   "replication": true,
   "autoDeploy": true,
   "hotAlignment": true,
   "resyncEvery": 15,
   "clusters": {
      "internal": {
         "replication": false
      },
      "index": {
         "replication": false
      },
      "*": {
         "replication": true,
         "readQuorum": 1,
         "writeQuorum": 2,
         "failureAvailableNodesLessQuorum": false,
         "readYourWrites": true,
         "partitioning":{
            "strategy": "round-robin",
            "default":0,
            "partitions": ["","node1383734730415","node1384015873680"]("","node1383734730415","node1384015873680".md)
         }
      }
   },
   "version": 1
}
---------- [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 v1.6.1-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`.

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.

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.

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.

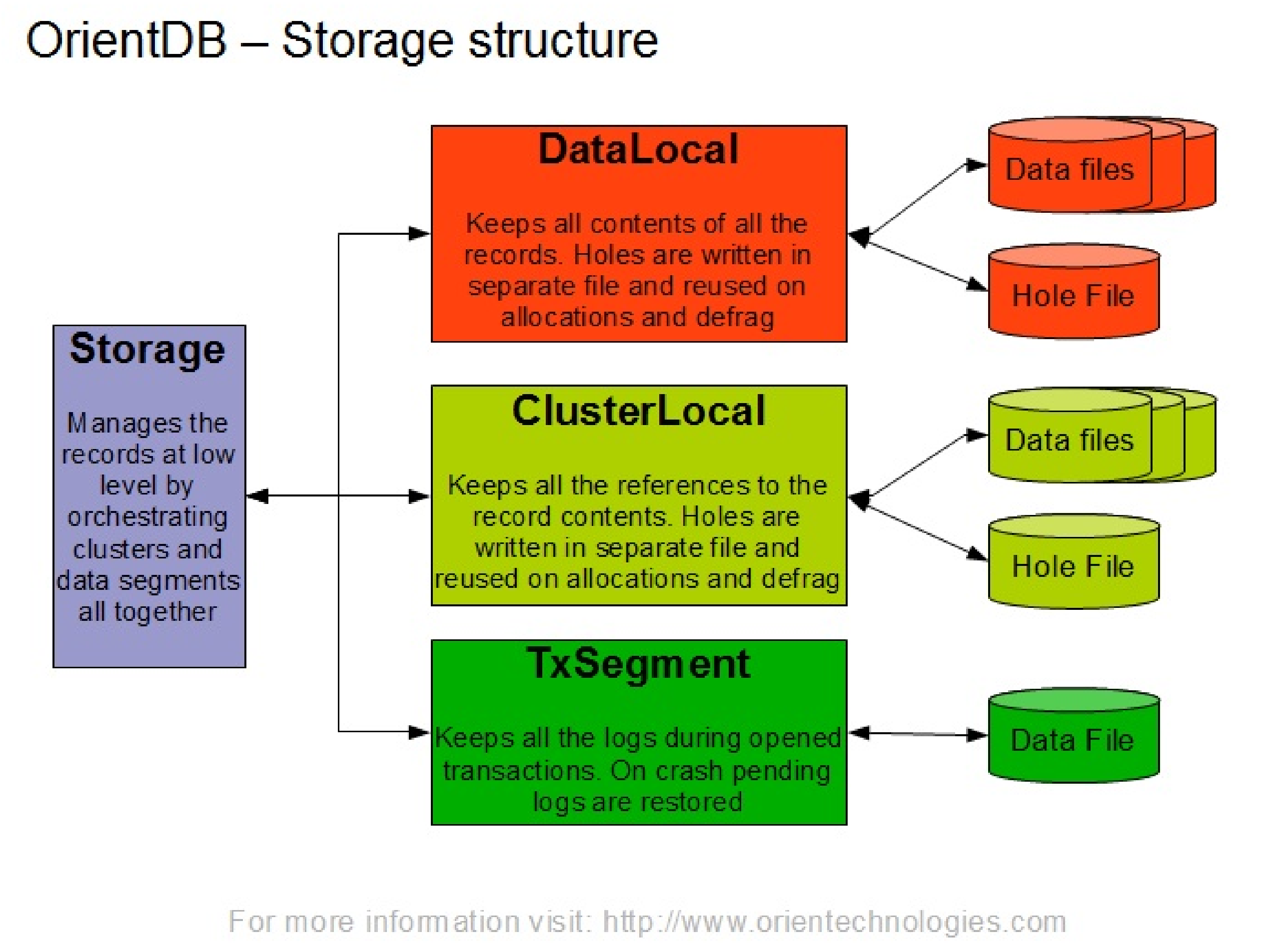
Second case a bit more tricky. Because WAL is append only log, there is two possible sub-cases, lets suppose we have 3 pages after 2-nd (broken) flush. First and first half of second page were flushed during first flush and second half of second page and third page were flushed during second flush. Because 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 (Not more available since 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 Cluster and Data Segments.


OrientDB – Storage structure
Storage
Manages the records at low level by orchestrating clusters and data segments all together
DataLocal
Keeps all contents of all the records. Holes are written in separate file and reused on allocations and defrag
ClusterLocal
Keeps all the references to the record contents. Holes are written in separate file and reused on allocations and defrag
TxSegment
Keeps all the logs during opened transactions. On crash pending logs are restored
Data files
Hole File
Data files
Hole File
Data File
For more information visit: http://www.orientechnologies.com

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).

