
OrientDB®

OrientDB Manual - version 2.0

This documentation is also available in PDF format.

Past releases:

- v1.7.8

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. OrientDB is sponsored by Orient Technologies, LTD.

OrientDB has features of both Document and Graph DBMSs. Written in Java and designed to be exceptionally fast: it can store up to 220,000 records per second on common hardware. Not only can it embed documents like any other Document database, but it manages relationships like Graph Databases with direct connections among records. You can traverse parts of or entire trees and graphs of records in a few milliseconds.

OrientDB supports schema-less, schema-full and schema-mixed modes and it has a strong security profiling system based on users and roles. Thanks to the SQL layer, OrientDB query language is straightforward and easy to use, especially for those skilled in the relational DBMS world.

To learn some key advantages of using OrientDB, take a look at Why OrientDB?

Is OrientDB really FREE?

OrientDB is free for any use including commercial with its Apache 2 Open Source License.

Orient Technologies, the company behind OrientDB, offers optional Professional Services such as Developer and Production Support, Training and Consultancy with transparent and competitive pricing. These options are available to ensure you're maximizing OrientDB's capabilities for your particular needs and use case.

OrientDB Community

Start learning about OrientDB with the OrientDB Manual. For any questions, visit the OrientDB Community Group. Need help? Go to Online Support. Do you want to hear about OrientDB at a conference or meetup? Take a look at Events.

Need Further Assistance?

If you have any questions or need assistance with OrientDB, please let us know. Check out our Get in Touch page for different ways of getting in touch with us.

Getting Started

In the last few years, there's been an explosion of many NoSQL 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 production environments.

What motivates companies to leave the comfort of a well established relational database world? It all surrounds the fact that you can better solve today's data problems with a modern database. Specifically, there are a few key areas:

- Performance
- Scalability (often extreme)
- Smaller footprint
- Developer productivity
- Schema flexibility

Most of these areas also happen to be 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 DBMS have remained unchanged for the most part: 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**: 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**: 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**: data model consists of document collections where individual documents can have multiple fields without necessarily defining a schema. The best known products of this group are MongoDB and CouchDB.
- **Graph databases**: domain model consists of vertices interconnected by edges creating a rich graph structure. The best known products of this group are OrientDB and Neo4j.

> 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 not a single category or model which is better than all 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 performant database that can perform optimally in the widest possible set of use cases. Our product is designed to be the best "go to" solution for data persistence. In the following parts of this tutorial, we will look closely at **OrientDB**, the original open-source, multi-model, next generation NoSQL product.

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 its ability to act as a true **Multi-Model** DBMS by combining all the features of the four models into one. These are not just interfaces to the database engine, but the engine, itself, was built to support all four models. This is also the main difference with other DBMSs that claim to be Multi-Model since they just implement an additional layer with an API that mimics additional models, but under the hood they're truly one model therefore limiting speed and scalability.

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 a 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 a benefit 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" instead of "collections" for grouping documents. This provides several benefits that we will discuss in further sections of the documentation. OrientDB's Document model 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 the document all the links are automatically resolved by OrientDB. This is the most important difference with any other Document Database like MongoDB.

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

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 incoming Vertex (also known as head)
  - link to 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

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 a richer model than the classic Key/Value one. The Key/Value model provides "buckets" to group key/value pairs in different containers. The most classic use cases with 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

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

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 by providing Enterprise features such as:
  - Query Profiler
  - Distributed Clustering configuration
  - Metrics Recording
  - Live Monitoring with configurable Alerts

An Enterprise Edition license is included without charge if you purchase Support.

Prerequisites

Both editions run on every operating system that has an implementation of the Java Virtual Machine (JVM), for example:

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

This means the only requirement for using OrientDB is to have Java version 1.6 or higher installed.

Download Binaries

The easiest and fastest way to start using OrientDB is to download binaries from the Official OrientDB Download Page.

Compile Your Own Community Edition

Alternatively, you can clone the Community Edition project from GitHub and compile it. This allows you access to the latest functionality without waiting for a distribution binary. To build the Community Edition, you must first install the Apache Ant tool and follow these steps:

```
> git clone git@github.com:orientechnologies/orientdb.git
> cd orientdb
> ant clean install
```

After the compilation, all the binaries are placed under the `../releases` directory.

Change Permissions

The Mac OS X, Linux, and UNIX based operating systems typically require you to change the permissions to execute scripts. The following command will apply the necessary permissions for these scripts in the `bin` directory of the OrientDB distribution:

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

Use inside of OSGi container

OrientDB uses a ConcurrentLinkedHashMap implementation provided by https://code.google.com/p/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
```

Other Resources

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

- Install as service on Unix, Linux and MacOSX
- Install as service on Windows

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

Run the server

After you have downloaded the binary distribution of OrientDB and unpacked it, you are now able to start the server. This is done by executing `server.sh` (or `server.bat` on Windows) located in `bin` directory:

```
> cd bin
> ./server.sh
```

You should now see the following output:

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

2012-12-28 01:25:46:319 INFO Loading configuration from: config/orientdb-server-config.xml..
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 [OServerNetworkLis
2012-12-28 01:25:47:290 INFO Listening http connections on 0.0.0.0:2480 [OServerNetworkListe
2012-12-28 01:25:47:317 INFO OrientDB Server v1.6 is active. [OServer]
```

The log messages explain what happens when the server starts:

1. The server configuration is loaded from `config/orientdb-server-config.xml` file. To

know more about this step, look into OrientDB Server
2. By default, a "temp" database is always loaded into memory. This is a volatile database that can be used to store temporary data
3. Binary connections are listening on port 2424 for all configured networks (0.0.0.0). If you want to change this configuration edit the `config/orientdb-server-config.xml` file and modify port/ip settings
4. HTTP connections are listening on port 2480 for all configured networks (0.0.0.0). If you want to change this configuration edit the `config/orientdb-server-config.xml` file and modify port/ip settings

OrientDB server listens on 2 different ports by default. Each port is dedicated to binary and HTTP connections respectively:

- **binary** port is used by the console and clients/drivers that support the network binary protocol
- **HTTP** port is used by OrientDB Studio web tool and clients/drivers that support the HTTP/REST protocol or tools like CURL

Run the console

OrientDB provides a command line interface. It can be used to connect to and work with remote or local OrientDB servers.

You can start the command line interface by executing `console.sh` (or `console.bat` on Windows) located in the `bin` directory:

```
> cd bin
> ./console.sh
```

You should now see a welcome message:

```
OrientDB console v.1.6 www.orientechnologies.com
Type 'help' to display all the commands supported.

orientdb>
```

Type the "help" or "?" command to see all available console commands:

```
orientdb> help

AVAILABLE COMMANDS:

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

Connecting to server instance

Some console commands such as `list databases` or `create database` can be run while only connected to a server instance (you do not have to be connected to a database). Other commands require you to be connected to a database. Before you can connect to a fresh server instance and fully control it, you need to know the root password. The root password is located in `config/orientdb-server-config.xml` (just search for the **users** element). If you want to change it, modify the XML file and then restart the server.

If you have the required credentials, you should now be able to connect using the following command:

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

Next, you can (for example) list databases using the command:

```
orientdb> list databases

Found 1 databases:

* GratefulDeadConcerts (plocal)
```

To connect to another database we can again use the `connect` command from the console and specify the server URL, username, and password. By default each database has an "admin" user with password "admin" (change the default password on your real database). To connect to the *GratefulDeadConcerts* database on the local server execute the following:

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

Let's analyze the URL we have used: `remote:localhost/GratefulDeadConcerts`. The first part is the protocol, "remote" in this case, which contacts the server using the TCP/IP protocol. "localhost" is the host name or IP address where the server resides; in this case it is on the same machine. "GratefulDeadConcerts" is the name of the database to which we want to connect.

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.

For more detailed information about the commands see the console page.

Classes

It's easier to introduce OrientDB's basic concepts by outlining the Document Database API. It is more similar to Relational DBMS concepts and therefore a little more natural to follow for many developers. These basic concepts are shared between all of OrientDB's APIs: Document, Object, and Graph.

As with the relational DBMS, OrientDB has the concept of records as an element of storage. There are different types of records, but for the next examples we will always use the document type. A document is composed of attributes and can belong to one class. Going forward we will also refer to the attributes with the terms "fields" and "properties".

The concept of class is well known to those who program using object-oriented languages. Classes are also used in OrientDB as a type of data model according to certain rules. To learn more about Classes in general take a look at Wikipedia.

To list all the configured classes, type the `classes` command in the console:

```
orientdb> classes

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

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

```
orientdb> create class Student

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

OrientDB allows you to work in a schema-less mode, without defining properties.

However, properties are mandatory if you define indexes or constraints. To create a new property use the `create property` command. Here is an example of creating three properties against the `Student` class:

```
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
```

To display 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/CLASS             | MANDATORY | R
------------------------------+-------------+-------------------------------+-----------+--
 birthDate                    | DATE        | null                          | false     | f
 name                         | STRING      | null                          | false     | f
 surname                      | STRING      | null                          | false     | f
------------------------------+-------------+-------------------------------+-----------+--
```

To add a constraint, use the `alter class` command. For example, let's specify that the `name` field should be at least 3 characters:

```
orientdb> alter property Student.name min 3

Property updated successfully
```

To see all the records in a class, use the `browse class` command:

```
> browse class OUser
```

In this case we are listing all of the users of the database. This is not particularly secure. You should further deepen the OrientDB security system, but for now `OUser` is a class like any other. For each query the console always shows us the number of the records in the result set and the record's ID.

```
---+---------+--------------------+--------------------+--------------------+--------------
  #| RID     |name                |password            |status              |roles
---+---------+--------------------+--------------------+--------------------+--------------
  0|     #5:0|admin               |{SHA-256}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8
  1|     #5:1|reader              |{SHA-256}3D0941964AA3EBDCB00CCEF58B1BB399F9F898465E9886D5
  2|     #5:2|writer              |{SHA-256}B93006774CBDD4B299389A03AC3D88C3A76B460D538795BC
---+---------+--------------------+--------------------+--------------------+--------------
```

The first column is a number used as an identifier to display the record's detail. To show the first record in detail, it is necessary to use the `display record` command with the number of the record, in this case 0:

```
orientdb> display record 0
--------------------------------------------------
ODocument - Class: OUser   id: #5:0   v.0
--------------------------------------------------
                name : admin
            password : {SHA-256}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8A81F6F2AB448
              status : ACTIVE
               roles : [#4:0=#4:0]
```

Clusters

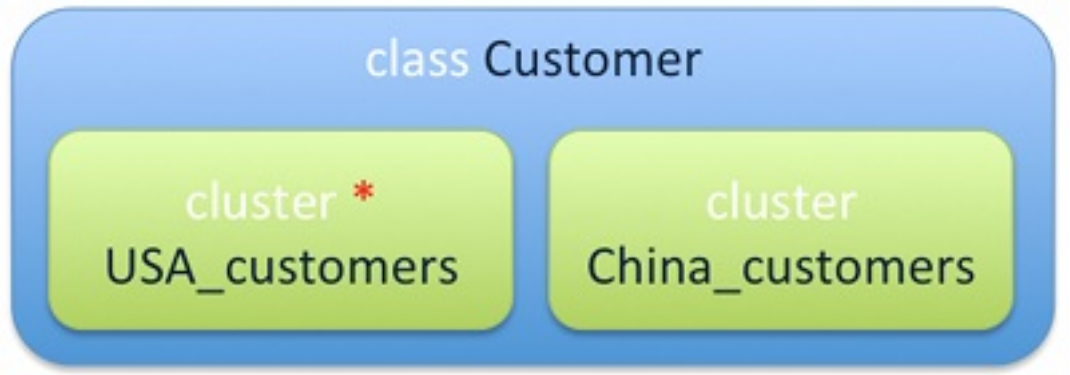
We've already talked about classes. A class is a logical concept in OrientDB. Clusters are also an important concept in OrientDB. Records (or documents/vertices) are stored in clusters.

What is a cluster?

A cluster is a place where a group of records are stored. Perhaps the best equivalent in the relational world would be a *Table*. By default, OrientDB will create one cluster per class. All the 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 (2^15-1) clusters in a database.

Understanding the concepts of classes and clusters allows you to take advantage of the power of clusters while designing your new database.

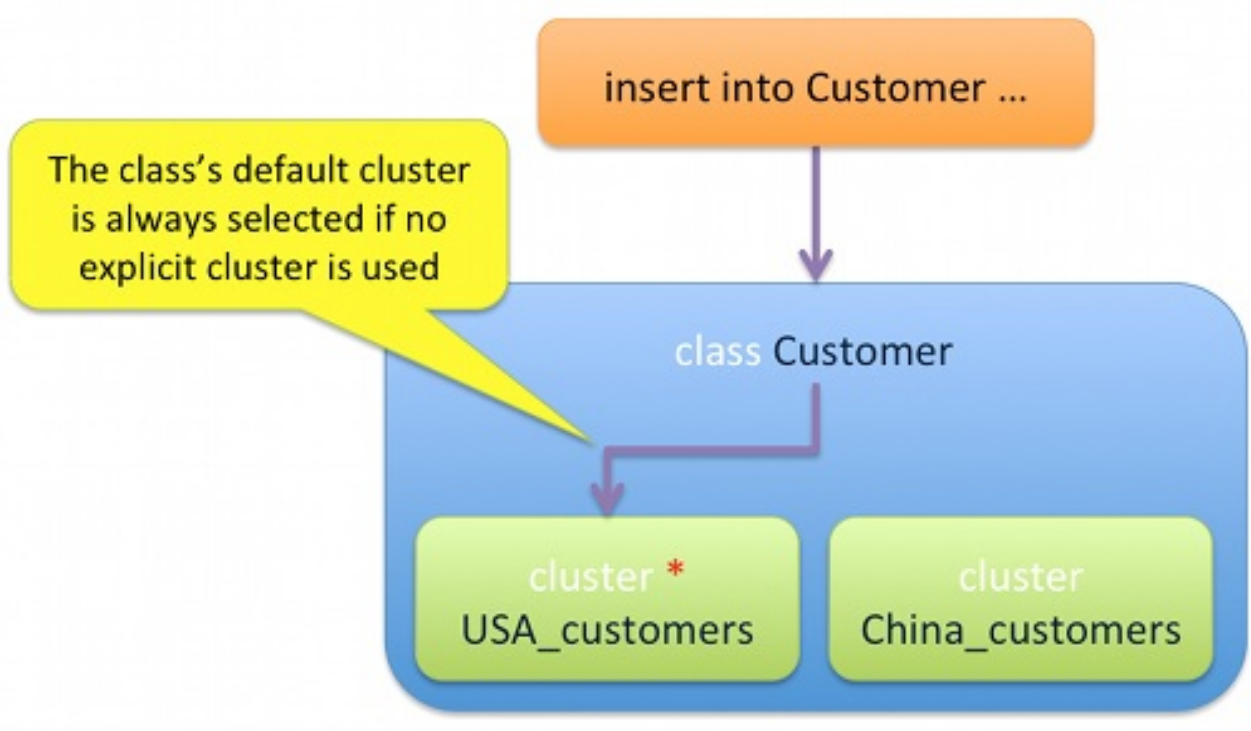
Even though the default strategy is that each class maps to one cluster, a class can rely on multiple clusters. You can spawn records physically in multiple places, thereby creating multiple clusters. For example:



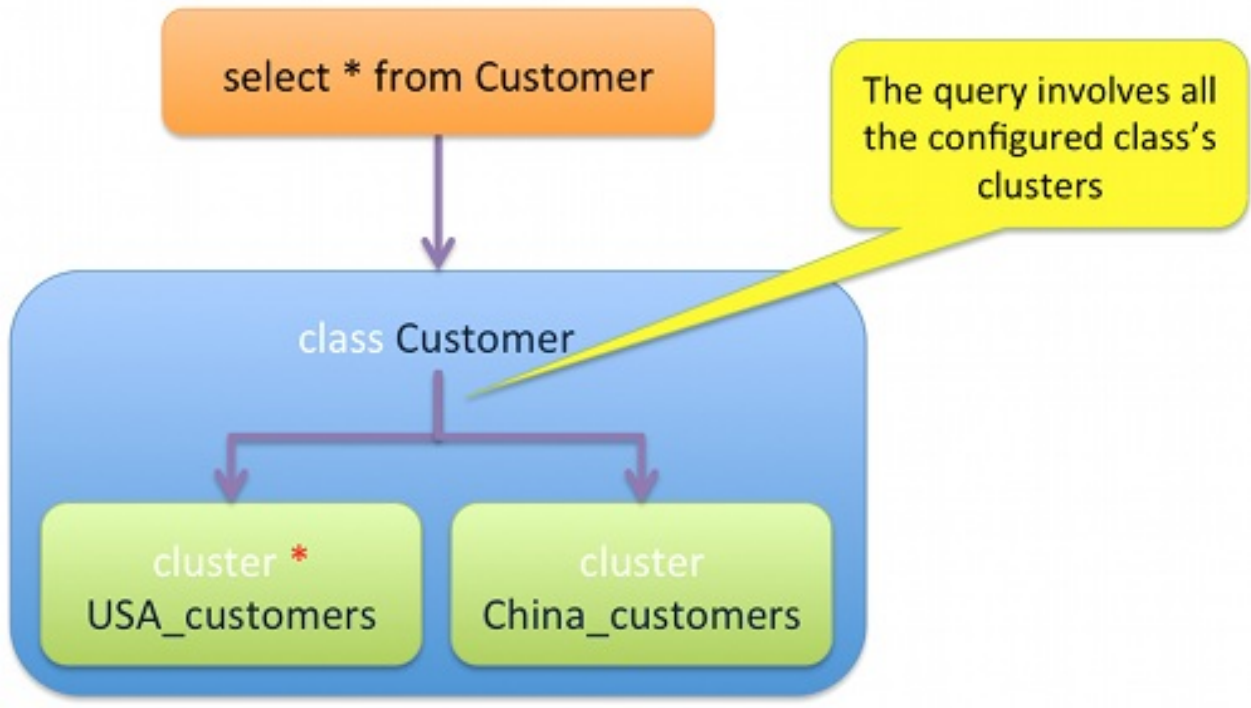
The class "Customer" relies on 2 clusters:

- *USA_customers*, containing all USA customers. This is the default cluster as denoted by the red star.
- *China_customers*, containing all Chinese customers.