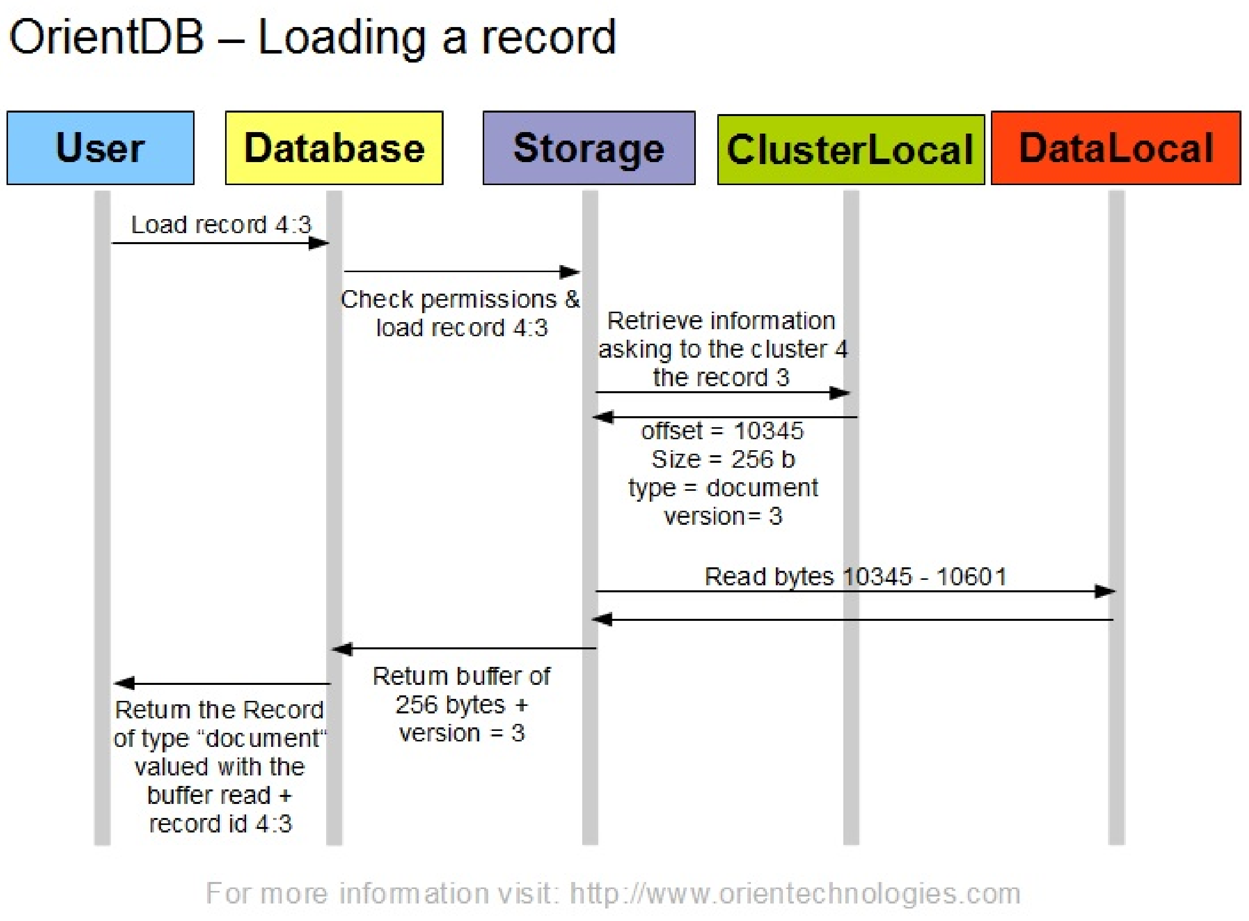
By default OrientDB creates the first data segment named "default". In the folder that contains your database you will find the following files:

- 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:


OrientDB – Loading a record
User
Database
Storage
ClusterLocal
DataLocal
Load record 4:3
Check permissions & load record 4:3
Retrieve information asking to the cluster 4 the record 3
offset = 10345
Size = 256 b
type = document
version= 3
Read bytes 10345 - 10601
Return buffer of 256 bytes + version = 3
Return the Record of type "document" valued with the buffer read + record id 4:3
For more information visit: http://www.orientechnologies.com

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-Member 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.*

Limitations running distributed

OrientDB has some limitations you should notice when you work in Distributed Mode:

- `hotAlignment:true` could bring the database status as inconsistent. Please set it always to 'false`, the default
- 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
- split network case: this is not well managed and in case you setup 4 nodes and the network is split between 2 nodes on the left, and 2 nodes on the right, each partition will think to be the only survived and on rejoin database could be inconsistent. Please always setup an odd number of nodes, so there will always be a majority in quorum
- 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.

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")>
| <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

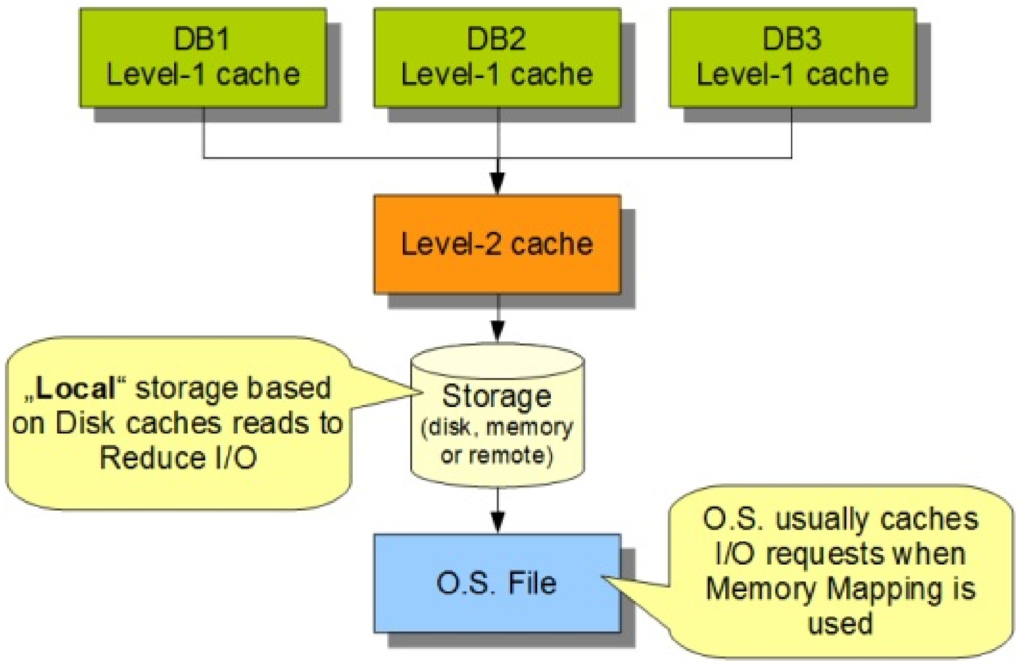
You can create an index with your Index Engine with sql with this syntax

```
CREATE INDEX Foo.bar ON Foo (bar) NOTUNIQUE ENGINE CUSTOM
```

where CUSTOM is the name of your index engine

Caching

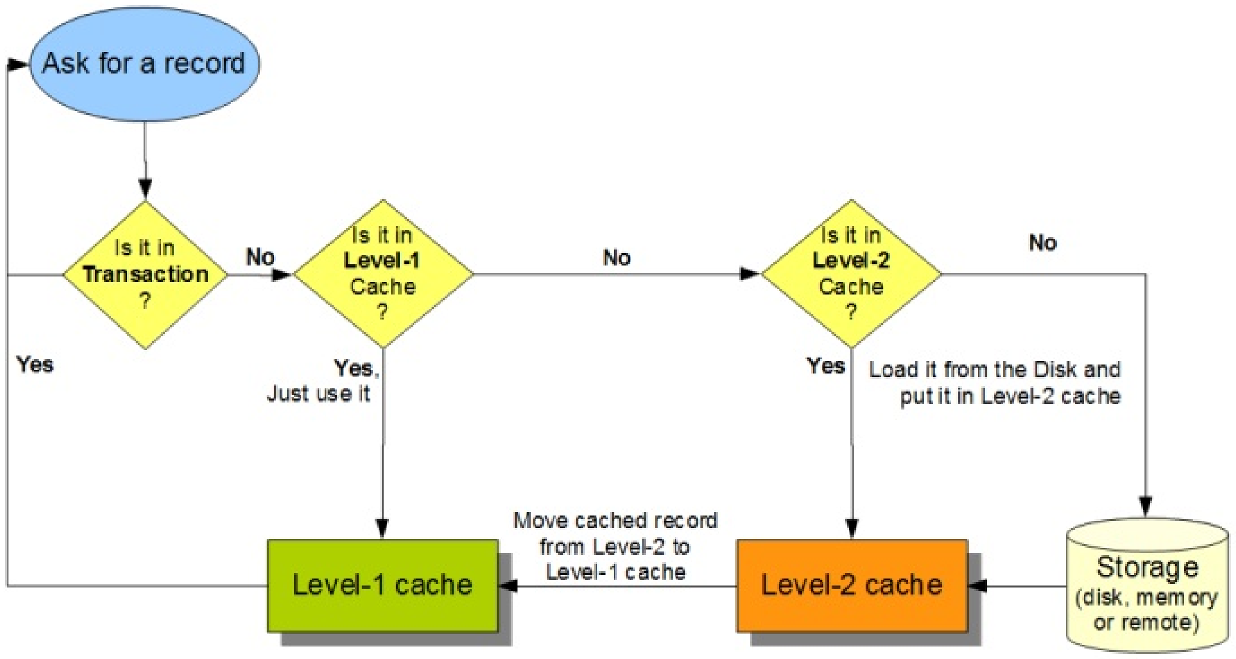
OrientDB has several caching mechanisms that act at different levels. Look at this picture:


DB1 Level-1 cache
DB2 Level-1 cache
DB3 Level-1 cache
Level-2 cache
„Local" storage based on Disk caches reads to Reduce I/O
Storage (disk, memory or remote)
O.S. File
O.S. usually caches I/O requests when Memory Mapping is used

- **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?