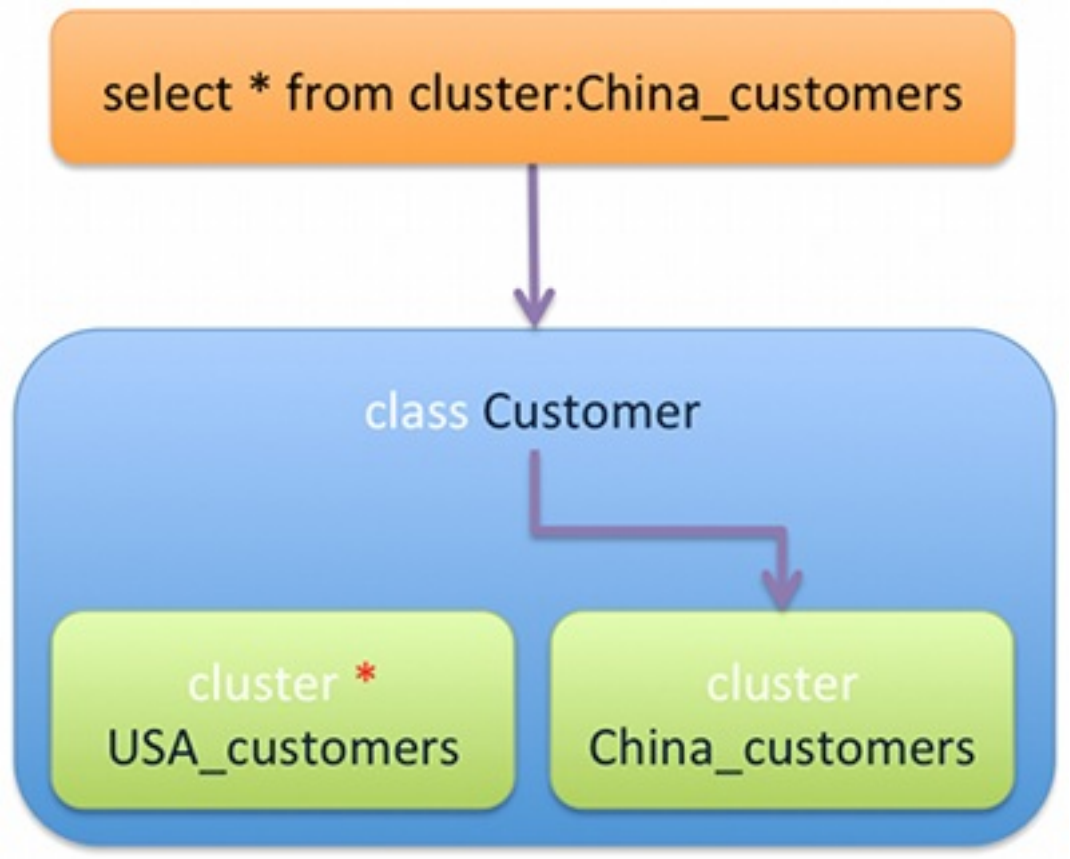
The default cluster (in this case, the USA_customers cluster) is used by default when the generic class "Customer" is used. Example:



When querying the "Customer" class, all the involved clusters are scanned:



If you know the location of a customer you're looking for you can query the target cluster *directly*. This avoids scanning the other clusters and optimizes the query:



To add a new cluster to a class, use the ALTER CLASS command. To remove a cluster use REMOVECLUSTER in ALTER CLASS command. Example to create the cluster "USA_Customers" under the "Customer" class:

```
ALTER CLASS Customer ADDCLUSTER USA_Customers
```

The benefits of using different physical places to store records are:

- faster queries against clusters because only a sub-set of all the class's clusters must be searched
- good partitioning allows you to reduce/remove the use of indexes
- parallel queries if on multiple disks
- sharding large data sets across multiple disks or server instances

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 the database must be

persistent. OrientDB creates physical clusters by default so you don't have to worry too much about it for now.

To view all clusters, from the console run the `clusters` command:

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

Since by default each class has its own cluster, we can query the database's users by class or by cluster:

```
orientdb> browse cluster OUser

---+---------+--------------------+--------------------+--------------------+--------------
  #| RID     |name                |password            |status              |roles
---+---------+--------------------+--------------------+--------------------+--------------
  0|     #5:0|admin               |{SHA-256}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8
  1|     #5:1|reader              |{SHA-256}3D0941964AA3EBDCB00CCEF58B1BB399F9F898465E9886D5
  2|     #5:2|writer              |{SHA-256}B93006774CBDD4B299389A03AC3D88C3A76B460D538795BC
---+---------+--------------------+--------------------+--------------------+--------------
```

The result is identical to `browse class ouser` executed in the classes section because there is only one cluster for the OUser class in this example.

The strategy where OrientDB selects the cluster when inserts a new record is configurable and pluggable. For more information take a look at Cluster Selection.

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>
```

- **cluster-id** is the id of the cluster. Each database can have a maximum of 32,767 clusters (2^15-1)
- **cluster-position** is the position of the record inside the cluster. Each cluster can handle up to 9,223,372,036,854,780,000 (2^63) records, namely 9,223,372 Trillion of records!

So the maximum size of a database is 2^78 records = 302,231,454,903 Trillion of records. We have never tested such high numbers due to the lack of hardware resources, but we most definitely have users working with OrientDB databases in the billions of records.

A RID (Record ID) is the physical position of the record inside the database. This means that loading a record by its RID is blazing fast, even with a growing database. With document and relational DBMS the more data you have, the slower the database will be. Joins have a heavy runtime cost. OrientDB handles relationships as physical links to the records. The relationship is assigned only once when the edge is created O(1). Compare this to an RDBMS that “computes“ the relationship every single time you query a database O(LogN). Traversing speed is not affected by the database size in OrientDB. It is always constant, whether for one record or 100 billion records. This is critical in the age of Big Data!

To load a record directly via the console, use the `load record` command. Below, we load the record #12:4 of the "demo" database.

```
orientdb> load record #12:4
----------------------------------------------------
ODocument - Class: Company   id: #12:4   v.8
----------------------------------------------------
           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's a *document*. OrientDB supports different types of records. This tutorial covers documents only.
- The *class* is "Company"
- The current *version* is 8. OrientDB has a MVCC system. It will be covered at a later point. Just know that every time you update a record its version is incremented by 1.
- We have different field types: "salary" and "salary2" are floats, "employees" and "id" are integers, "name" is a string, "initialized" and "checkpoint" are booleans and "created" is a date-time
- The field "addresses" has been NOT LOADED. It is also a LINK to another record #19:159. This is a relationship. (More explanation to follow)

SQL

Most NoSQL products have a custom query language. OrientDB focuses on standards when it comes to query languages. Instead of inventing "Yet Another Query Language", we started from the widely used and well understood SQL. We then extended it to support more complex graph concepts like Trees and Graphs. Why SQL? SQL ubiquitous in the database developer world, it is familiar. Also, it is more readable and concise than Map Reduce scripts.

Select

To start, let's write a query that returns the same result as the previous `browse cluster ouser` and `browse class ouser`:

```
select from OUser
```

Starting from this simple query, we can notice 2 interesting things:

- This query has no projections. This stands for "the entire record" like using the star (*).
- OUser is a class. By default queries are executed against classes.

The target can also be:

- a **cluster**, by prefixing with "cluster:". Example: `select from cluster:OUser`
- one or more **Record IDs**, by using the RecordID directly. For example: `select from #10:3` or `select from [#10:1, #10:3, #10:5]`
- an **index**, by prefixing with "index:". For example: `select value from index:dictionary where key = 'Jay'`

Similar to standard SQL, OrientDB supports WHERE conditions to filter the returning records by specifying one or more conditions. For example:

```
select from OUser where name like 'l%'
```

Returns all OUser records where the name starts with 'l'. For more information, look at all the supported operators and functions: SQL-Where.

OrientDB also supports the `ORDER BY` clause to order the result set by one or more fields. For example:

```
select from Employee where city = 'Rome' order by surname asc, name asc
```

This will return all of the Employees who live in Rome, ordered by surname and name in ascending order. You can also use the `GROUP BY` clause to group results. For example:

```
select sum(salary) from Employee where age < 40 group by job
```

This returns the sum of the salaries of all the employees with age under 40 grouped by job type. To limit the result set you can use the `LIMIT` keyword. For example, to limit the result set to maximum of 20 items:

```
select from Employee where gender = 'male' limit 20
```

Thanks to the SKIP keyword you can easily manage pagination. Use SKIP to pass over records from the result set. For example, to divide the result set in 3 pages you could do something like:

```
select from Employee where gender = 'male' limit 20
select from Employee where gender = 'male' skip 20 limit 20
select from Employee where gender = 'male' skip 40 limit 20
```

Now that we have the basic skills to execute queries, let's discuss how to manage relationships.

Insert

OrientDB supports ANSI-92 syntax:

```
insert into Employee (name, surname, gender) values ('Jay', 'Miner', 'M')
```

And the simplified:

```
insert into Employee set name = 'Jay', surname = 'Miner', gender = 'M'
```

Since OrientDB was created for the web, it can natively ingest JSON data:

```
insert into Employee content {name : 'Jay', surname : 'Miner', gender : 'M'}
```

Update

The ANSI-92 syntax is supported. Example:

```
update Employee set local = true where city = 'London'
```

Also using JSON with the "merge" keyword to merge the input JSON with current record:

```
update Employee merge { local : true } where city = 'London'
```

Delete

This also respects the ANSI-92 compliant syntax:

```
delete from Employee where city <> 'London'
```

Relationships

The most important feature of a **graph** database is the management of **relationships**. Many users come to OrientDB from MongoDB or other document databases because they lack efficient support of relationships.

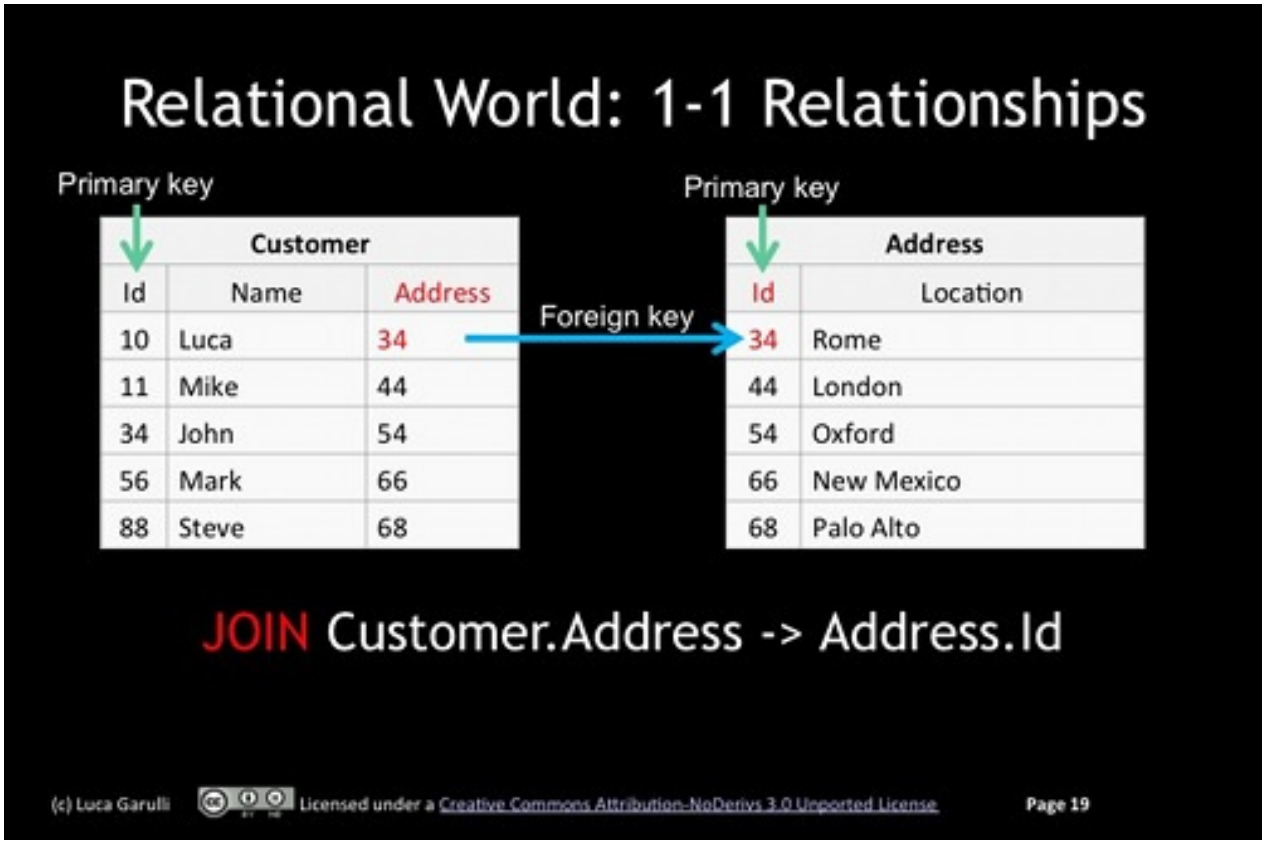
Relational Model

The relational model (and RDBMS - relational database management systems) has long been thought to be the best way to handle relationships. Graph databases suggest a more modern approach to this topic.

Most database developers are familiar with the relational model given it's 30+ years of dominance, spreading over generations of developers. Let's review how these systems manage relationships. As an example, we will use the relationships between the Customer and Address tables.

1-to-1 relationship

RDBMSs store the value of the target record in the "address" column of the Customer table. This is called a **foreign key**. The foreign key points to the **primary key** of the related record in the Address table:



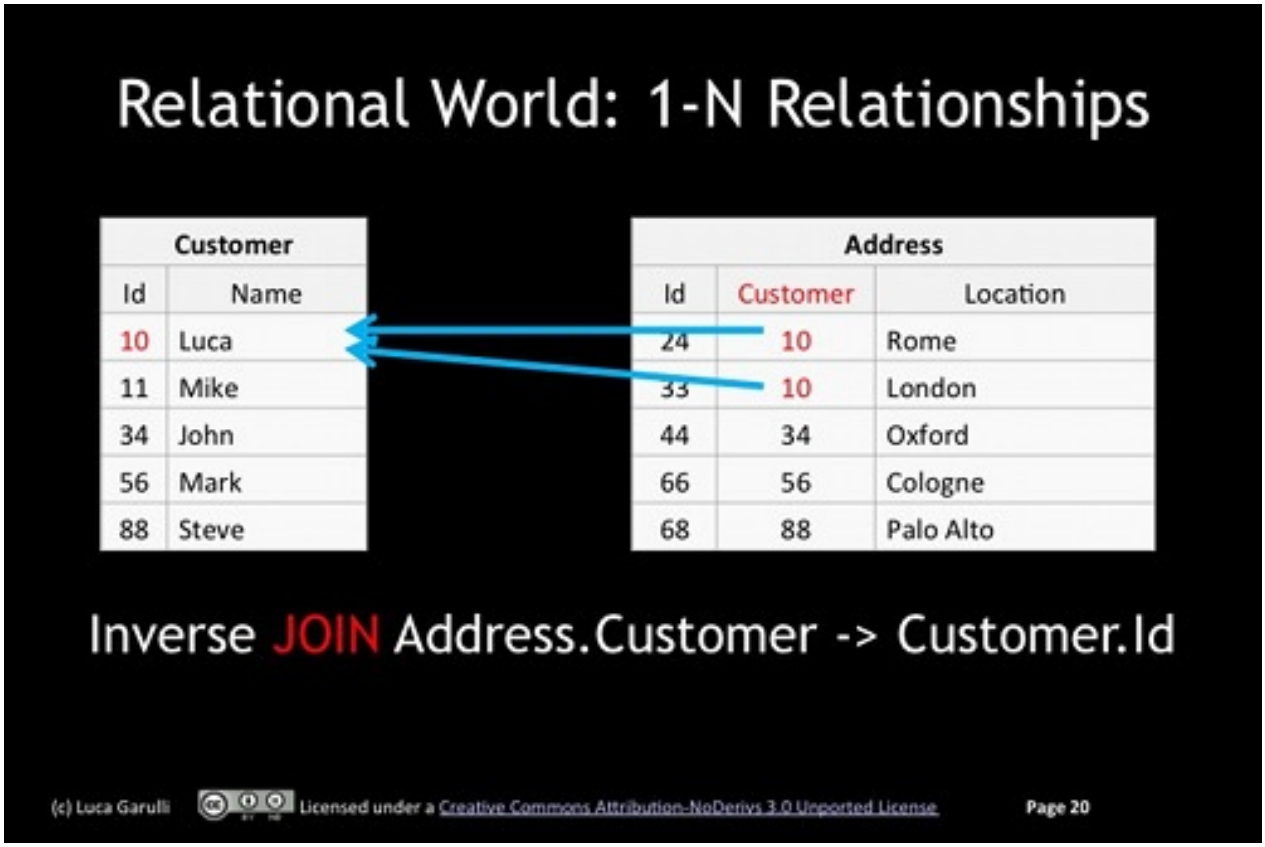
To retrieve the address pointed to by customer "Luca", the query in a RDBMS would be:

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

This is a **JOIN**! A JOIN is executed at run-time every time you retrieve a relationship.

1-to-Many relationship

Since RDBMS have no concept of collections the Customer table cannot have multiple foreign keys. The way to manage a 1-to-Many relationship is by moving the foreign key to the Address table.