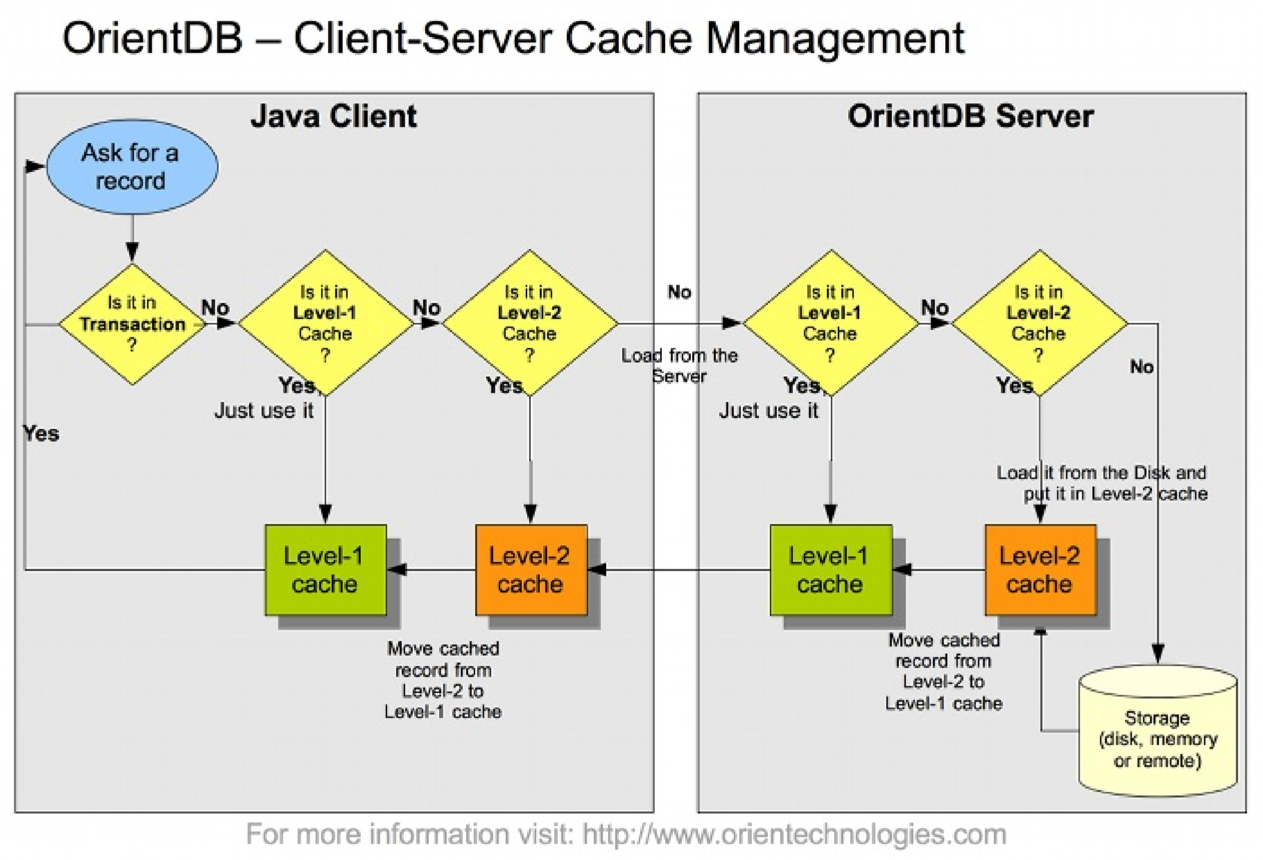
Local Mode (embedded database)



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();
```

Disable Local cache

Disabling of local cache may lead to situation when 2 different instances of the same record will be loaded and `OConcurrentModificationException` may be thrown during record update even in single-thread mode.

To disable it use the system property `cache.local.enabled` by setting it at startup:

```
java ... -Dcache.local.enabled=false ...
```

or via code before to open the database:

```
OGlobalConfiguration.CACHE_LOCAL_ENABLED.setValue(false);
```

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	

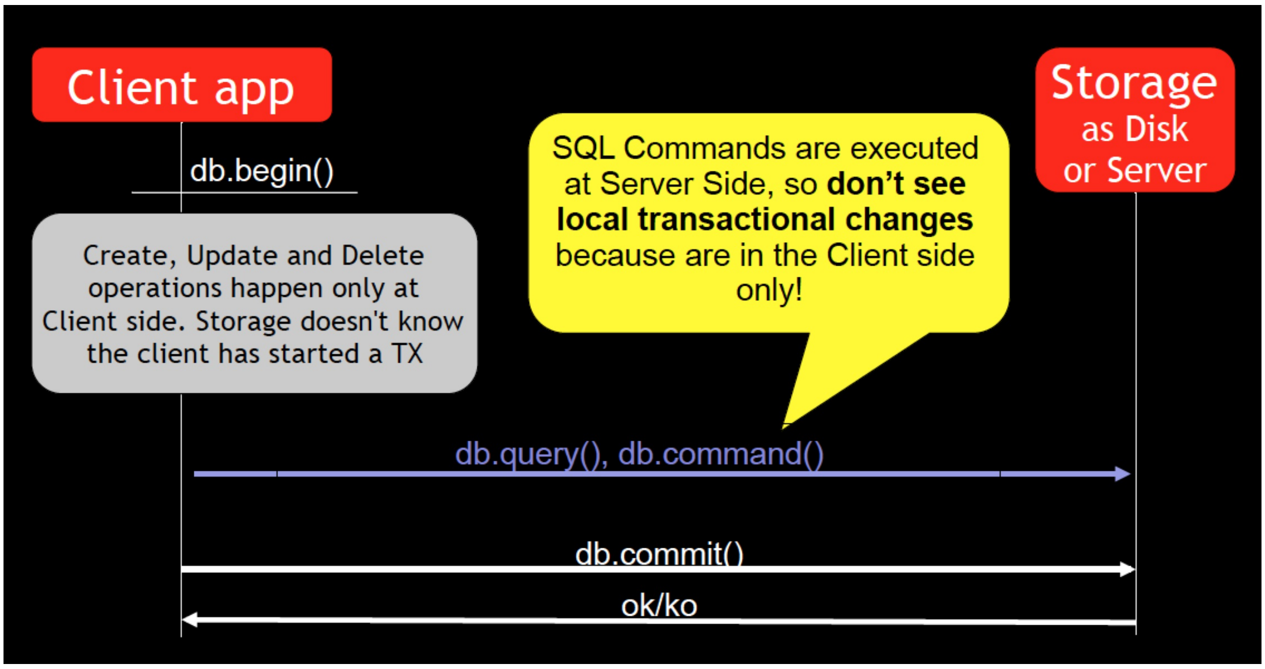
At operation 7 the client 1 continues to read the same version of x read in operation 2.

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.


Client app
db.begin()
Create, Update and Delete operations happen only at Client side. Storage doesn't know the client has started a TX
SQL Commands are executed at Server Side, so **don't see local transactional changes** because are in the Client side only!
Storage as Disk or Server
db.query(), db.command()
db.commit()
ok/ko

In this scenario you can have different isolation levels with commands.

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 RecordIDs to avoid to ask to the server a new RecordID every time. Temporary RecordIDs 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 RecordIDs 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 RecordIDs 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 to extend `OTriggered` base class. Then define a custom property for the event you're interested on. 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 language supported by OrientDB and 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 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 in 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

You could need to define a special action only against one or more documents. To do this, let your class to extend `OTriggered` class.

Example to execute a trigger, as Javascript function, against an existent 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 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.

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).

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.

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-enterprise-**.jar`
- `orientdb-server-**.jar`

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.

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 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

```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. 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.

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

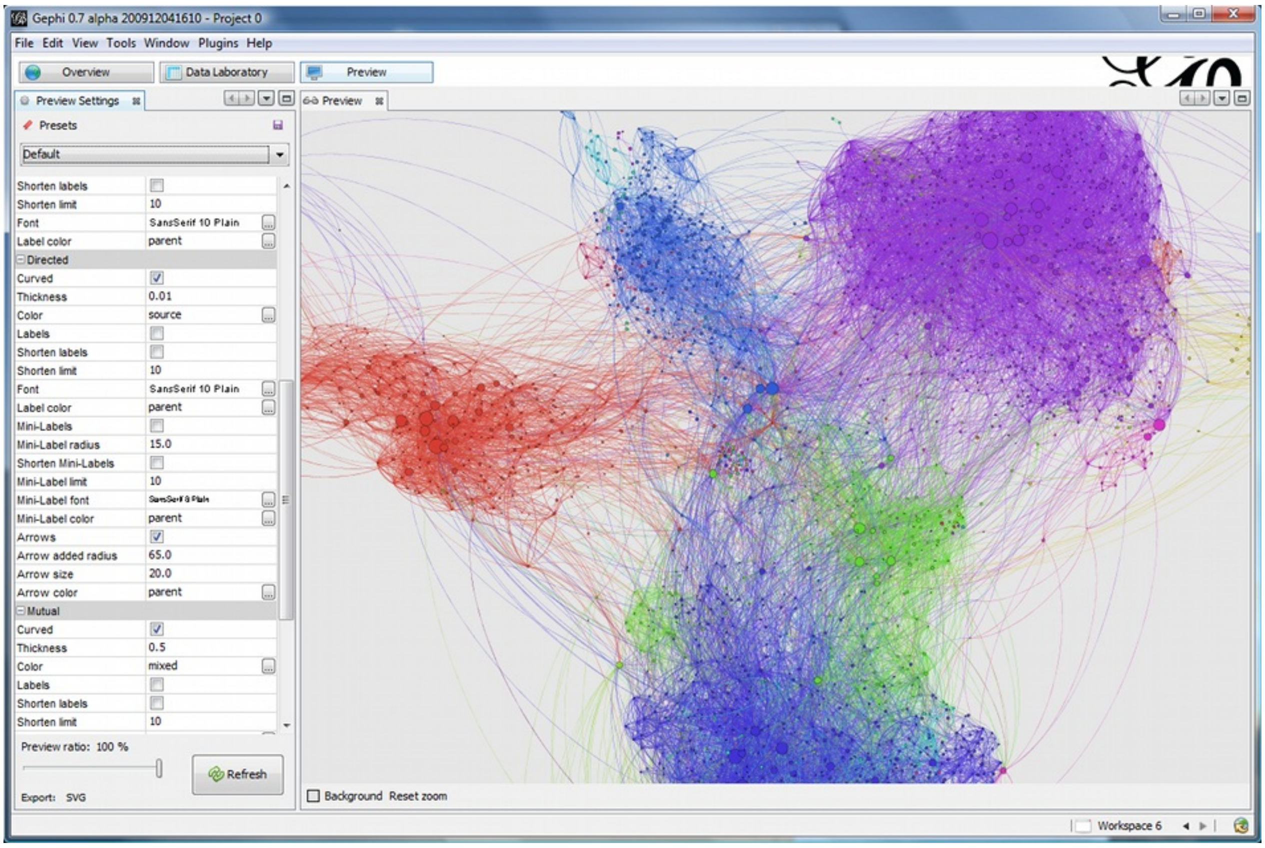
Note: only Rexster 0.5-SNAPSHOT and further releases work with OrientDB GraphEd
In this section we present a step-by-step guide to Rexster-ify an OrientDB graph.
We assume that:

- 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



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.

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. As the time of writing this notes, the state of branches is:

- develop: work in progress for next 2.2.x release (2.2.0-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.

The Team

If you want to contribute to the project, follow the Contributor rules.

Committers

Committers have reached the **Joda Level** OrientDB certification. They coordinates updates, patches, new tasks and answer actively to the Google Group. They talk in a private Mailing List to take decision all together. All the committers refer to the Committer's guide.