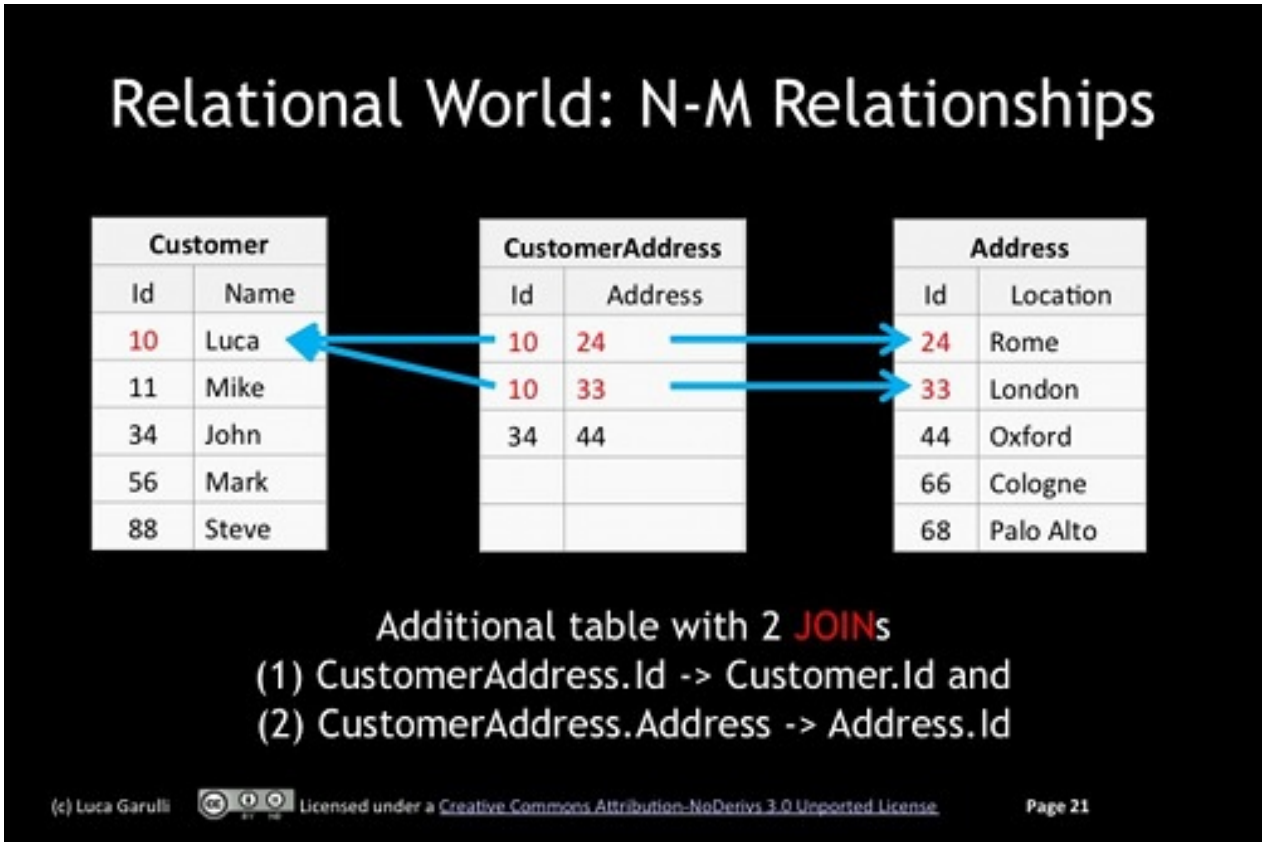


To extract all addresses of Customer 'Luca', the query in RDBMS reads:

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

Many-to-Many relationship

The most complex case is the Many-to-Many relationship. To handle this type of association, RDBMSs need a separate, intermediary table that matches both Customer and Addresses in all required combinations. This results in a **double JOIN** per record at runtime;



To extract all addresses of Customer 'Luca's the query in RDBMS becomes:

```
SELECT B.location FROM Customer A, Address B, CustomerAddress C WHERE A.name = 'Luca' AND B.
```

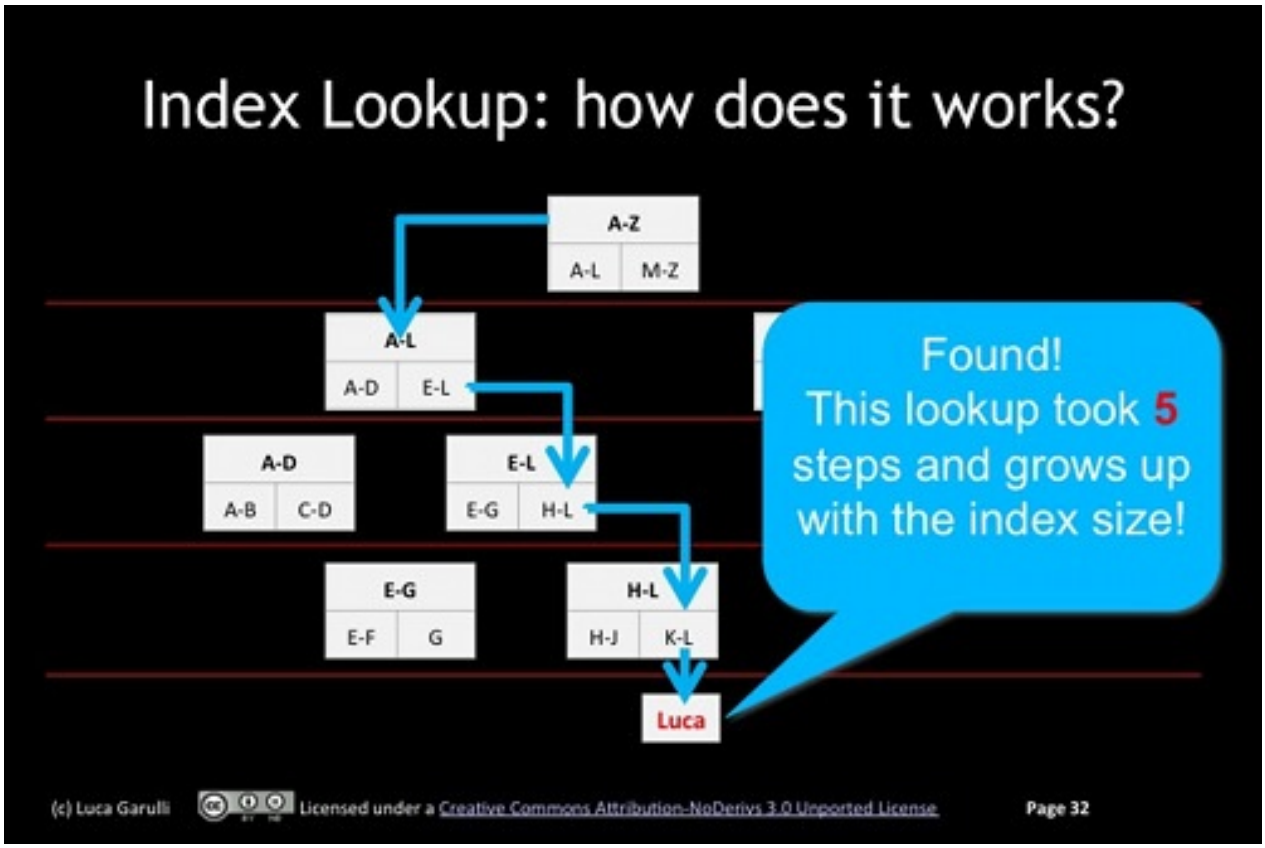
The problem with JOINS

With document and relational DBMS, the more data you have, the slower the database will perform. Joins have heavy runtime costs. In comparison, OrientDB handles relationships as physical links to the records, assigned only once when the edge is created O(1). Compare this to an RDBMS that “computes“ the relationship every single time you query a database O(LogN). With OrientDB, speed of traversal is not affected by the database size. It is always constant regardless if it has one record or 100 billion records. This is critical in the age of Big Data.

Searching for an ID at runtime each time you execute a query, for every record could be very expensive! The first optimization with RDMS is using indexes. Indexes speed up searches but they slow down `INSERT`, `UPDATE` and `DELETE` operations. In addition, they occupy substantial space on disk and in memory. You also need to qualify - are you sure the lookup into an index is actually fast? Let's try to understand how indexes work.

Do indexes solve the problem with JOIN?

The database industry has plenty of indexing algorithms. The most common in both Relational and NoSQL DBMS is the B+Tree. All balanced trees work in similar ways. Here is and example of how it would work when you're looking for "Luca": after only 5 hops the record is found.



But what if there were millions or billions of records? There would be many, many more hops. And this operation is executed on every JOIN per record! Imagine joining 4 tables with thousands of records: the number of JOINS could be in the millions!

Relations in OrientDB

OrientDB doesn't use JOINs. Instead it uses LINKs. A LINK is a relationship managed by storing the target RID in the source record. It's much like storing a pointer between 2 objects in memory. When you have Invoice -> Customer, then you have a pointer to Customer inside Invoice as an attribute. It's exactly the same. In this way it's like your database was in memory, a memory of several exabytes.

What about 1-to-N relationships? These relationships are handled as a collection of RIDs, like you would manage objects in memory. OrientDB supports different kinds of relationships:

- **LINK**, to point to one record only
- **LINKSET**, to point to several records. Like Java Sets, the same RID can only be included once. The pointers also have no order
- **LINKLIST**, to point to several records. Like Java Lists, they are ordered and can contain duplicates
- **LINKMAP**, to point to several records with a key stored in the source record. The Map values are the RIDs. Works like the Java `Map<?,Record>`.

Working with Graphs

We already met the Graph Model a few pages ago. Now we have all the basic knowledge needed to work with OrientDB as a GraphDB! This requires the graph edition of OrientDB. Connect to the `GratefulDeadConcerts` database for experimentation. It contains the concerts performed by the "Grateful Dead" band.

Create Vertexes and Edges

OrientDB comes with a generic Vertex persistent class called "V" (OGraphVertex in previous releases) and "E" (OGraphEdge in the past) for Edge. You can create a new Vertex with:

```
orientdb> insert into V set name = 'Jay'
create record with RID #9:0
```

In effect, the GraphDB model works on top of the underlying Document model, so all the stuff you have learned until now (Records, Relationships, etc.) remains valid. But in order to simplify the management of the graph we've created special commands, so don't use the SQL Insert command anymore to create a new vertex. Instead, use the ad-hoc "create vertex" command:

```
orientdb> create vertex V set name = 'Jay'
create vertex with RID #9:1
```

By using graph commands, OrientDB takes care of ensuring that the graph remains always consistent. All the Graph commands are:

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

Create custom Vertices and Edges classes

Even though you can work with Vertices and Edges, OrientDB provides the possibility to extend the `V` and `E` classes. The pros of this approach are:

- better understanding about meaning of entities
- optional constraints at class level
- performance: better partitioning of entities
- object-oriented inheritance among graph elements

So from now on, we will avoid using plain `V` and `E` and will always create custom classes. Let's develop an example graph to model a social network based on restaurants:

```
orientdb> create class Person extends V
orientdb> create class Restaurant extends V
```

Now that the schema has been created let's populate the graph with some vertices:

```
orientdb> create vertex Person set name = 'Luca'
create record with RID #11:0

orientdb> create vertex Person set name = 'Bill'
create record with RID #11:1

orientdb> create vertex Person set name = 'Jay'
create record with RID #11:2

orientdb> create vertex Restaurant set name = 'Dante', type = 'Pizza'
create record with RID #12:0

orientdb> create vertex Restaurant set name = 'Charlie', type = 'French'
create record with RID #12:1
```

Before we connect them using edges, let's go create a new Edge type:

```
orientdb> create class Eat extends E
```

This will represent the relationship from Person to Restaurant. The orientation is important when you create an edge because it gives the meaning of the relationship. If we wanted to model the edge in the opposite orientation, from Restaurant to Person, we might call the Edge class "Attendee", or something similar.

Now let's create a connection between person "Luca" and restaurant "Dante":

```
orientdb> create edge Eat from (select from Person where name = 'Luca') to (select from Rest
```

If you know the RID of vertices you can connect them with a shorter and faster command. Below we will connect "Bill" with the same "Dante" Restaurant and 'Jay' to 'Charlie' Restaurant:

```
orientdb> create edge Eat from #11:1 to #12:0
orientdb> create edge Eat from #11:2 to #12:1
```

Now that our small graph has been created let's play with queries. To cross edges we can use special graph functions like:

- `out()`, to retrieve the adjacent outgoing vertices
- `in()`, to retrieve the adjacent incoming vertices
- `both()`, to retrieve the adjacent incoming and outgoing vertices

To know all the people who eat in the "Dante" restaurant (RID = #12:0), we can get Dante's record and then traverse the incoming edges to discover the Person records connected:

```
orientdb> select in() from Restaurant where name = 'Dante'

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

Those are the RIDs of the Person instances connected. In these cases the `expand()` special function becomes very useful to transform the collection of vertices in the resultset 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   |
+-------+-------------+-------------+---------+
```

Much better! Now let's create the new relationship "Friend" to connect people:

```
orientdb> create class Friend extends E
```

And connect "Luca" with "Jay":

```
orientdb> create edge Friend from #11:0 to #11:2
```

"Friend" relationship is one of these edge types where the orientation is not important: if "Luca" is a friend of "Jay" the opposite is usually true, so the orientation looses importance. To discover Luca's friends, we should use the `both()` function:

```
orientdb> select expand( both('Friend') ) from Person where name = 'Luca'

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

In this case I've passed the Edge's class "Friend" as argument of the `both()` function to cross only the relationships of kind "Friend" (so skip the "Eat" this time). Note also in the result set that the relationship with "Luca" (RID = #11:0) is in the "in_" field.

Now let's make things more complicated. Get all the restaurants where Luca's friends go.

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

Cool, isn't it?

Lightweight edges

Starting from OrientDB v1.4.x edges, by default, are managed as **lightweight edges**: they don't have own identities as record, but are physically stored as links inside vertices. OrientDB automatically uses Lightweight edges only when edges have no properties, otherwise regular edges are used. From the logic point of view, **lightweight edges** are edges at all the effects, so all the graph functions work correctly. This is to improve performance and reduce the space on disk. But as a consequence, since lightweight edges don't exist as separate records in the database, the following query will not return the lightweight edges:

```
SELECT FROM E
```

In most of the cases Edges are used from Vertices, so this doesn't cause any particular problem. In case you need to query Edges directly, even those with no properties, disable **lightweight** edge feature by executing this command once:

```
ALTER DATABASE CUSTOM useLightweightEdges=false
```

This will only take effect for new edges. For more information look at: https://github.com/orientechnologies/orientdb/wiki/Troubleshooting#why-i-cant-see-all-the-edges.

For more information look at Graph API.

Using Schema with Graphs

OrientDB is a Graph Database "on steroids" because it supports concepts taken from both the Document Database and Object-Oriented worlds. This Tutorial step shows the power of Graphs used in conjunction with schema and constraints.

Take a look at this use case: Creating a graph to map the relationships between Person and Cars.

Let's open a shell (or command prompt in Windows) and launch the OrientDB Console (use `console.bat` on Windows):

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

Now we're going to use the console to create a brand new local database:

```
orientdb> create database plocal:../databases/cars admin admin plocal
```

Ok, now let's create the first graph schema with "Person" and "Car" as 2 new Vertex types and "Owns" as an Edge type:

```
orientdb> create class Person extends V
orientdb> create class Car extends V
orientdb> create class Owns extends E
```

And let's go populate the database with the first Graph elements:

```
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).
```

Ok, now we can traverse vertices. For example, what is Luca's car? Traverse from Luca vertex to the outgoing vertices following the "Owns" relationships:

```
orientdb> select name from ( select expand( out('Owns') ) from Person where name = 'Luca' )

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

Perfect. Now we want to have the location of each Person. We need another Vertex type called "Country" to connect to the Person with a new "Lives" Edge type:

```
orientdb> create class Country extends V
orientdb> create class Lives extends E

orientdb> create vertex Country set name = 'UK'

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

Next, let's associate Luca with the UK Country:

```
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).
```

So far so good. Our graph has been extended. Now, try to search the country where there are "Ferrari" cars in our database.

```
orientdb> select name from ( select expand( in('Owns').out('Lives') ) from Car where name li

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

Setting constraints on Edges

Now we've modeled our graph using a schema without any constraints. But it would be useful to require an Owns relationship to exist only between the Person and Car vertices. So, let's create these constraints:

```
orientdb> create property Owns.out LINK Person
orientdb> create property Owns.in LINK Car
```

The `MANDATORY` setting against a property prevents OrientDB from using a lightweight edge (no physical document is created). Be sure to pay attention and not put spaces between `MANDATORY=true`.

```
orientdb> alter property Owns.out MANDATORY=true;
orientdb> alter property Owns.in MANDATORY=true;
```

If we want to prohibit a Person vertex from having 2 edges against the same Car vertex, we have to define a UNIQUE index against out and in properties.

```
orientdb> create index UniqueOwns on Owns(out,in) unique

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

Unfortunately, the index tells us 0 entries are indexed. Why? We have already created the Owns relationships between "Luca" and "Ferrari Modena." In that case, OrientDB had already created a lightweight edge before we set the rule to force creation documents for Owns instances. So, 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)
```

Now check that the record has been created.

```
orientdb> select from Owns

----+-----+-----+-----
#   |@RID |out  |in
----+-----+-----+-----
```

```
0   |#13:0|#11:0|#12:0
----+-----+-----+-----
```

So far so good. The constraints works. Now try to create a "Owns" edge between Luca and UK (Country vertex):

```
orientdb> create edge Owns from (select from Person) to (select from Country)

Error: com.orientechnologies.orient.core.exception.OCommandExecutionException: Error on exec
Error: com.orientechnologies.orient.core.exception.OValidationException: The field 'Owns.in'
```

Now we have a typed graph with constraints.

For more information look at Graph Schema.

Setup a Distributed Database

OrientDB can run in a Distributed Architecture by sharing a database across multiple server instances.

For the purpose of this tutorial we're going to run 2 servers. There are 2 ways to share the same database across multiple nodes:

- Prior to startup, the database directory must be copied to all the servers. Copy & Paste-ing the database directory under the "databases" folder is enough.
- Alternately, keep the database on the first node running. The default configuration automatically shares the database with new servers that join.

Start the first server node

To start OrientDB in distributed mode, don't use `bin/server.sh` (or `.bat` on Windows), but rather the `bin/dserver.sh` (or `bin/dserver.bat`) script:

```
> cd bin
> ./dserver.sh

INFO OrientDB Server v1.6 (build 897) is starting up... [OServer]
INFO Databases directory: ./databases [OServer]
INFO Listening binary connections on 0.0.0.0:2424 (protocol v.18) [OServerNetworkListener]
INFO Listening http connections on 0.0.0.0:2480 (protocol v.10) [OServerNetworkListener]
INFO Installing dynamic plugin 'studio-1.6.zip'... [OServerPluginManager]
INFO Installing GREMLIN language v.2.5.0-SNAPSHOT - graph.pool.max=20 [OGraphServerHandler]
```

Note that the configuration file isn't `orientdb-server-config.xml` but the distributed version of it: `orientdb-dserver-config.xml`. For more information, look at Distributed Configuration.

The rest of the server startup log is below:

```
INFO Starting distributed server 'node1383734730415'... [OHazelcastPlugin]
INFO Configuring Hazelcast from ./config/hazelcast.xml'. [FileSystemXmlConfig]
INFO [10.1.28.101]:2434 [orientdb]
Members [1] {
    Member [10.1.28.101]:2434 this
}
[MulticastJoiner]
WARN [node1383734730415] opening database 'GratefulDeadConcerts'... [OHazelcastPlugin]
INFO [node1383734730415] loaded database configuration from disk: ./config/default-distribut
---------- [OHazelcastPlugin]
INFO [node1383734730415] adding node 'node1383734730415' in partition: GratefulDeadConcerts.
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":["<NEW_NODE>","node1383734730415"]("<NEW_NODE>","node1383734730415".md)
      }
    }
```

```
    }
  }
```

By reading the last piece of log we should notice that by default the `nodeId` is empty in `config/orientdb-dserver-config.xml`, so it's automatically assigned to random value: "node1383734730415" in this case. You should set a more familiar name like "europe0" or "production1".

Upon starting, OrientDB loads all the databases in the "databases" directory and configures them to run in distributed mode. For this reason, on the first load the default distributed configuration contained in `config/default-distributed-db-config.json` is copied into the database's directory. On subsequent starts, the file `databases/GratefulDeadConcerts/default-distributed-db-config.json` will be used instead of default configuration. This is because the shape of the cluster of servers changes every time nodes join or leave, and the configuration is kept updated by OrientDB on each server node.

To know more about the meaning of the configuration contained in the `config/default-distributed-db-config.json` file look at Distributed Configuration.

Start the second server node

Now start the second server like the first one. Make sure that both servers have the same Hazelcast's credentials to join the same cluster in the `config/hazelcast.xml` file. The fastest way to do this is to copy & paste the OrientDB directory from the first node to the other ones. If you run multiple server instances in the same host (makes sense only for testing purpose) remember to change the port in `config/hazelcast.xml`.

Once the other node is online, both nodes see each other and dump a message like this:

```
WARN [node1384014656983] added new node id=Member [192.168.1.179]:2435 name=null [OHazelcast
INFO [192.168.1.179]:2434 [orientdb] Re-partitioning cluster data... Migration queue size: 1
INFO [192.168.1.179]:2434 [orientdb] All migration tasks has been completed, queues are empt
INFO [node1384014656983] added node configuration id=Member [192.168.1.179]:2435 name=node13
INFO [node1384014656983] update configuration db=GratefulDeadConcerts from=node1384015873680
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":["<NEW_NODE>","node1383734730415","node1384015873680"]("<NEW_NODE>","node13
    }
}
    },
  "version":1
}
---------- [OHazelcastPlugin]
WARN [node1383734730415]->[node1384015873680] deploying database GratefulDeadConcerts... [OD
WARN [node1383734730415]->[node1384015873680] sending the compressed database GratefulDeadCo
```

In this case 2 server nodes were started on the same machine (ip=10.37.129.2), but with 2 different ports (2434 and 2435 where the current is "this"). The rest of the log is relative

to the distribution of the database to the second server.

On the second server node output you'll see these messages:

```
WARN [node1384015873680]<-[node1383734730415] installing database GratefulDeadConcerts in da
WARN [node1384015873680] installed database GratefulDeadConcerts in databases/GratefulDeadCo
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]
```

This means that the database "GratefulDeadConcerts" has been correctly installed from the first node (node1383734730415) through the network.

Working with Distributed Graphs

Once a server has joined the distributed cluster, all the clients are constantly notified about it so that in case of failure they will switch transparently to the next available server. Check this by using the console. When OrientDB runs in distributed configuration, the current cluster shape is visible with the `info` command.

```
$ cd 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)

Cluster configuration:
```


```
{
  "members":[{
    "name":"node1384015873680",
    "listeners":[{"protocol":"ONetworkProtocolBinary","listen":"192.168.1.179:2425"},{"proto
    "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"},{"proto
    "id":"5cb7972e-ccb1-4ede-bfda-c835b0c2e5da",
    "startedOn":"2013-11-09 17:30:56",
    "databases":[]
    }],
  "localName":"_hzInstance_1_orientdb",
  "localId":"5cb7972e-ccb1-4ede-bfda-c835b0c2e5da"
}
```


Now let's create a new vertex by connecting with the console against Node1:

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

Now from another shell (Command Prompt on Windows) connect to the node2 and execute this command:

```
orientdb> select from v where node = 1
----+------+----
#   |@RID  |node
----+------+----
0   |#9:815|1
----+------+----
1 item(s) found. Query executed in 0.19 sec(s).
```

The vertex has been correctly replicated on Node2. Cool! Now kill the node1 process. You will see these messages in the console of node2:

```
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
INFO [192.168.1.179]:2435 [orientdb] Partition balance is ok, no need to re-partition cluste
```

Node2 recognizes that Node1 is unreachable. Let's see if the console connected to the node1 reports the failure. Test it by executing a query:

```
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
WARN Connection re-acquired transparently after 30ms and 1 retries: no errors will be thrown
----+----+-------------+---------+------------+----+--------------+-------------+--------
#   |@RID|name         |song_type|performances|type|out_followed_by|out_written_by|out_sung
----+----+-------------+---------+------------+----+--------------+-------------+--------
1   |#9:1|HEY BO DIDDLEY|cover    |5           |song|[5]           |#9:7          |#9:8
2   |#9:2|IM A MAN      |cover    |1           |song|[2]           |#9:9          |#9:9
----+----+-------------+---------+------------+----+--------------+-------------+--------
```

Wow! The console auto switched to the next available node2. The warning reports that everything happens in a transparent way, so the application doesn't need to manage this.

Now, from the console connected to the Node2, create a new vertex:

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

The operation has been journaled to be synchronized to the Node1 once it comes online again. Now let's restart Node1 and see if auto re-alignment succeeds. Connect with the console against Node1 to check if the node has been aligned after the restart:

```
orientdb> select from v where node = 2
----+------+----
#   |@RID  |node
----+------+----
0   |#9:816|2
----+------+----
1 item(s) found. Query executed in 0.209 sec(s).
```

Aligned! You can do the same with N servers where every server is a Master. There are no limits on the number of running servers. With many servers across a not-fast network, you could tune the network timeouts to be more permissive and let a big, distributed cluster of servers do the work properly.

For more information look at Distributed Architecture.

Java Tutorial

If you're only used to working with traditional RDBMS databases, you'll find that OrientDB is a very different beast. Since OrientDB is able to support document mode, graph mode, and object-oriented mode, different Java APIs are required. But there are some similarities too: in a roughly similar way to JDBC, a Blueprints API exists, made by Tinkerpop, which supports the basic operations on a graph database. There is an OrientDB "driver" (or, better, "adapter") which makes it possible to operate without having to deal with OrientDB classes, and the resulting code should be mainly portable (Blueprints offers more adapters for other graph database products).

In any case, if you need to tweak the database configuration, you need to use the OrientDB APIs directly. It's a good idea to use a mix: Blueprints when you can and the OrientDB APIs when you need them.

OrientDB Java APIs

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

- Graph API
- Document API
- Object API

Graph API

Connecting to a database

The first object you need is a `Graph` or a `TransactionalGraph` (which supports transaction demarcation):

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

TransactionalGraph graph = new OrientGraph("local:test", "username", "password");
// or TransactionalGraph graph = new OrientGraph("remote:localhost/test", "username", "passw
```

In the following examples, until we introduce transactions, `TransactionalGraph` and `Graph` are used interchangeably.

Another possibility is to create the database connection with the OrientDB APIs (this would make it possible to call tuning APIs, for instance), and then "wrap" it into an `OrientGraph`:

```
import com.orientechnologies.orient.core.db.graph.OGraphDatabase;
import com.tinkerpop.blueprints.TransactionalGraph;
import com.tinkerpop.blueprints.impls.orient.OrientGraph;

OGraphDatabase odb = new OGraphDatabase("local:test").create();
TransactionalGraph graph = new OrientGraph(odb);
```

In any case, from a `TransactionalGraph` (or a `Graph`) it's always possible to get a reference to the OrientDB API:

```
import com.tinkerpop.blueprints.impls.orient.OrientGraph;
import com.orientechnologies.orient.core.db.graph.OGraphDatabase;

OGraphDatabase odb = ((OrientGraph)graph).getRawGraph();
```

Even though OrientDB can work with the generic class "V" for Vertices and "E" for Edges, it's much more powerful to define custom types for both Vertices and Edges:

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

The Blueprint adapter is **thread-safe** and will **automatically create a transaction when needed** (e.g. at the first operation if a transaction hasn't been explicitly started). You have to specify where the transaction ends (commit or rollback) - see below for more details.

Inserting vertices and edges

To add vertices into the database with the Blueprints API:

```
import com.tinkerpop.blueprints.Graph;
import com.tinkerpop.blueprints.Vertex;

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");
```

Note the specific Blueprints syntax `"class:<class name>"` that you must use in the creation of an object to specify its class. It is not mandatory: it is also possible to specify a `null` value, which means that a vertex will be created with the class `OGraphVertex`, as it's the superclass of all vertices in OrientDB):

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

A consequence is that we won't be able to distinguish it from other vertices in a query.

To add an edge a similar API is used:

```
import com.tinkerpop.blueprints.Graph;
import com.tinkerpop.blueprints.Edge;

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

We passed `null` to `addEdge()`, so we created an edge with the `OGraphEdge` class, which is the superclass of all edges in OrientDB. A consequence is that in a query we won't be able to distinguish it from other edges (except for its label).

The Blueprints adapter **automatically saves changes to the database** (in contrast to the native OrientDB API, which requires an explicit call to `save()`). Please remember that saving is a different operation than committing.

Now we have created

```
[John Smith:Person] --[lives]--> [Van Ness Ave. …:Address]
```

Please note that, in this example, we have used a partially schema-full mode, as we defined the vertex types, but not their properties. OrientDB will dynamically accept everything.

More on Tutorials

This is a collection of tutorials about OrientDB.

Video tutorials
Getting Started with OrientDB by Petter Graff

External tutorials

Miscellaneous

- 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:

- http://java.html.it/articoli/leggi/4039/nosql-in-java-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):

- http://d.hatena.ne.jp/tm8r/20120416/1334581009

Make GraphDB OrientDB app deployment experience:

1. Part 1: http://fungoing.jp/2011/08/379

Step-by-step tutorial by Junji Takakura:

- Part 1: http://snakemanshow.blogspot.com/2010/09/nosql-orientdb-1.html
- Part 2: http://snakemanshow.blogspot.com/2010/09/nosql-orientdb-2.html
- Part 3: http://snakemanshow.blogspot.com/2011/04/nosql-orientdb-3.html

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

- Video : **NoSQL adoption: what's the next step?** by Luca Garulli at NoSQL Matters in Cologne (Germany) on May 30th 2012

    -
    - Slides

- Video : **Design your applications using persistent Graphs and OrientDB** by Luca Garulli at NoSQL Matters in Cologne (Germany) on May 30th 2012

    -
    - Slides (English)

- Video (English): **Works with persistent graphs using OrientDB** by Luca Molino at FOSDEM on February 2012 in Bruxelles (Belgium) on Video
    - Slides (English)

- Video (pseudo-English): **Interview to Luca Garulli about OrientDB** by Tim Anglade on 2010

°

Presentations in English

- Slides (English): OrientDB distributed architecture 1.1
- Slides (English): OrientDB the database for the Web 1.1
- What to select between the Document database and the Graph one?
- A walk in Graph Databases

Videos in Italian, Presentations in English

- Video (Italian): **Graph databases: time for the serious stuff** by Alessandro Nadalin and David Funaro at Codemotion in Rome on March 2012
  - Video
  - Slides (English)

- Video (Italian): **Dal modello Relazionale al Grafo: cosa cambia?** by Alfonso Focareta at Codemotion in Rome on March 2012
  - Video
  - Slides (English)

- Video (Italian): **Perché potresti avere bisogno di un database NOSQL anche se non sei Google o Facebook** (Italian only) by Luca Garulli at Codemotion in Rome (Italy) on March 2011
  - http://www.orientdb.org/images/video-2011-codemotion-roma.png
  - Slides (English)

- Video (Italian): **OrientDB e lo sviluppo di WebApp** (Italian only) by Luca Garulli at NoSQL Day in Brescia on 2011

  - Slides (English)

Basic Concepts

Record

A record is the smallest unit that can be loaded from and stored into the database.

Record types

There are several types of records.

Document

Documents are the most flexible record available in OrientDB. They are softly typed and are defined by schema classes with defined constraints, but can also be used in schema-less mode. Documents handle fields in a flexible way. A Document can be easily imported and exported in JSON format. Below is an example of a Document in JSON format:

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

OrientDB Documents support complex relationships. From a programmer's perspective this can be seen as a sort of persistent Map.

Flat

Records are strings. No fields are supported, no indexing, no schema.

RecordID

In OrientDB, each record has an auto assigned Unique ID. The RecordID (or RID) is composed in this way:

```
#[<cluster>:<position>]
```

Where:

- cluster is the cluster id. Positive numbers mean persistent records. Negative numbers mean temporary records, like those used in result sets for queries that use projections.
- position is the absolute position of the record inside a cluster.

NOTE: The prefix character # is mandatory to recognize a RecordID.

The record never loses its identity unless it is deleted. Once deleted its identity is never recycled (but with "local" storage). You can access a record directly by its RecordID. For this reason you don't need to create a field as a primary key like in a Relational DBMS.

Record version

Each record maintains its own version number that is incremented at every update. In optimistic transactions the version is checked in order to avoid conflicts at commit time.

Class

A Class is a concept taken from the Object Oriented paradigm. In OrientDB it defines a type of record. It's the closest concept to a Relational DBMS Table. Classes can be schema-less, schema-full, or mixed.

A class can inherit from another, creating a tree of classes. Inheritance means that a sub-class extends a parent class, inheriting all its attributes.

Each class has its own clusters. A class must have at least one cluster defined (its default cluster), but can support multiple ones. When you execute a query against a class, it's automatically propagated to all the clusters that are part of the class. When a new record is created, the cluster that is selected to store it is picked by using a configurable strategy.

When you create a new class by default a new persistent cluster is created with the same name of the class in lowercase.

Abstract Class

If you know Object-Orientation you already know what an abstract class is. For all the rest:

- http://en.wikipedia.org/wiki/Abstract_type
- http://docs.oracle.com/javase/tutorial/java/IandI/abstract.html For our purpose, we can sum up an abstract class as:
- A class used as a foundation for defining other classes (eventually, concrete classes)
- A class that can't have instances

To create a new abstract class look at SQL Create Class.

Abstract classes are essential to support Object Orientation without the typical spamming of the database with always empty auto-created clusters. *NOTE: available since 1.2.0*

When do you use a class or a cluster in queries?

Let's use an example: Let's assume you created a class "Invoice" and two clusters "invoice2011" and "invoice2012".

You can now query all the invoices by using the class as a target in the SQL select:

```
SELECT FROM Invoice
```

If you want to filter per year (2012) and you've created a "year" field in the Invoice class do this:

```
SELECT FROM Invoice where year = 2012
```

You may also query specific objects from a single cluster (so, by splitting the Class Invoice in multiple clusters, e.g. one per year, you narrow your candidate objects):

```
SELECT FROM cluster:invoice2012
```

This query may be significantly faster because OrientDB can narrow the search to the targeted cluster.

The combination of Classes and Clusters is very powerful and has many use cases.

Relationships

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

Referenced relationships

Relationships in OrientDB are managed natively without computing costly JOINs, as in a Relational DBMS. In fact, OrientDB stores direct link(s) to the target objects of the relationship. This boosts up the load speed of the entire graph of connected objects like in Graph and Object DBMSs. Example:

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

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

1-1 and N-1 referenced relationships

These kinds of relationships are expressed using the **LINK** type.

1-N and N-M referenced relationships

These kinds of 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 accepts duplicates.
- **LINKMAP**, as an ordered map of links with **String** as the key type. A keys doesn't accept duplicates.

Embedded relationships

Embedded records, instead, are contained inside the record that embeds them. It's a kind of relationship that's stronger than the reference. It can be represented like the UML Composition relationship. The embedded record will not have its own RecordID, since it can't be directly referenced by other records. It's only accessible through 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

These kinds of relationships are expressed using the **EMBEDDED** type.

1-N and N-M embedded relationships

These kinds of relationships are expressed using a collection of links such as:

- **EMBEDDEDLIST**, as an ordered list of records.
- **EMBEDDEDSET**, as an unordered set of records. It doesn't accept duplicates.
- **EMBEDDEDMAP**, as 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 Graph Model edges (connections between vertices) are bi-directional. This differs from the Document Model where relationships are always mono-directional, thus requiring the developer to maintain data integrity. In addition, OrientDB automatically maintains the consistency of all bi-directional relationships (aka edges).

Database

A database is an interface to access the real Storage. The database understands high-level concepts like Queries, Schemas, Metadata, Indices, etc. OrientDB also provides multiple database types. Take a look at the Database types to learn more about them.

Each server or JVM can handle multiple database instances, but the database name must be UNIQUE. So you can't manage two databases named "customer" in two different directories at the same time. To handle this case use the `$` (dollar) as a separator instead of `/` (slash). OrientDB will bind the entire name, so it will be unique, but at the file system level it will convert `$` with `/` allowing multiple databases with the same name in different paths. Example:

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

The database must be opened as:

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

Database URL

OrientDB has its own URL format:

```
<engine>:<db-name>
```

Where:

- **db-name** is the database name and depends on the engine used (see below)
- **engine** can be:

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

The database must always be closed once you've finished working with 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
0	Boolean	Handles only the values *True* or *False*	`java.lang.Boolean` or `boolean`	0 1
1	Integer	32-bit signed Integers	`java.lang.Integer` or `int`	-2,147,483,648 +2,147,483,647
2	Short	Small 16-bit signed integers	`java.lang.Short` or `short`	-32,768 32,767
3	Long	Big 64-bit signed integers	`java.lang.Long` or `long`	-2^{63} $+2^{63}-1$
4	Float	Decimal numbers	`java.lang.Float` or `float`	2^{-149} $(2-2^{-23})*2^{127}$
5	Double	Decimal numbers with high precision	`java.lang.Double` or `double`	2^{-1074} $(2-2^{-52})*2^{1023}$
6	Datetime	Any date with the precision up to milliseconds	`java.util.Date`	- 1002020303
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
9	Embedded	The Record is contained inside the owner. The contained Record has no RecordId	`ORecord`	- -
		The Records are contained inside the		

10	Embedded list	owner. The contained records have no RecordIds and are reachable only by navigating the owner record	`List<Object>`	0 41,000,000 items
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
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
13	Link	Link to another Record. It's a common one-to-one relationship	`ORID , <? extends ORecord>`	1:-1 32767:2^63-1
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

15	Link set	Links to other Records. It's a common one-to-many relationship	`Set<? extends ORecord>`	0 41,000,000 items
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
17	Byte	Single byte. Useful to store small 8-bit signed integers	`java.lang.Byte` or `byte`	-128 +127
18	Transient	Any value not stored on database		
19	Date	Any date as year, month and day	`java.util.Date`	--
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`	? ?
22	LinkBag	List of RecordIds as spec RidBag	`ORidBag`	? ?
23	Any	Not determinated type, used to specify Collections of mixed type, and null	-	-

Inheritance

OrientDB doesn't split Documents between different classes (as many OR-Mapping tools do). Each Document will reside in the cluster of its most base class. When you execute a query against a class that has sub-classes, OrientDB will search into the clusters of the target class and all its sub-classes. To know more see How it works.