Luca Garulli

Description Luca is the original author of OrientDB product and the main committer. In order to handle indexes in efficient way Luca has created the new MVRB-Tree algorithm (it was called RB+Tree but another different algorithm already exists with this name) as mix of Red-Black Tree and B+Tree. MVRB stands for Multi Value Red Black because stores multiple values in each tree node instead of just one as RB-Tree does. MVRB-Tree consumes less than half memory of the RB-Tree implementation mantaining the original speed while it balances the tree on insertion/update. Furthermore the MVRB-Tree allows fast retrieving and storing of nodes in persistent way. He is member of the Sun Microsystems JDO 1.0 Expert Group (JSR#12) and JDO 2.0 Expert Group (JSR#243) for the writing of JDO standard.
Company OrientDB Ltd
Links Twitter - Google+ - VisualizeMe - LinkedIn - Blog - Ohloh
Since 2009

Artem Orobets

Description Committer since 2012 and contributor since 2011. He started diving into indexes, composite indexes and many other was introduced.
He have deep knowledge about the MVRB-Tree algorithm, the optimization of the indexes on queries, Transactions and Binary storage.
Links Twitter LinkedIn
Since 2012

Andrey Lomakin

Description Committer since 2012 and contributor since 2011. He started diving into indexes, composite indexes and many other was introduced.
He is:

1. Author of disk based storage system in OrientDB (plocal) which provides such features as durability and thread safety. Durability is achieved using write-ahead logging approach.
2. Author of "direct memory" disk cache (it is replacement of MMAP which is used underneath of all plocal components) which is based on 2Q and WOW cache algorithms.
3. Author of index system. Both hash and sbtree indexes.
4. Co-author (together with Artem Orobets) modern implementation of graph relationships.

Company OrientDB Ltd
Links Twitter LinkedIn
Since 2012

Luigi Dell'Aquila

Description 10 years experience as an ICT consultant, passionate software developer and Open Source enthusiast. Luigi managed OrientDB Academy since 2013, and since 2014 he manages Orient Technologies consulting services. He is also one of the main OrientDB core committers, mainly focused on OrientSQL, query execution and optimization.
Company OrientDB Ltd
Links Twitter - LinkedIn
Since 2013

Luca Molino</h1>

Description Contributor since 2010 and committer since 2012 Luca is author of various Http commands and the network protocol multipart management; author of the v1.0 ObjectDatabase implementation; author of the centralized Fetching strategy; author of the FIND REFERENCES SQL command; author of the ttl bash orient console; worked on SQL commands, Storage creation\deleting and more.
Company Asset Data
Links Twitter GitHub
Since 2012

Contributors

Contributors are all the people that contribute in any way to the OrientDB development as code, tests, documentation or other. They talk in a private Mailing List with all the other committers and contributors and are updated on changes in internal stuff like binary protocol. One time patch doesn't make you a contributor, but if you're developing a driver or you sent more patches then you are a candidate to figure in this list.

Contributors (in alphabetic order):

Anton Terekhov

Description Web developer since 2001, PHP developer since 2002. Developer and maintainer of OrientDB-PHP driver for binary protocol (https://github.com/AntonTerekhov/OrientDB-PHP), bug hunter, binary protocol tester :-) . Speaker on two Russian IT-conferences. Founder, CEO and Lead Developer of own company. Now specialized at high load, distributed web systems.
Company NetMonsters
Links Facebook
Since 2011

Artyom Loginov

Description Artem took part in MMAP file improvement
Since 2011

Dino Ciuffetti

Description Dino is responsable of the Cloud infrastructure of NuvolaBase, providing OrientDB databases as service. He develop in PHP but his main skill is System Administrator and hacking on Linux platforms.
Since 2012

Federico Fissore

Description Federico references to himself in the third person and develops the node.js driver, which powers its baby creature, http://presentz.org
Links GitHub Twitter LinkedIn Google+
Since

Gabriel Petrovay

Description Gabriel has been started the node.js OrientDB driver that implements the binary protocol. This helped discovering some problems in the binary protocol. Also helped a little with the corresponding documentation pages.
Links Twitter LinkedIn
Since 2012

Johann Sorel

Description Java Developer specialized in Geographic Information Systems (GIS). Core developer on Geotoolkit project (http://www.geotoolkit.org) and related projects : GeoAPI, Mapfaces, Constellation, MDWeb, Puzzle-GIS. Member at the Open Geospatial Consortium (OGC) for elaboration of geographic specifications. Contributions on OrientDB are related to modularisation and performances.
Company Geomatys
Links Web Site Ohloh
Since 2013, contributors since 2012

Rus V. Brushkoff

Description Contributed C++ binding (https://github.com/Sfinx/orientpp)
Company SfinxSoft
Links LinkedIn
Since 2012

Tomas Bosak

Description Tomas is developing and maintaining C# binary protocol driver for OrientDB.

Company ONXPO

Links Home Page GitHub Twitter LinkedIn

Since 2012

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 is 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!

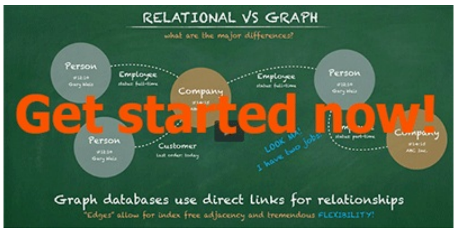
If you want to brainstorm a potential new feature, then the OrientDB Google Group (see above) is probably a better place to start.

Email

info@orientdb.com

If you want more information about Commercial Support, Consultancy or Training, email us.

More on Tutorials

	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.