Declare in schema

OrientDB needs to know the relationships between the class inheritance. Note that this is an abstract concept that applies to both POJOs and Documents.

Example:

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

Polymorphic Queries

By default all the queries are polymorphics. Using the example above with this SQL query:

```
SELECT FROM account WHERE name = 'Google'
```

Will be returned all the instances of Account and Company classes that has the property name equals to "Google".

How it works

Consider this example. We have 3 classes, with the cluster-id between parenthesis:

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

OrientDB, by default, creates a separate cluster for each class.

This cluster is indicated by the "**defaultClusterId**" property in OClass class and indicates the cluster used by default when not specified. However the OClass has the property "**clusterIds**" (as int[]) that contains all the clusters able to host the records of that class.

By default "clusterIds" and "defaultClusterId" are the same.

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

In this way when you execute this:

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

Will return all the records for "GOOGLE" contained in all the three classes because for the class "Account" OrientDB searches inside the clusters 10, 13 and 27 following the inheritance specified in the schema.

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 gain access to the schema APIs you need in the Schema instance of the database you're using.

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

Class

A Class 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. This means that the sub-class extends the parent one inheriting all the attributes.

Each class has its 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 of the class in lower-case.

Create a persistent class

Each class contains one or more properties. This mode is similar to the classic Relational DBMS approach where you define tables before storing records.

Example of creation of Account class. By default a new [Cluster](Concepts#cluster] will be created to keep the class instances:

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

Get a persistent class

To retrieve a persistent class use the `getClass(String)` method. If the class not exists 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 be not deleted unless you explicitly delete them before to drop 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 class level by calling the `setStrictMode(true)` method. In this case record of that class can't have not-defined properties.

Property

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

Create the Class property

Once the class has been created, you can define fields (properties). Below 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 supported 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 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 costly JOINs of 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 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);
```

Embedded relationships

Embedded records, instead, are contained inside the record that embeds them. It's a kind of relationship stronger than the reference. The embedded record will not have a own 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 of records as value with key a **String**. It doesn't accepts duplicated 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 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.

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

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 let a property value to be UNIQUE use the UNIQUE index as constraint:

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

To let to a group of properties to be UNIQUE create a composite index made of multiple fields:

Creation of composite index:

```
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 specifying its Class, OrientDB automatically selects the Class where to store the physical record, by using configurable strategies.

The available 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. Calculation of cluster size is made every 5 or more seconds to avoid to slow down insertion
- **local**. This is injected when OrientDB is running in distributed mode. With this strategy the cluster that is the master on current node is always preferred. This avoids conflicts and reduces network latency with remote calls between nodes.

Create custom strategy

To create your custom strategy follow the following steps:

1) Create the implementation in Java

The class must implements interface OClusterSelectionStrategy. Example:

```
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"; }
}
```

Note that the method `getCluster()` receives also the ODocument to insert. This is useful if you want to assign the clusterId based on the Document content.

2) Register the implementation as service

Create a new file under META-INF/services called `com.orientechnologies.orient.core.metadata.schema.clusterselection.OClusterSelectionStrategy` and write your class with full package.

Example of the content:

```
mypackage.RandomSelectionStrategy
```

This is the default content in OrientDB core is:

```
com.orientechnologies.orient.core.metadata.schema.clusterselection.ORoundRobinClusterSelecti
com.orientechnologies.orient.core.metadata.schema.clusterselection.ODefaultClusterSelectionS
com.orientechnologies.orient.core.metadata.schema.clusterselection.OBalancedClusterSelection
```

3) Assign it

To assign your new strategy to a class, use the ALTER CLASS command. Example:

```
ALTER CLASS Employee CLUSTERSELECTION random
```

Fetching Strategies

OrientDB supports fetching strategies by using the **Fetch Plans**. Fetch Plans are used to customize how OrientDB must load linked records.

Example:

```
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   ]
```

By default OrientDB loads all the linked records in lazy way. So in this example the linked "customer", "city" and "orders" fields are not loaded until are traversed. If you need the entire tree it could be slow the lazy loading of every single linked record. In this case it would need 7 different loads. If the database is open on a remote server they are 7 different network calls.

This is the reason why OrientDB supports custom fetching strategies using the **Fetch Plans**. The aim of fetch plans is to pre-load connected records in one shot.

Where use fetch-plans?

- On record loading through the remote connection
- On JSON serializer to produce JSON with nested records

Remote connection

When a client executes a query (or load directly one single record) setting a fetch plan with level different to 0, then the server traverses all the records of the returning result set and sends them to the client in the same call.

The client avoid to connect directly them to the record by using always the lazy collections (i.e.: OLazyRecordList). Instead, loads all the connected records into the local client. In this ways the collections remain lazy but when you're accessing to the content, the record is early loaded from the local cache avoiding other connections.

Format

The fetch plan comes in form of a String and can be used at run-time on:

- query
- record loading

The syntax is:

```
[[levels]]<fieldPath>:<depth-level>*
```

Where:

- **levels**, optional, tells at which levels the rules must apply. Levels starts from 0. Since 2.1. Supported syntax is:
  - **level**, example `[0]` to apply only at first level
  - **ranges**, example `[0-3]` form 0 to 3th level. Ranges can be also partial, like `[-3]` means 0-3 and `[3-]` means form 3rd to infinite
  - **any**, by using `*`. Example `[*]` to apply at any level

- **fieldPath**, is the field name path, expected in dot notation, starting from the root record or the wildcard `*` *for "any" field. The wildcard* can be also at the end of the path to specify all the paths that starts for a name
- **depth-level**, is the deep level requested:
  - 0 = Load only current record,
  - 1-N = load only the first-Nth connected record,
  - -1 = unlimited,
  - -2 = exclude it

To express multiple rules separate them by spaces.

Examples with the record tree above:

- `"*:-1"` : fetches the entire tree recursively
- `"*:-1 orders:0"` : fetches all the records recursively but the "orders" field in root class. Note that in "orders" field will be loaded only its direct content (only records 8:12,8:19,8:23, none of other records inside them will be loaded).
- `"*:0 address.city.country:0"` : fetches only not-document fields in the root class and address.city.country field (records 10:1,11:2,12:3).

- `"[*]in_*:-2 out_*:-2"` : returns all the properties, but edges (at any level)

Circular dependencies

OrientDB handles circular dependencies to avoid any loop while fetches linking records.

Example using the Java APIs

Execute a query with a custom fetch plan

```
List<ODocument> resultset = database.query(new OSQLSynchQuery<ODocument>("select * from Prof
```

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.*

Indexes

OrientDB supports 4 kinds of indexes:

Index type	Durable	Transactional	Range queries	Best features	Description
SB-Tree	**YES**	**YES**	**YES**	Good mix of all	the default one is durable and transactional
Hash	**YES**	**YES**	no	Super fast lookup, very light on disk	Works like a HashMap so it's faster on punctual lookup (select from xxx where salary = 1000) and consumes less resources, but you cannot use it for range queries (select from xxx where salary between 1000 and 2000)
Lucene	**YES**	**YES**	**YES**	Good on full-text and spatial indexes	Lucene indexes can be used only for full-text and spatial

What's an index?

Indexes can be handled like classes (or tables for RDBMS users) using the SQL language and prefixing with "index:" the index name. The index is like a class (or table) with 2 properties:

- **key**, as the index's key
- **rid**, as the RecordId that points to the record associated with the key

Index target

Indexes can be updated:

- **Automatically** when are bound to schema properties, 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 indexing.
- **Manually**, handled by the developer using Java API and SQL commands (see below). You can use them as Persistent Maps where they entry's value are the records pointed by index.

Index types

The index type cannot be changed once created. The supported index types are the following:

- **SB-Tree** algorithm:
  - **UNIQUE**, doesn't allow duplicates. For composite index means uniqueness of composite keys.
  - **NOTUNIQUE**, allows duplicates
  - **FULLTEXT**, by indexing any single word of the text. It's used in query with the operator CONTAINSTEXT
  - **DICTIONARY**, like UNIQUE but in case the key already exists replace the record with the new one

- **HashIndex** algorithm:
  - **UNIQUE_HASH_INDEX**, doesn't allow duplicates. For composite index means uniqueness of composite keys.
  - **NOTUNIQUE_HASH_INDEX**, allows duplicates
  - **FULLTEXT_HASH_INDEX**, by indexing any single word of the text. It's used in query with the operator CONTAINSTEXT
  - **DICTIONARY_HASH_INDEX**, like UNIQUE but in case the key already exists replace the record with the new one

- **Lucene** engine:

  - **FULLTEXT**, it uses Lucene to index the string content. Use the **LUCENE** operator to retrieve it.
  - **SPATIAL**, it uses Lucene to index the geo spatial coordinates.

- Any 3rd party index plugged

Dictionary

Every single database has a default manual index of type "DICTIONARY" called **dictionary** with Strings as keys. This is very useful to:

- handle root records of trees and graphs
- handle singleton records used for configuration

Operations against indexes

Create an index

Creates a new index. To create an automatic index bound to a schema property use section "ON" of create index command or use as name the `<class.property>` notation. But assure to have created the schema for it before the index. See the example below.

Syntax:

```
CREATE INDEX <name> [ON <class-name> (prop-names)] <type> [<key-type>]
                    [METADATA {<metadata>}]
```

Where:

- **name** logical name of index. Can be `<class>.<property>` to create an automatic index bound to a schema property. In this case **class** is the class of the schema and **property**, is the property created into the class. Notice that in another case index name can't contain '.' symbol
- **class-name** name of class that automatic index created for. Class with such name must already exist in database
- **prop-names** comma-separated list of properties for which automatic index is created for. Property with such name must already exist in schema. If property belongs to one of the Map types (LINKMAP, EMBEDDEDMAP) you can specify the keys or values used for index generation. Use "by key" or "by value" expressions for that, if nothing will be specified keys will be used during index creation.
- **type**, can be any index among the supported ones:
  - **unique**, uses the SB-Tree algorithm. Supports range queries.
  - **notunique**, uses the SB-Tree algorithm. Supports range queries.
  - **fulltext**, uses the SB-Tree algorithm. Supports range queries.
  - **dictionary**, uses the SB-Tree algorithm. Supports range queries.
  - **unique_hash_index**, uses the Hash algorithm. Doesn't supports range queries. Available since 1.5.x.
  - **notunique_hash_index**, uses the Hash algorithm. Doesn't supports range queries. Available since 1.5.x.
  - **fulltext_hash_index**, uses the Hash algorithm. Doesn't supports range queries. Available since 1.5.x.
  - **dictionary_hash_index**, uses the Hash algorithm. Doesn't supports range queries. Available since 1.5.x.

- **key-type**, is the type of key (Optional). On automatic indexes is auto-determined by reading the target schema property where the index is created. If not specified for manual indexes, at run-time during the first insertion the type will be auto determined by reading the type of the class.
- **metadata** is a json representing all the additional metadata as key/value

Examples of custom index:

```
CREATE INDEX mostRecentRecords unique date
```

Examples of automatic index bound to the property "id" of class "User":

```
CREATE PROPERTY User.id BINARY
CREATE INDEX User.id UNIQUE
```

Another index for "id" property of class "User":

```
CREATE INDEX indexForId ON User (id) unique
```

Examples of index for "thumbs" property of class "Movie".

```
CREATE INDEX thumbsAuthor ON Movie (thumbs) unique
CREATE INDEX thumbsAuthor ON Movie (thumbs by key) unique
CREATE INDEX thumbsValue ON Movie (thumbs by value) unique
```

Example of composite index

```
CREATE PROPERTY Book.author STRING
CREATE PROPERTY Book.title STRING
CREATE PROPERTY Book.publicationYears EMBEDDEDLIST INTEGER
CREATE INDEX books ON Book (author, title, publicationYears) unique
```

For more information look at Create index command.

Drop an index

Drop an index. Linked records will be not removed. Syntax:

```
DROP INDEX <name>
```

Where:

- **name** of the index to drop

For more information look at Drop index command.

Lookup

Returns all the records with the requested *key*.

```
select from index:<index-name> where key = <key>
```

Example:

```
select from index:dictionary where key = 'Luke'
```

Case insensitive match

To set a case-insensitive match in index, set the COLLATE attribute of indexed properties to "ci" (stands for Case Insensitive). Example:

```
create index OUser.name on OUser (name collate ci) UNIQUE
```

Put an entry

Inserts a new entry in the index with *key* and *rid*.

```
insert into index:<index-name> (key,rid) values (<key>,<rid>)
```

Example:

```
insert into index:dictionary (key,rid) values ('Luke',#10:4)
```

Query range

Retrieves the key ranges between *min* and *max*.

```
select from index:<index-name> where key between <min> and <max>
```

Example:

```
select from index:coordinates where key between 10.3 and 10.7
```

Remove entries by key

Deletes all the entries with the requested *key*.

```
delete from index:<index-name> where key = <key>
```

Example:

```
delete from index:addressbook where key = 'Luke'
```

Remove an entry

Deletes an entry by passing *key* and *rid*. Returns true if removed, otherwise false if the entry wasn't found.

```
delete from index:<index-name> where key = <key> and rid = <rid>
```

Example:

```
delete from index:dictionary where key = 'Luke' and rid = #10:4
```

Remove all references to a record

Removes all the entries with the *rid* passed.

```
delete from index:<index-name> where rid = <rid>
```

Example:

```
delete from index:dictionary where rid = #10:4
```

Count all the entries

Returns the number of entries on that index.

```
select count(*) as size from index:<index-name>
```

Example:

```
select count(*) as size from index:dictionary
```

Retrieve all the keys

Retrieves all the keys of the index.

```
select key from index:<index-name>
```

Example:

```
select key from index:dictionary
```

Retrieve all the entries

Retrieves all the entries of the index as pairs *key* and *rid*.

```
select key, value from index:<index-name>
```

Example:

```
select key, value from index:dictionary
```

Clear the index

Removes all the entries. The index will be empty after this call. This removes all the entries of an index.

```
delete from index:<index-name>
```

Example:

```
delete from index:dictionary
```

Null values

Indexes by default ignore null values. For such reason queries against NULL value that use indexes return no entries.

If you want to index also null values set `{ ignoreNullValues : false }` as metadata. Example:

```
CREATE INDEX addresses ON Employee (address) notunique
             METADATA {ignoreNullValues : false}
```

Composite keys

You can do the same operations with composite indexes.

A composite key is a collection of values by its nature, so syntactically it is defined as a collection. For example, if we have class book, indexed by its three fields:

- *author*,
- *title* and
- *publication year*

We can use following query to look up a book:

```
select from index:books where key = ["Donald Knuth", "The Art of Computer Programming", 1968
```

Or to look up a book within a *publication year* range:

```
select from index:books where key between ["Donald Knuth", "The Art of Computer Programming"
```

Partial match search

This is a mechanism that allows searching index record by several first fields of its composite key. In this case the remaining fields with undefined value can match any value in result.

Composite indexes are used for partial match search only when the declared fields in composite index are used from left to right. Using the example above, if you search only for *title*, the composite index cannot be used, but it will be used if you search for *author* + *title*.

For example, if we don't care when books have been published, we can throw away the *publication year* field from the query. So, the result of the following query will be all books with this *author* and *title* and any *publication year*

```
select from index:books where key = ["Author", "The Art of Computer Programming"]
```

If we also don't know *title*, we can keep only *author* field in query. Result of following query will be all books of this *author*.

```
select from index:books where key = ["Donald Knuth"]
```

Or equivalent

```
select from index:books where key = "Donald Knuth"
```

Range Queries

In case of range queries, the field subject to the range must be the last one. Example:

```
select from index:books
  where key between ["Donald Knuth", "The Art of Computer Programming", 1900]
    and ["Donald Knuth", "The Art of Computer Programming", 2014]
```

Direct insertion for composite indexes

Unsupported yet.

Custom keys

OrientDB since release 1.0 supports custom keys for indexes. This could give a huge improvement if you want to minimize memory used using your own serializer.

Below an example to handle SHA-256 data as binary keys without using a STRING to represent it saving disk space, cpu and memory.

Create 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;
  }
}
```

Create your own binary serializer

```
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 startP
    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;
  }
}
```

Register your serializer

```
OBinarySerializerFactory.INSTANCE.registerSerializer(new OHash256Serializer(), null);
index = database.getMetadata().getIndexManager().createIndex("custom-hash", "UNIQUE", new OR
```

Usage

```
ComparableBinary key1 = new ComparableBinary(new byte[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1
ODocument doc1 = new ODocument().field("k", "key1");
index.put(key1, doc1);
```

Tips and Tricks

Retrieve the list of indexes

Since OrientDB 1.6:

```
select expand(indexes) from metadata:indexmanager
```

Since OrientDB 1.0:

```
select expand(indexes) from cluster:0
```

Before OrientDB 1.0:

```
select expand(indexes) from #0:2
```

Create your index engine

Here a guide how to create a custom index engine.

SB-Tree index

This index is based on B-Tree index with several optimizations related to data insertion and range queries. As any other tree based indexes they have log(N) complexity, but base of this logarithm is about 500.

There is an issue about replacement of B-Tree based index by COLA Tree based index to avoid slowdown introduced by random I/O operations Issue #1756.

Hash Index

Hash index allows to perform index read operations for 1 (one) I/O operation, and index write for 3 (three) I/O operations as maximum. Hash index algorithm is based on extendible hashing Extendible Hashing algorithm. Hash index does not support range queries, but it's noticeable faster (about 2 times on 10M records) than SB-Tree index.

NOTE: There is an issue about enhancement of hash index to avoid slowdown introduced by random I/O operations using LSM Tree approaches: Issue #1757.

Full Text

Full Text indexes allow to index text as single word and its radix. Full text indexes are like a search engine on your database. If you are using the Lucene engine, please refer to Lucene Full-Text index.

Create a FullText Index

Example:

```
create index City.name on City (name) FULLTEXT
```

This will create a FullText index on the property name of the class City, with default configuration.

Configuration parameters of FullText Index:

Parameter	Default	Description
indexRadix	true	Word prefixes will be also index
ignoreChars	"	Chars to skip when indexing
separatorChars	\r\n\t:;,.|+*/\=!?[](.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

Configure a FullText Index (OrientDB v. 1.7)

To configure fulltext index use the metadata field.

Example with SQL:

```
create index City.name on City (name) FULLTEXT METADATA {"indexRadix" : true, "ignoreChars"
```

Example with 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 Full Text Index

Full text index can be created using the OrientDB SQL syntax as written here. Specify the index engine to use the lucene full text capabilities.

Syntax:

```
CREATE INDEX <name> ON <class-name> (prop-names) FULLTEXT ENGINE LUCENE
```

Example

```
create index City.name on City (name) FULLTEXT ENGINE LUCENE
```

Index can also be created on n-properties:

Example:

```
create index City.name_description on City (name,description) FULLTEXT ENGINE LUCENE
```

This will create a basic lucene index on the properties specified. If the analyzer is not specified, the default will be the StandardAnalyzer. To use a different analyzer use the field analyzer in the metadata JSON object in the CREATE INDEX syntax.

Example:

```
create index City.name on City (name) FULLTEXT ENGINE LUCENE METADATA {"analyzer":"org.apach
```

The Index can also be created with the Java Api. Example:

```
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"});
```

How to query a Full Text Index

The full text index can be queried using the custom operator `LUCENE` using the Query Parser Syntax of Lucene. Example:

```
select * from V where name LUCENE "test*"
```

will look for test, tests or tester etc..

Working with multiple field

To query multiple fields use this special syntax:

```
select * from Class where [prop1,prop2] LUCENE "query"
```

If query is a plain string the engine will parse the query using MultiFieldQueryParser on each indexed field.

To execute a more complex query on each fields surround your query with `()` parenthesis, to address specific field.

Example:

```
select * from Class where [prop1,prop2] LUCENE "(prop1:foo AND props2:bar)"
```

With this syntax the engine parse the query using the QueryParser.

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 City 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[]
```

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 [<x>,<y>]
```

To specify `maxDistance` we have to pass a special variable in the context:

```
select  from Class where [<lat-field>,<long-field>,$spatial] NEAR [<x>,<y>,{"maxDistance": d
```

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 [<x>,<y>,{"maxl
```

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,{"
```

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 [ [ <lng1>, <lat1> ] , [ <lng2>,
```

Examples

Example with previous configuration:

```
select * from Places where [latitude,longitude] WITHIN [[51.507222,-0.1275],[55.507222,-0.12
```

This query will return all Places within the given Bounding Box.

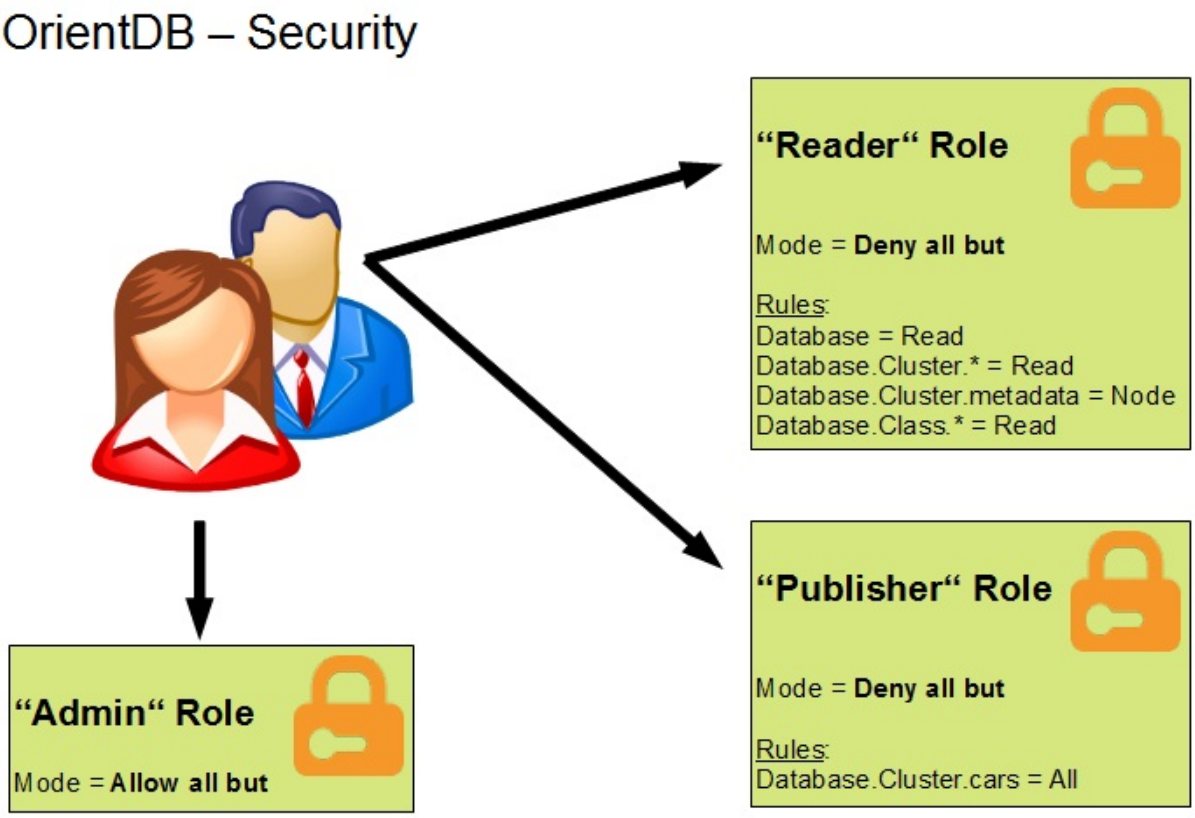
Future Plans

- Index All types of shape
- Adding more operators such as INTERSECT
- Extend the WITHIN operator to support not only Bounding Box

Security

The security model of OrientDB is based on well known concepts built on users and roles. A database has **"users"**. Each User has one or more **"roles"**. Role is compound by the mode of working (more later) and the set of permission rules.

Database security



Users

A User is an actor of the database. When you open a database you need to specify the user name and password used. Each user has own credentials and permissions.

By convention 3 users are always created by default every time you create a new database. Passwords are the same as the user name. Default users are:

- **admin**, with default password "admin", has access to all the functions without limitation
- **reader**, with default password "reader", is the classic read-only user. Can read any records but can't modify or delete them. Can't access to internal information such as user and role themselves.
- **writer**, with the default password "writer", is like the "reader" but can also create, update and delete records.

Users are themselves records stored inside the cluster "OUser". The passwords are stored in hash format using the strong algorithm SHA-256.

The user status is stored in the field "status" and can be: "SUSPENDED" and "ACTIVE". Only ACTIVE users can log in.

Work with users

To browse all the database's users use:

```
select from ouser
```

To create a new user use the SQL INSERT remembering to assign the status 'ACTIVE' and a valid role as in this example:

```
insert into ouser set name = 'jay', password = 'JaY', status = 'ACTIVE', roles = (select fro
```

To change the user name use:

```
update ouser set name = 'jay' where name = 'reader'
```

In the same way to change the user password use:

```
update ouser set password = 'hello' where name = 'reader'
```

The password will be saved in hash format using the algorithm SHA-256. The trigger "OUserTrigger" will encrypt the password transparently before the record is saved.

To disable a user change the status from 'ACTIVE' to 'SUSPENDED'. In this example we disable all the users but "admin":

```
update ouser set status= 'SUSPENDED' where name <> 'admin'
```

Roles

A role decides if it's allowed to execute an operation against a resource. Mainly this decision depends by the "working mode" and by the "rules". Rules work differently based on the "working mode".

Working with roles

Create a new role

To create a new role use the SQL INSERT remembering to assign the status 'ACTIVE' and a valid role as in this example:

```
insert into orole set name = 'developer', mode = 0
```

Inherited roles

Roles can inherit permissions from other roles in a Object Oriented fashion. To let a role extend another one add the parent role in the "inheritedRole" attribute. Example to let "appuser" role to inherit the "writer" role settings:

```
update orole set inheritedRole = (select from orole where name = 'writer') where name = 'app
```

Working modes

The supported "working modes" are:

1: allow all but (the rules)

By default is a super user and exceptions are enlisted in the rules. If no rule is found for the requested resource, then it's allowed to execute the operation. Use this mainly for power users. "Admin" default role uses this mode and has no exception rules. This mode is written as "1" in database.

0: deny all but (the rules)

By default it can't make nothing but the exceptions enlisted in the rules. This should be

the default mode for all classic users. "Reader" and "Writer" default roles use this mode. This mode is written as "0" in database.

Operations

The supported operations are the classic CRUD operations:

- (**C**)reate
- (**R**)ead
- (**U**)pdate
- (**D**)elete

A role can have none or all the permissions above. Each permission is internally represented by a flag of a 4 digit bitmask. So the above permissions are:

```
NONE:   #0000 - 0
CREATE: #0001 - 1
READ:   #0010 - 2
UPDATE: #0100 - 4
DELETE: #1000 - 8
ALL:    #1111 - 15
```

Of course you could make a combination of them. For example, if you want to allow only the Read and Update permissions, you could use

```
READ:               #0010 - 1
UPDATE:             #0100 - 4
Permission to use:  #0110 - 5
```

Resources

Resources are strings bound to OrientDB concepts. *Note: resources are case sensitive*:

- `database`
- `database.class`
- `database.class.<class-name>`
- `database.cluster`
- `database.cluster.<cluster-name>`
- `database.query`
- `database.command`
- `database.config`

- `database.hook.record`
- `server.admin`

Example:

Enable to the role "motorcyclist" the access to all the classes but the "Car" class:

```
update orole put rules = "database.class.*", 15 where name = "motorcyclist"
update orole put rules = "database.class.Car", 0 where name = "motorcyclist"
```

Note: resources are case sensitive

Grant and revoke permissions

To grant and revoke permissions use the SQLGrant and SQLRevoke commands.

Record level security

This is also called "horizontal security" because it doesn't act at schema level (vertically), but per each record. Due to this, we can totally separate the database records as sand-boxes where each "Restricted" records can't be accessed by non authorized users.

To activate this kind of advanced security, let the classes you want extend the **ORestricted** super class. If you're working with a Graph Database you should let V (Vertex) and E (Edge) classes extend ORestricted class:

```
alter class V superclass ORestricted
alter class E superclass ORestricted
```

In this way, all the vertices and edges will inherit the record level security.

Every time a class extends the **ORestricted** class, OrientDB, by a hook, injects a check before each CRUD operation:

- **CREATE** new document: set the current database's user in the `_allow` field. To change this behavior look at Customize on creation
- **READ** a document: check if the current user or its roles are enlisted in the `_allow` or `_allowRead` fields. If not the record is skipped. This let each queries to work per user basis
- **UPDATE** a document: check if the current user or its roles are enlisted in the `_allow` or `_allowUpdate` field. If not a OSecurityException is thrown
- **DELETE** a document: check if the current user or its roles are enlisted in the `_allow` or `_allowDelete` field. If not a OSecurityException is thrown

The "allow" fields (`_allow` , `_allowRead` , `_allowUpdate` , `_allowDelete`) can contain instances of **OUser** and **ORole** records (both classes extends OIdentity). Use **OUser** to allow single users and **ORole** to allow all the users that are part of these roles.

Customize on creation

By default everytime someone creates a Restricted record (when its class extends the ORestricted class) the current user is inserted in the " `_allow` " field. This can be changed by setting custom properties in the class schema supporting these properties:

- **onCreate.fields**, to specify the names of the fields to set. By default is " `_allow` " but

you can specify here " `_allowRead` ", " `_allowUpdate` " and " `_allowDelete` " or a combination of them. Use the comma to separate multiple fields
- **onCreate.identityType**, to specify if the user's object will be inserted or its role (the first one). By default is set "user", but you can also use "role"

Example to avoid the user can delete a new post:

```
orientdb> alter class Post custom onCreate.fields=_allowRead,_allowUpdate
```

Example to assign its role instead of user to the new Post instances created:

```
orientdb> alter class Post custom onCreate.identityType=role
```

Bypass security constraints

Sometimes you need to create a role that can bypass such restrictions, such as backup or administrative operations. For such reason we've created the special permission `database.bypassRestricted` to READ. By default, the "admin" role has such permission.

This permission is not inheritable, so if you need to give such high privilege to other roles set it to each role.

Use case

You want to enable this security in a BLOG like application. First create the document class, like "Post" that extends "ORestricted". Then if the user "Luke" creates a new post and the user "Steve" does the same, each user can't access the Post instances created by each other.

```
orientdb> connect remote:localhost/blog admin admin
orientdb> create class Post extends ORestricted
Class 'Post' created successfully
```

The user "Luke", registered as OUser "luke" having RID #5:5, logs in and create a new Post:

```
orientdb> connect remote:localhost/blog luke luke
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    |
+-----+--------------+-----------------------+
```

Then the user Steve, registered as OUser "steve" having RID #5:6, logs in too and create a new 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       |
+-----+--------------+-----------------------+
```

Each user can see only the record where they have access. Now try to allow the user Steve (rid #5:6) to access to the first Luke's post adding the Steve's RID in the `_allow` field:

```
orientdb> connect remote:localhost/blog luke luke
orientdb> update #18:0 add _allow = #5:6
```

Now if Steve executes the same query as before, the result changes:

```
orientdb> connect remote:localhost/blog steve steve
orientdb> select from Post
+-----+--------------+-----------------------+
| RID | _allow       | title                 |
+-----+--------------+-----------------------+
|#18:0| [#5:5]       | Yesterday in Italy    |
|#18:1| [#5:6]       | My Nutella cake       |
+-----+--------------+-----------------------+
```

Now we would like to let Steve only read posts by Luke, without the rights to modify them. So we're going to remove Steve from the generic "_allow" field to insert into the "_allowRead":

```
orientdb> connect remote:localhost/blog luke luke
orientdb> update #18:0 remove _allow = #5:6
orientdb> update #18:0 add _allowRead = #5:6
```

Now if Steve connects and displays all the Post instances he will continue to display the Luke's post but can't update or delete them.

```
orientdb> connect remote:localhost/blog steve steve
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
```

You can enable this feature even on graphs. Follow this tutorial to look how to create a partitioned graph.

OrientDB Server security

A single OrientDB server can manage several databases per time, each one with its users. In HTTP protocol is handled by using different realms. This is the reason why OrientDB Server instance has its own users to handle the server instance itself.

When the OrientDB Server starts check if there is configured the "root" user. If not creates it into the config/orientdb-server-config.xml file with an automatic generated very long password. Feel free to change the password, but restart the server to get the changes.

It is in a section that should look like this:

```
<users> <user name="root" password="FAFF343DD54DKFJFKDA95F05A" resources="*" /> </users>
```

Since the passwords are in clear, who is installing OrientDB have to protect the entire directory (not only config folder) to avoid any access to the not authorized users.

Server's resources

This section contains all the available server's resources. Each user can declare which resources have access. Wildcard * means any resources. *root* server user, by default, has all the privileges, so it can access to all the managed databases.

Resource	Description
server.info	Retrieves the server information and statistics
server.listDatabases	Lists the available databases on the server
database.create	Creates a new database in the server
database.drop	Drops a database
database.passthrough	Starting from 1.0rc7 the server's user can access to all the managed databases if has the resource **database.passthrough** defined. Example: `<user name="replicator" password="repl" resources="database.passthrough" />`

SSL Secure connections

Starting from v1.7 OrientDB support secure SSL connections.

Restore admin user

If the class OUser has been dropped or the "admin" user has been deleted, you can follow this procedure to restore your database:

1) Assure the database is under the OrientDB Server's databases directory ($ORIENTDB_HOME/databases/ folder)

2) Open the Console or Studio and login into the database using "root" and the password contained in file $ORIENTDB_HOME/config/orientdb-server-config.xml

3) Execute this query:

```
select from OUser where name = 'admin'
```

4) If the class OUser doesn't exist, create it by executing:

```
create class OUser extends OIdentity
```

5) If the class OIdentity doesn't exist, create it by executing:

```
create class OIdentity
```

And then retry to create the class OUser (5)

6) Now execute:

```
select from ORole where name = 'admin'
```

7) If the class ORole doesn't exist, create it by executing:

```
create class ORole extends OIdentity
```

8) If the role "admin" doesn't exist, create it by executing the following command:

```
insert into ORole set name = 'admin', mode = 1, rules = {"database.bypassrestricted":15}
```

9) If the user "admin" doesn't exist, create it by executing the following command:

```
insert into OUser set name = 'admin', password = 'admin', status = 'ACTIVE',
                      roles = (select from ORole where name = 'admin')
```

Now your "admin" user is active again.

SSL

Starting from v1.7, OrientDB provides the ability to secure is HTTP and BINARY protocols using SSL (For Distributed SSL see the HazelCast Documentation).

Setting up the Key and Trust Stores

OrientDB uses the JAVA Keytool to setup and manage certificates. This tutorial shows how to create key and trust stores that reference a self signed cert. Use of CA signed certs is outside the scope of this document. For more details on using the java Keytool please visit http://docs.oracle.com/javase/7/docs/technotes/tools/index.html#security and for more information.

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's certificate so it can be shared with clients:

   ```
   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, and import the server's certificate. This establishes that the client "trusts" the server:

   ```
   keytool -import -alias server -keystore orientdb-console.ts -file orientdb.cert
   ```

NOTE: You will need to repeat steps 3 and 4 for each remote client vm you wish to connect to the server. Remember to change the alias and keystore and trust-store filenames accordingly.

Configuring the Server

The OrientDB server config ($ORIENTDB_HOME/config/orientdb-server-config.xml) does not enable SSL by default. To enable SSL on a protocol listener you simply change the "socket" attribute of the from "default" to one of your configured definitions.