RELATIONAL VS GRAPH
what are the major differences?
Get started now!
Person
Employee
Company
Customer
Graph databases use direct links for relationships
"Edges" allow for index free adjacency and tremendous FLEXIBILITY!

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 : **NoSQLadoption: 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 NOSQLanche 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 enhancement for 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 2.2

```
- Development started on.: June 25th 2015
- Expected BETA..........: February 2016
- Expected first RC......: March 2016
- Planned final GA......: April 2016
```

Status

Last update: January 12, 2016

Module	Feature	Status
OrientJS	Native unmarshaling of requests by using C++ code	100%
Core	Dirty Manager	100%
Core	Incremental Backup	100%
Core	Automatic minimum clusters	100%
Core	AES and DES enchryption	100%
Core	Support SALT in passwords	100%
Distributed	Fast synchronization by using Incremental Backup	100%
Distributed	Faster replication by using remote binary protocol instead of hazelcast queues	100%
Distributed	Support for `majority` (default) and `all` in quorum	100%
Distributed / Remote protocol	Load balancing on client	100%
Remote	Support for IPV6	100%
SQL	Pattern matching	100%
SQL	Command Cache	100%
SQL	Automatic parallel queries	100%
SQL	Prefetching of disk pages	100%
SQL	Live Query -> Stable	100%
SQL	Update Edge	100%
SQL	Sequences, PR	100%
SQL	'Move cluster' command	100%
SQL	Command to manage users	100%
Java API	ODocument.eval()	100%
Studio	New P2P architecture, new Enterprise modules (it replaces the Enterprise Workbench)	100%
Lucene	Spatial Module New module for indexing of shapes, not only points	100%

Release 3.0

```
- Development started on.: September 2015
- Expected first RC......: June 2016
- Expected final GA......: August 2016
```

Status

Last update: December 8, 2015

Module	Feature	Status
Core	Multi-Threads WAL	0%
Core	WAL Compaction	0%
Core	Index rebuild avoid using WAL	0%
Core	Compression of used space on serialization	0%
Core	Increase cluster-id from short to int	15%
Core	Indexing of embedded properties	0%
Core	Index per cluster	0%
SQL	Distributed SQL Executor	0%
SQL	Multi-line queries in batch scripts	0%
Java API	Support fot TinkerPop 3	30%
Distributed	Replication of in-memory databases	0%
Distributed	Auto-Sharding	0%
Scheduler	Improve scheduler	0%
Console	Display distributed information about sharding and nodes	50%

Release 3.1

```
- Development started on.: -
- Expected first RC......: June 2016
- Expected final GA......: August 2016
```

Status

Last update: December 8, 2015

Module	Feature	Status
Core	Parallel Transactions	0%
Core	Override of properties	0%
Core	Auto close storages	0%
Core	Enhance isolation level also for remote commands	0%
Distributed	Optimized replication for cross Data Center	0%
Lucene	Faceted search	20%
Java API	ODocument.update()	0%
Java API	Improve SQL UPDATE syntax	100%
Remote protocol	Push messages on schema change	0%
Remote protocol	Push messages on record change	0%
SQL	shortestPaths() function	0%
SQL	New functions (strings, maths)	40%

Enterprise Edition

This documentation is referring to v2.1. In v2.2 the Enterprise Edition has been completely revisited.

This is the main guide on using **OrientDB Enterprise Edition**. For more information look at OrientDB Enterprise Edition.

To try Enterprise Edition ask for a Trial by writing to: info@orientdb.com.

OrientDB Enterprise Edition is composed by 2 modules:

- Enterprise **Agent**
- Enterprise **Workbench**

OrientDB Enterprise Agent

The Agent contains the license generated by Orient Technologies. If you're a client you already own Agent jar files to install. If you don't have them or you want to try Enterprise Edition write to: info@orientdb.com.

The Agent contains the Profiler component to get monitored by Workbench.

Installation

In order to enable Enterprise feature, copy the provided **agent-*.jar** file under the OrientDB Server "plugins" directory of each server. The plugin will be hot loaded by the server after few seconds (look at the server's output). In case the plugin is not loaded restart the OrientDB Server.

Once installed, the Agent Plugin displays the license information. Example:

```
2013-12-18 16:52:43:206 INFO Installing dynamic plugin 'agent-1.6.2.jar'...
*************************************************
*       ORIENTDB  -  ENTERPRISE EDITION         *
*                                               *
* Copyrights (c) 2015 Orient Technologies LTD  *
*************************************************
```

NOTE: OrientDB Enterprise Plugin and OrientDB Server must be of the same main version. Workbench 1.7.x works against all Agents 2.0.x. If you don't have the right version please write to the Orient Technologies: info@orientdb.com.

OrientDB Enterprise Workbench

NOTE: OrientDB Workbench runs as separate application. In order to avoid slow down of OrientDB Servers, it's a best practice to run Workbench on a separate server.

Download

Download the right OrientDB Workbench distribution, using the same Agent version:

- Workbench Web Application v. **2.1.0**:
  - Windows users: TAR.GZ
  - MacOSX: TAR.GZ
  - Linux, Any Unix like OSs: TAR.GZ
  - Help
- Workbench Web Application v. **1.7.4**:
  - Windows users: TAR.GZ
  - MacOSX: TAR.GZ
  - Linux, Any Unix like OSs: TAR.GZ

- Help

Install

Uncompress the Workbench distribution to a local directory. For Windows user it's a **ZIP** file, for all the others is a **TAR.GZ** archive.