There are two default definitions named "ssl" and "https". These should be sufficient for most uses cases, however more can be defined if you want to secure different listeners with there own certificates or want custom socket factory implementations. When using the "ssl" implementation keep in mind that the default port for OrientDB SSL is 2434 and that your port-range should be changed to 2434-2440.

By default, the OrientDB Server looks for its keys and trust stores in $ORIENTDB_HOME/config/cert. This is configured using the parameters. Make sure that all of the key and trust stores created in the previous setup are in the correct directory and that the passwords used are also correct.

Note that paths are relative to $ORIENTDB_HOME. Absolute paths are supported.

Example Configuration

```
<sockets>
  <socket implementation="com.orientechnologies.orient.server.network.OServerSSLSocketFact
    <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 sec

      <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"/>
```

Configuring the Console

To enable SSL for remote connections using the console, a few changes to the console script are required.

1. Confirm that your KEYSTORE, TRUSTSTORE and repective PASSWORD variables are set correctly.
2. In the SSL_OPTS definition set the "client.ssl.enabled" system property to "true"

Configuring Client

Configuring remote clients can be done using standard java system property patterns.

Properties:

- `client.ssl.enabled` : Used to enable/disable SSL. Accepts "true" or "false". Only needed when using remote binary client connections.
- `javax.net.ssl.keyStore` : Path to key store
- `javax.net.ssl.keyStorePassword` : Key store password
- `javax.net.ssl.trustStore` : Path to trust store
- `javax.net.ssl.trustStorePassword` : Trust store password

Use steps 3 and 4 from the "Setting up the Key and Trust Stores" section to create client certificates and trust of the server. Paths to the stores will be client specific and do not need to be the same as the server.

If you would like to use key and/or truststores other that the default JVM they should use instead:

- `client.ssl.keyStore` : Path to key store
- `client.ssl.keyStorePass` : Key store password
- `client.ssl.trustStore` : Path to trust store
- `client.ssl.trustStorePass` : Trust store password

Example Java Command Line:

```
java -Dclient.ssl.enabled=false -Djavax.net.ssl.keyStore=</path/to/keystore> -Djavax.net.ssl
    -Djavax.net.ssl.trustStore=</path/to/truststore> -Djavax.net.ssl.trustStorePassword=<tru
```

Example Java Implementation:

```
System.setProperty("client.ssl.enabled", <"true"|"false">); # This will only be needed for r
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>);
```

If you want to verify/authenticate client certificates, you need to take a few extra steps on the server:

1. Export the client's certificate so it can be shared with server:

   ```
   keytool -export -alias <client_alias> -keystore <client.ks> -file client_cert
   ```

   Example using console:

   ```
   keytool -export -alias console -keystore orientdb-console.ks -file orientdb-console.cert
   ```

2. Create a truststore for the server if one does not exist, and import the client's certificate. This establishes that the server "trusts" the client:

   ```
   keytool -import -alias <client_alias> -keystore orientdb.ts -file client_cert
   ```

   Example using console:

   ```
   keytool -import -alias console -keystore orientdb.ts -file orientdb-console.cert
   ```

In the server config make sure that client authentication is enabled for the and that the trust-store path and password are correct:

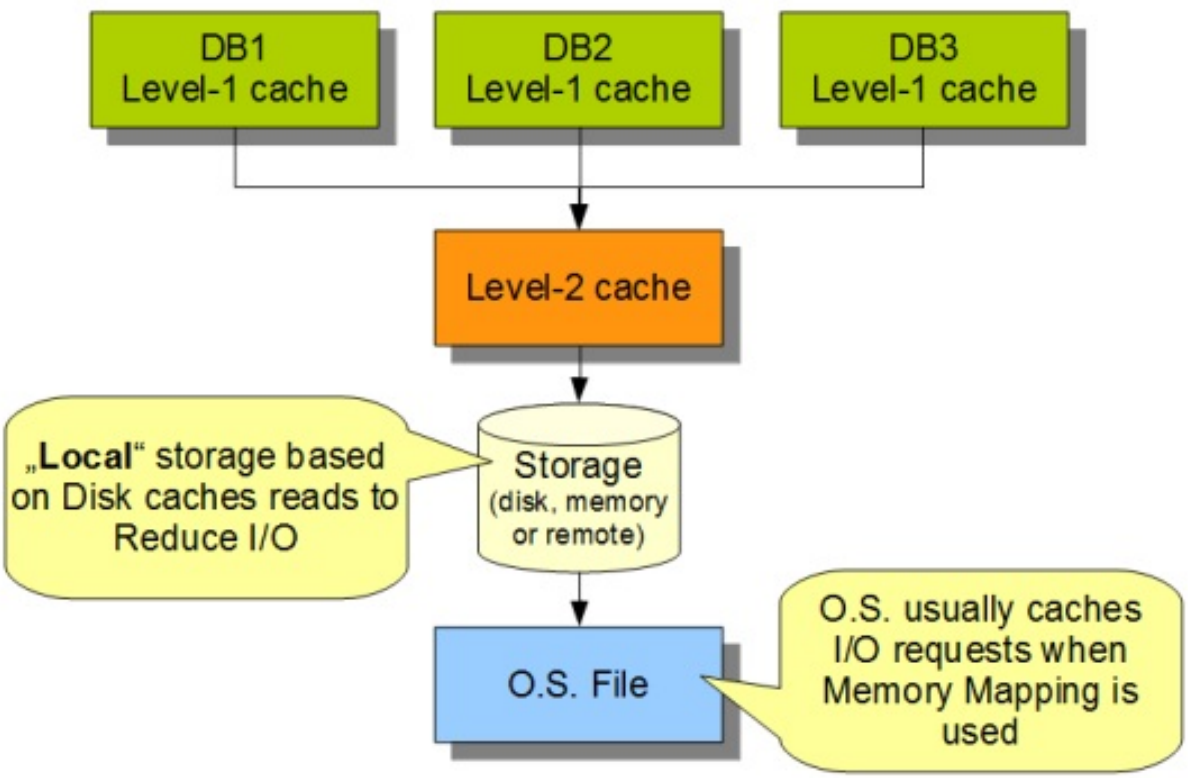
Example

```
<sockets>
  <socket implementation="com.orientechnologies.orient.server.network.OServerSSLSocketFact
    <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 impor
      <parameter value="config/cert/orientdb.ts" name="network.ssl.trustStore"/>
      <parameter value="password" name="network.ssl.trustStorePassword"/>
    </parameters>
  </socket>
```

Caching

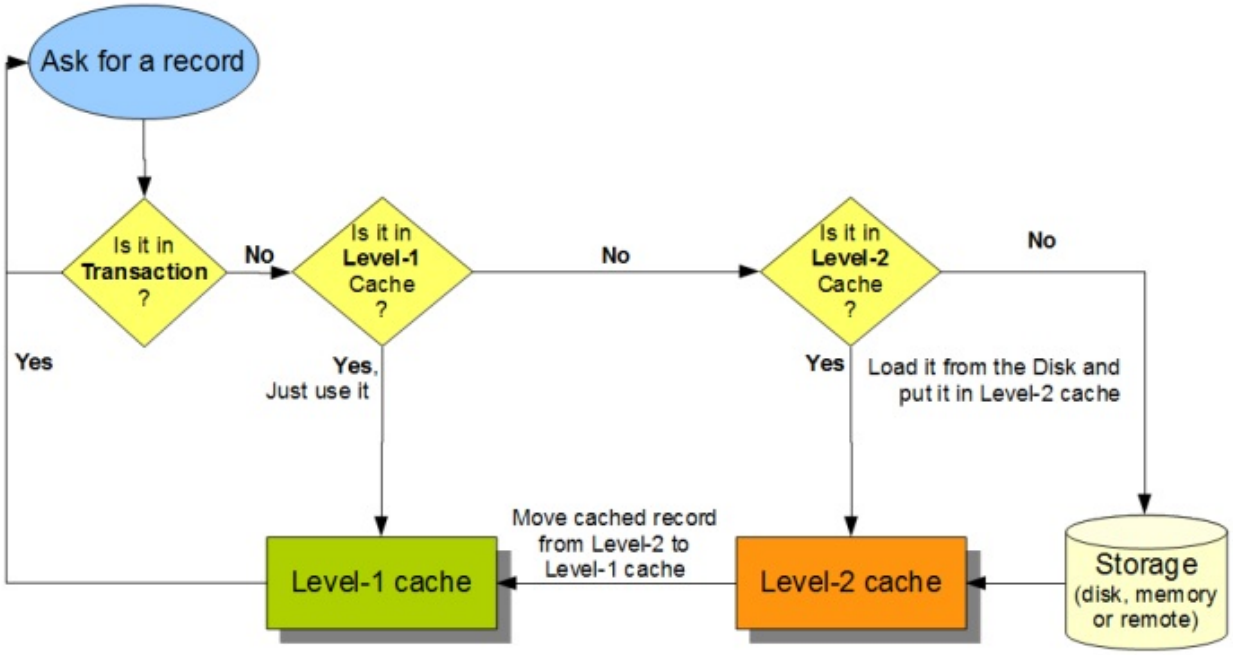
OrientDB has several caching mechanisms that act at different levels. Look at this picture:



- **Local cache** is one per database instance (and per thread in multi-thread environment)
- **Storage**, depending by the implementation could cache. This is the case for the **Local Storage** (disk based) that caches file reads to reduce I/O requests

How cache works?

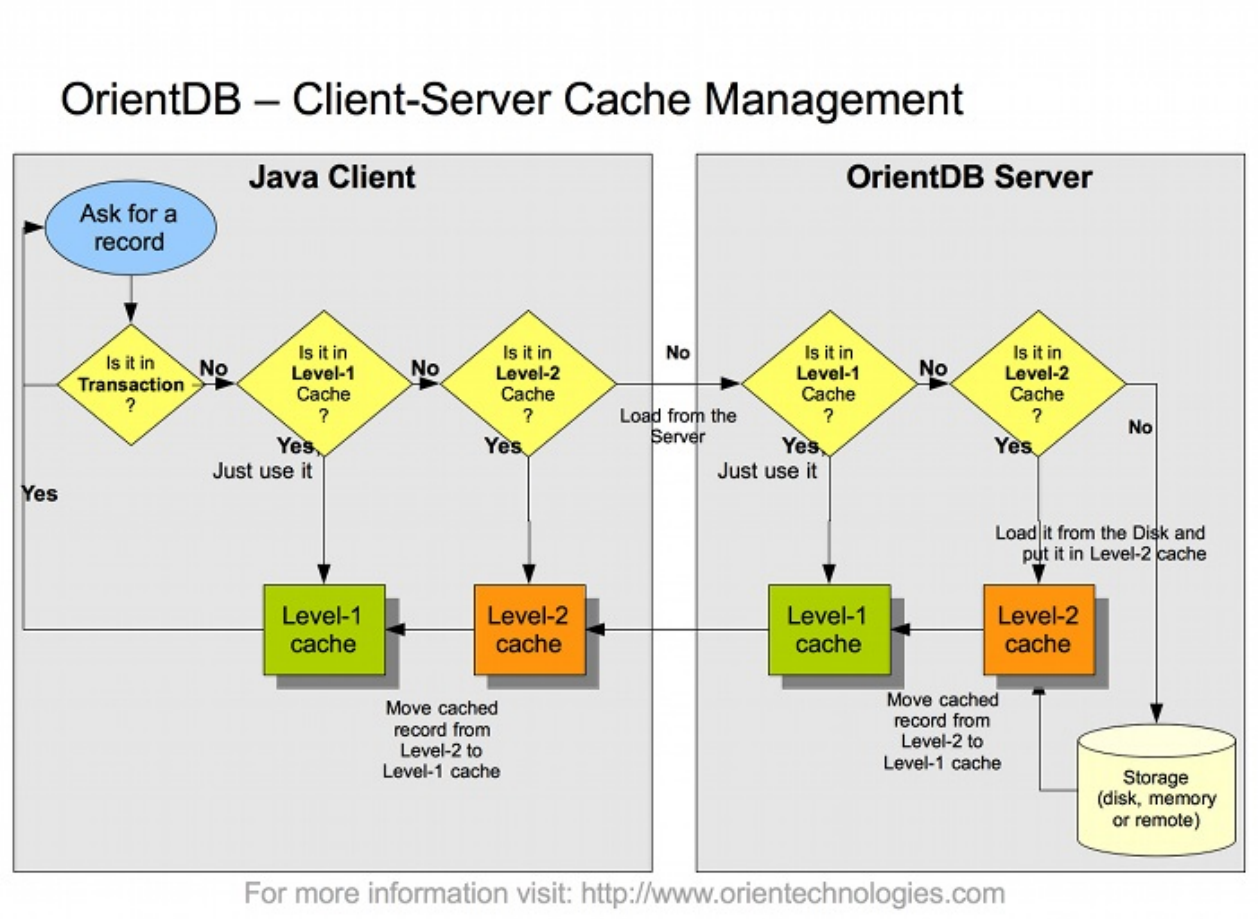
Local Mode (embedded database)



When the client application asks for a record OrientDB checks:

- if a **transaction** is begun searches it inside the transaction changed records and returns it if found
- if the **Local cache** cache is enabled and contains the requested record then return it
- at this point the record is not in cache, then ask it to the **Storage** (disk, memory)

Client-Server Mode (remote database)



When the client application asks for a record OrientDB checks:

- if a **transaction** is begun searches it inside the transaction changed records and returns it if found
- if the **Local cache** cache is enabled and contains the requested record then return it
- at this point the record is not in cache, then ask it to the **Server** through a TCP/IP call
- in the server if the **Local cache** cache is enabled and contains the requested record then return it
- at this point the record is not in cache in the server too, then ask 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 2 conditions will be satisfied:

1. There are no links to these records from outside of database
2. The Java Virtual Machine doesn't have enough memory to allocate for 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);
```

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:

- 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

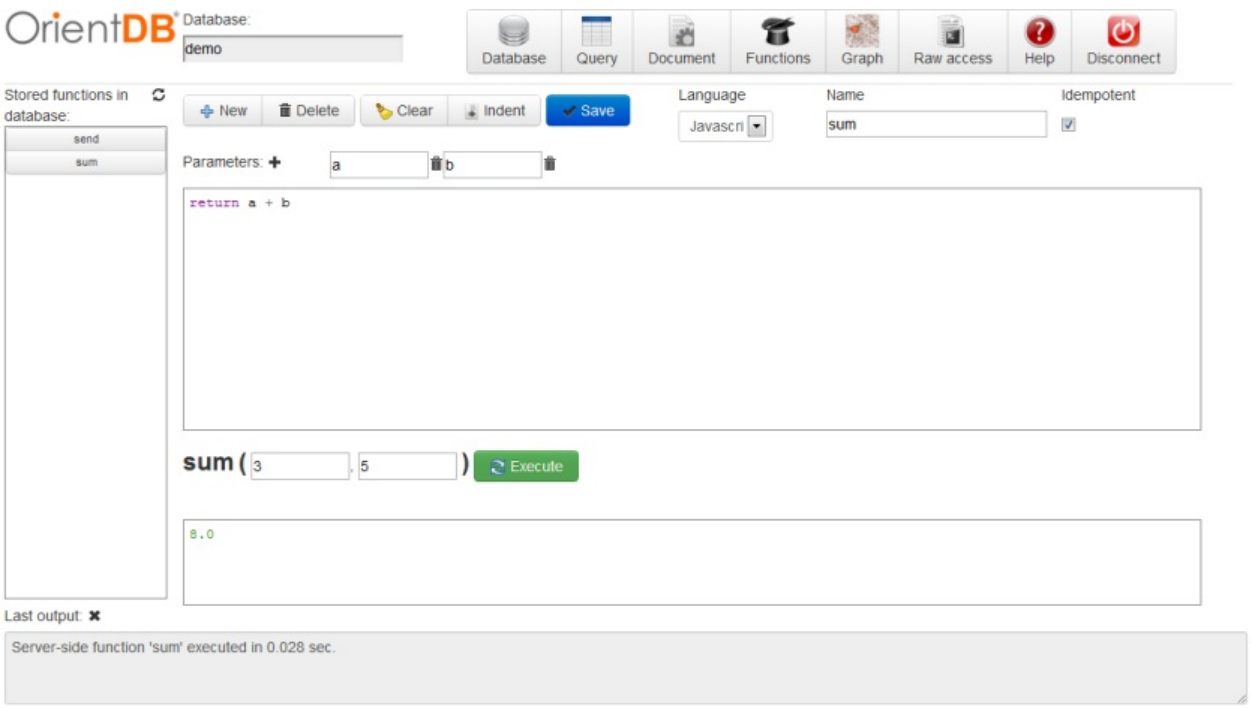
Create your first function

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 a + 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.



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 are passed by position.

Below how to execute the "sum" function created before:

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

This 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 Techno
```

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");
```

}

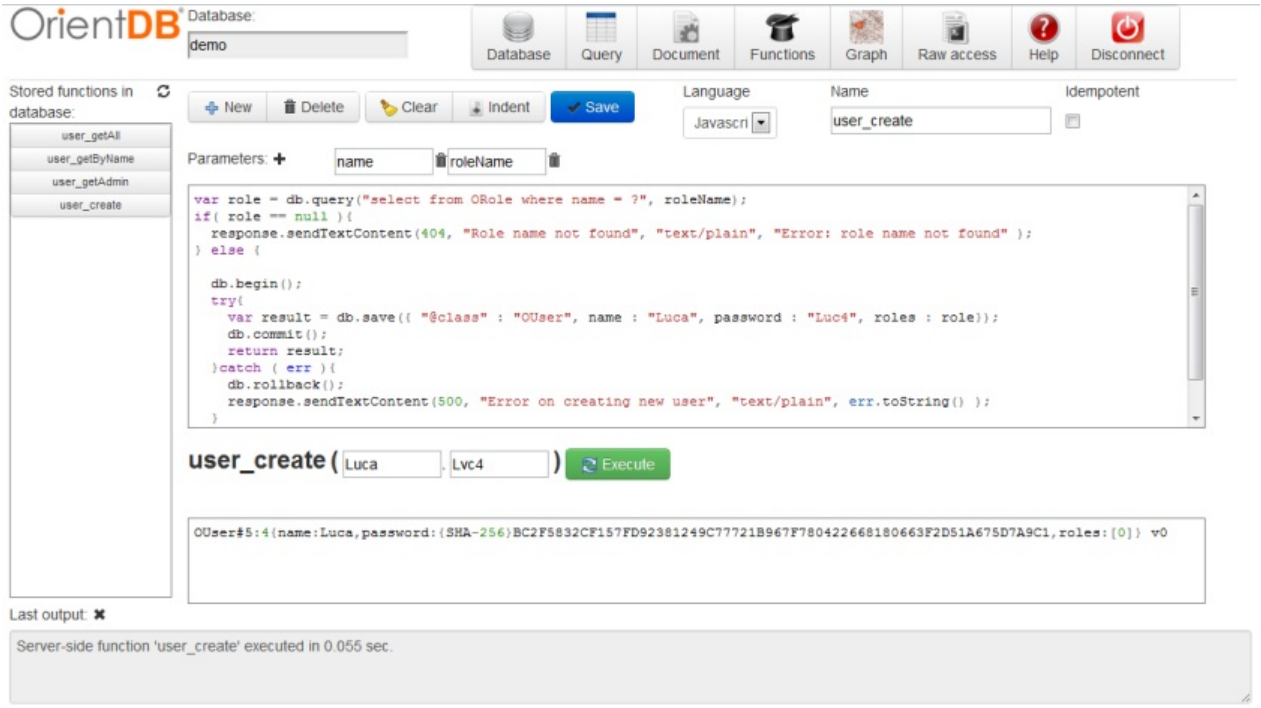
function user_create(name, role){