Start and Use Workbench

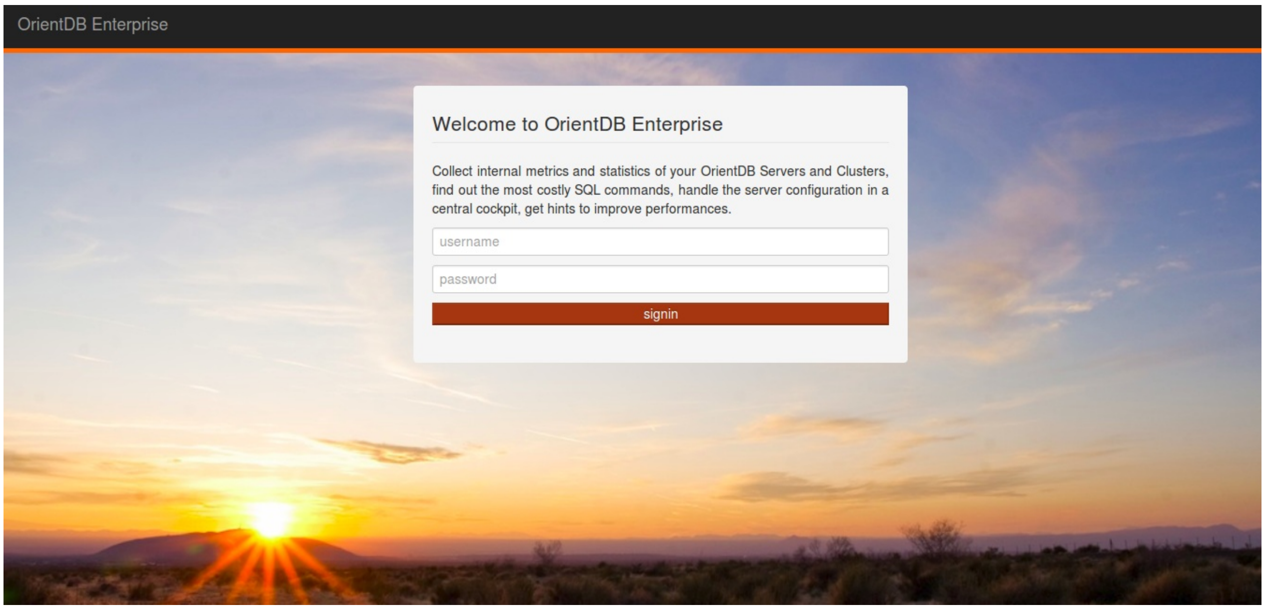
To start the **Workbench** go into the "bin" directory and double click on:

- **start-workbench.sh** for MacOSX, Linux and Unix users
- **start-workbench.bat**, for Windows users

Once started the Workbench ends with these messages:

```
************************************************
*   ORIENTDB WORKBENCH -  ENTERPRISE EDITION   *
*                                              *
* Copyrights (c) 2013 Orient Technologies LTD  *
************************************************
* Version...: 1.6.2                            *
************************************************

To open the Web Console open your browser to the URL: http://localhost:2491 and use 'admin' as user and password to log in, un
less you already changed it.
```

Now point your browser to the local server's IP address, port 2491, example: **http://localhost:2491**. This is the login page. Use the default credentials as user "admin" and password "admin" (you can change it once logged in).



Welcome to OrientDB Enterprise

Collect internal metrics and statistics of your OrientDB Servers and Clusters, find out the most costly SQL commands, handle the server configuration in a central cockpit, get hints to improve performances.

For the complete guide goto Workbench Guide.

Auditing

Starting from 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. Studio web tool provides a GUI on Auditing that makes configuration easier. Look at 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`.

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 TO writer
REVOKE ALL ON database.cluster.auditinglog TO reader
```

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.

Configuration

To turn on auditing, create the JSON configuration file with name `auditing-config.json` under the database folder. This is the syntax for configuration:

```
{
  "auditClassName": "<audit-class-name>",
  "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>"
    }
  ]
}
```

Where:

- `auditClassName` : document class used for auditing records. By default is "**AuditingLog**"
- `classes` : contains the mapping per class. Wildcard `*` represents any class
- `class-name` : class name to configure
- `polymorphic` : uses this class definition also for all sub classes. By default class definition is polymorphic
- `onCreateEnabled` : enable auditing for creation of records. Default is `false`

- `onCreateMessage` : custom message to write in the auditing record on create record. It supports dynamic binding of values, look at Customize the message
- `onReadEnabled` : enable auditing on reading of records. Default is `false`
- `onReadMessage` : custom message to write in the auditing record on read record. It supports dynamic binding of values, look at Customize the message
- `onUpdateEnabled` : enable auditing on updating of records. Default is `false`
- `onUpdateMessage` : custom message to write in the auditing record on update record. It supports dynamic binding of values, look at Customize the message
- `onUpdateChanges` : write all the previous values per field. Default is `true` if `onUpdateEnabled` is true
- `onDeleteEnabled` : enable auditing on deletion creation of records. Default is `false`
- `onDeleteMessage` : custom message to write in the auditing record on delete record. It supports dynamic binding of values, look at Customize the message
- `regexp` : is the regular expression to match in order to log the command execution
- `message` : is the optional message to log when the command is logged. It supports dynamic binding of values, look at Customize the message

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}"
    }
  }
}
```

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>}`	-

Customize the message

Messages can be customized, adding a placeholder for variables resolved at run-time. Below is a list of supported variables:

- `${command}` , is the executed command as text
- `${field.<field-name>}` , to use the field value. Example: `${field.surname}` to get the field "surname" from the current record