```
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", roles : rol
      db.commit();
      return result;
    }catch ( err ){
      db.rollback();
      response.send(500, "Error on creating new user", "text/plain", err.toString() );
    }
  }
```

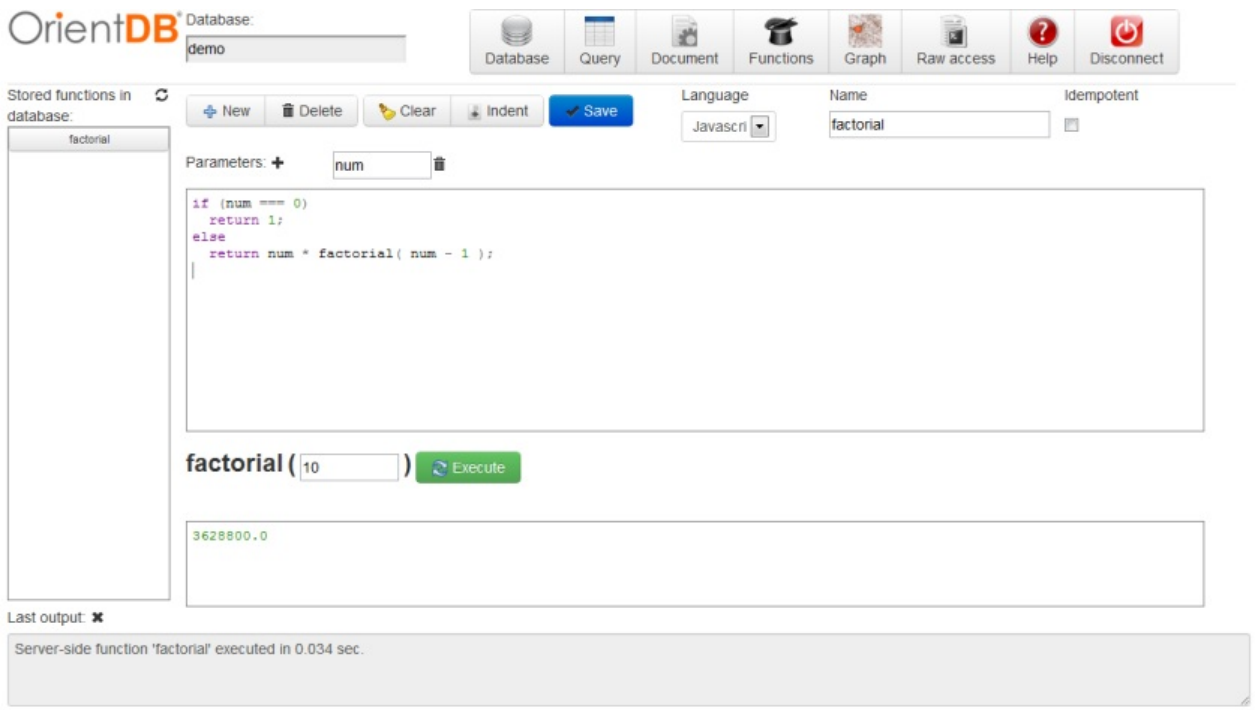
}



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 );
```



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
	Returns the number of parameters	

	found between those passed	
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" : r
    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

Transactions

A transaction comprises a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. Transactions in a database environment have two main purposes:

- to provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status
- to provide isolation between programs accessing a database concurrently. If this isolation is not provided, the program's outcome are possibly erroneous.

A database transaction, by definition, must be atomic, consistent, isolated and durable. Database practitioners often refer to these properties of database transactions using the acronym ACID. --- Wikipedia

OrientDB is an ACID compliant DBMS.

NOTE: OrientDB keeps the transaction on client RAM, so the transaction size is affected by the available RAM (Heap memory) on JVM. For transactions involving many records, consider to split it in multiple transactions.

ACID properties

Atomicity

"Atomicity requires that each transaction is 'all or nothing': if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes. To the outside world, a committed transaction appears (by its effects on the database) to be indivisible ("atomic"), and an aborted transaction does not happen." - WikiPedia

Consistency

"The consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including but not limited to constraints, cascades, triggers, and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors do not violate any defined rules." - WikiPedia

OrientDB uses the MVCC to assure consistency. The difference between the management of MVCC on transactional and not-transactional cases is that with transactional, the exception rollbacks the entire transaction before to be caught by the application.

Look at this example:

Sequence	Client/Thread 1	Client/Thread 2	Version of record X
1	Begin of Transaction		
2	read(x)		10
3		Begin of Transaction	
4		read(x)	10
5		write(x)	10
6		commit	10 -> 11
7	write(x)		10

8	commit		10 -> 11 = Error, in database x already is at 11

Isolation

"The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e. one after the other. Providing isolation is the main goal of concurrency control. Depending on concurrency control method, the effects of an incomplete transaction might not even be visible to another transaction." - WikiPedia

OrientDB has different levels of isolation based on settings and configuration:

- **READ COMMITTED**, the default and the only one available with `remote` protocol
- **REPEATABLE READS**, allowed only with `plocal` and `memory` protocols. This mode consumes more memory than **READ COMMITTED**, because any read, query, etc. keep the records in memory to assure the same copy on further access

To change default Isolation Level, use the Java API:

```
db.begin()
db.getTransaction().setIsolationLevel(OTransaction.ISOLATION_LEVEL.REPEATABLE_READ);
```

Using `remote` access all the commands are executed on the server, so out of transaction scope. Look below for more information.

Look at this examples:

Sequence	Client/Thread 1	Client/Thread 2
1	Begin of Transaction	
2	read(x)	
3		Begin of Transaction
4		read(x)
5		write(x)
6		commit
7	read(x)	
8	commit	

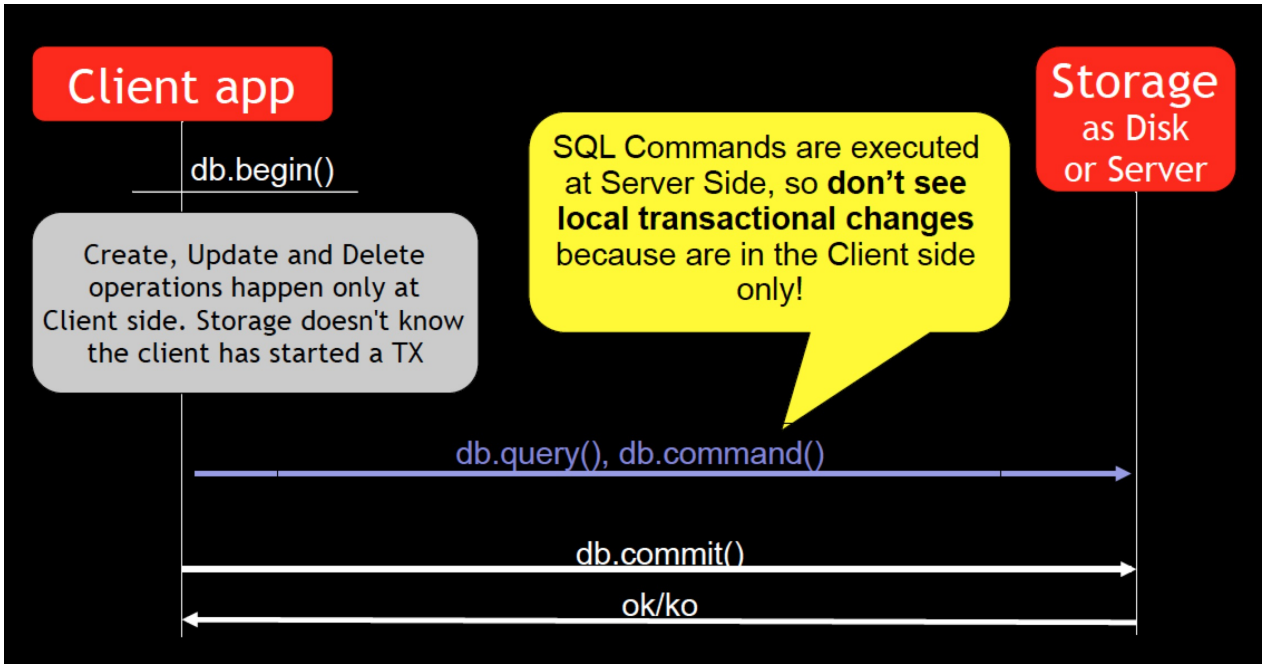
At operation 7 the client 1 continues to read the same version of x read in operation 2.

Sequence	Client/Thread 1	Client/Thread 2
1	Begin of Transaction	
2	read(x)	
3		Begin of Transaction
4		read(y)
5		write(y)
6		commit
7	read(y)	
8	commit	

At operation 7 the client 1 reads the version of y which was written at operation 6 by client 2. This is because it never reads y before.

Breaking of ACID properties when using remote protocol and Commands (SQL, Gremlin, JS, etc)

Transactions are client-side only until the commit. This means that if you're using the "remote" protocol the server can't see local changes



In this scenario you can have different isolation levels with commands.

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. 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.

Transaction-propagation

Hooks (Triggers)

Hook works like a trigger. Hook lets to the user application to intercept internal events before and after each CRUD operation against records. You can use to write custom validation rules, to enforce security or even to orchestrate external events like the replication against a Relational DBMS.

OrientDB supports two main kinds of Hooks:

- Dynamic Hooks, defined at schema and/or document level
- Native Java Hooks, defined as Java classes

What use? Pros/Cons?

Depends by your goal: Java Hooks are the fastest hooks. 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 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 rely to a class. Below 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 print to console the invoice number created.

```
CREATE FUNCTION invoiceCreated "print('\\nInvoice created: ' + doc.field('number'));" LANGUA
```

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 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
```

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:preven
```

(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 seperate 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 event from 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 to the document of 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 to the dirty fields and the original values. Example:

```
for( String field : document.getDirtyFields() ) {
  Object originalValue = document.getOriginalValue( field );
  ...
}
```

Self registration

Hooks could be installed only to certain database instances, but in most of the cases you'd 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 to be executed in the Server space you've 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;
    }
  }
}
...
```

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

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
	Oriento	Native	Binary protocol, new branch that has been updated with the latest functionality. Tested on 1.7.0. Branched from node-orientdb
	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		

	.NET driver for OrientDB	Binary	**Official Driver**
	PyOrient	Binary	**Official 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 interface. Use this driver if you're working against graphs.
	OrientDB-JRuby	Native	Through Java driver
	OrientDB Client	Binary	
	OrientDB4R	HTTP	
	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
	Clojure binding	Native	Through Java driver
	Clojure binding of Blueprints API		
			OrientDB-Android is a

	OrientDB Android	Porting	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 latest version

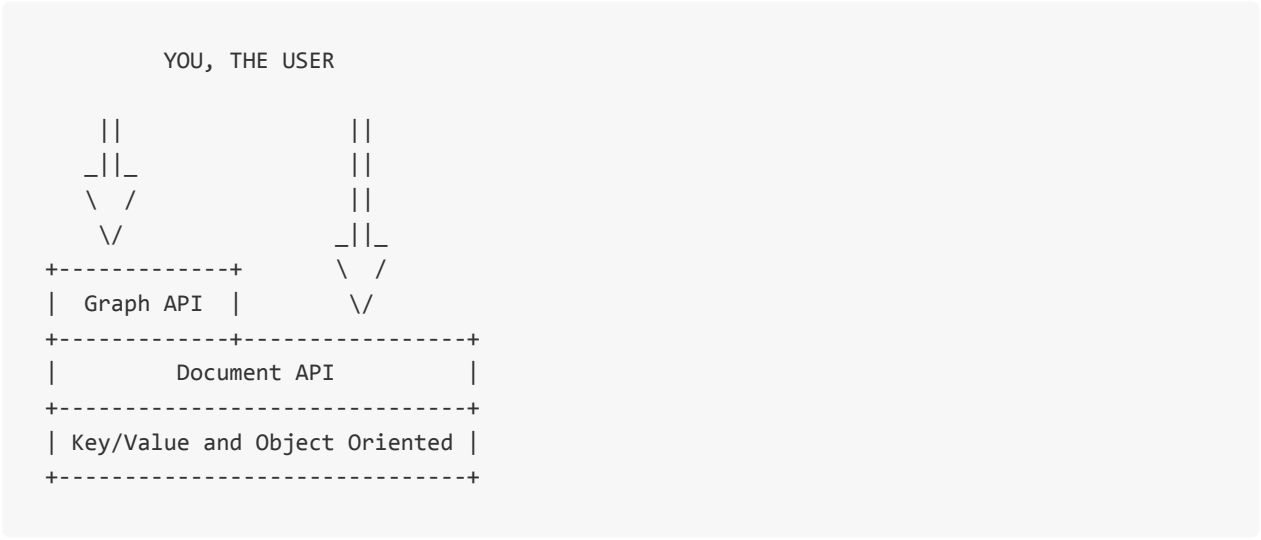
JDO

- JDO 2.2 and JPA 2 by using the Data Nucleus adapter: datanucleus

All the trademarks are property of their legal owners.

Graph or Document API?

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



Graph API

With OrientDB 2.0, we improved our Graph API to support all models in just one Multi-Model API. This API usually covers 80% of use cases, so this could be the default API you should use if you're starting with OrientDB.

In this way:

- Your Data ('records' in the RDBMS world) is modeled as Vertices and Edges. You can store properties on both.
- You can still work in Schema-Less, Schema-Full or Hybrid modes.
- Relationships are modeled as Bidirectional Edges. If 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 to have 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 look at Lightweight Edges.

Document API

What about the remaining 20%? In the case where you need a Document Database (keeping 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:

- 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 everything is connected as In & Out. For this reason, Graph operations must be done within Transactions. Instead, when you create a relationship between documents with a LINK, the target linked document is not involved in this operation. This results in better Multi-Thread support, especially with insert, deletes and updates operations.

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.

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: 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
- 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, $
```

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
@type	returns the record type between: 'document', 'column', 'flat', 'bytes'	**@type** = 'flat'	0.9.21

Operators

Conditional Operators

Apply to	Operator	Description
any	=	Equals to
string	like	Similar to equals, but allow the wildcard '%' that means 'any'
any	<	Less than
any	<=	Less than or equal to
any	>	Greater than
any	>=	Greater than or equal to
any	<>	Not equals (same of !=)
any	BETWEEN	The value is between a range. It's equivalent to <field> >= <from-value> AND <field> <= <to-value>
any	IS	Used to test if a value is NULL
record, string (as class name)	INSTANCEOF	Used to check if the record extends a class
collection	IN	contains any of the elements listed
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
collection	CONTAINSALL	true if all the elements of the collection satisfy the next condition
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
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
		used with 89cd72a14eb5493801e99a43c5034685. Current limitation is that it must be the unique condition of a query. When used

string	CONTAINSTEXT	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
string	MATCHES	Matches the string using a [http://www.regular-expressions.info/ Regular Expression]
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.

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** name = 'Luke'	Not supported yet

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

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 receive 1 or more parameters:

Aggregated mode

When only one parameter is passed. They aggregate the result in only one record. The classic example is the sum():

```
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 as inline when you've only one parameter, then add a second one like "null":

```
SELECT first( out('friends').name, null ) as firstFriend FROM Profiles
```

In this case `first()` function doesn't aggregate everything in only one record, but returns one record per `Profile` where the `firstFriend` is the first item of the collection received as parameter.

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()		first()	ifnull()
bothE()		intersect()	coalescence()
outV()	avg()	distinct()	uuid()
inV()	count()	expand()	if()
traversedElement()	mode()	union()	
traversedVertex()	median()	flatten()	
traversedEdge()	percentile()	last()	
shortestPath()	variance()		
dijkstra()	stddev()		

Functions by name

avg()	both()	bothE()	coalescence()
count()	date()	difference()	dijkstra()
distance()	distinct()	eval()	expand()
format()	first()	flatten()	if()

ifnull() Iin() I inE() I inV() I I intersect() Ilist() I map() I min() I I max() I median() I mode() I out() I I outE() I outV() I percentile() I set() I I shortestPath() Istddev()Isum()Isysdate()I I traversedElement() I traversedEdge() I traversedVertex() I union() I I uuid()I variance() I

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>)`

Available since: 1.3.0

Example

```
SELECT coalesce(amount, amount2, amount3) from Account
```

if()

Syntax: `if(<expression>, <result-if-true>, <result-if-false>)`

Evaluates a condition (first parameters) and returns the second parameter if the condition is true, the third one otherwise

Example:

```
select if(eval("name = 'John'"), "My name is John", "My name is not John") from Person
```

ifnull()

Returns the passed field/value (or optional parameter return_value_if_not_null). If field/value is not null, otherwise it returns return_value_if_null.

Syntax: `ifnull(<field|value>, <return_value_if_null>[,<return_value_if_not_null>](,<field&.md#124;value>]*)`

Available since: 1.3.0

Example

```
SELECT ifnull(salary, 0) from Account
```

expand()

Expands the collection in the field and use it as result.

Available since: 1.4.0

Syntax: `expand(<field>)`

Example

```
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
```

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 average(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.

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.

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.

Syntax: `shortestPath( <sourceVertex>, <destinationVertex> [, <direction>])`

Available since: 1.3.0

Example

```
select shortestPath(#8:32, #8:10, 'BOTH')
```

---

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
```

---

union()

Syntax: `union(<field> [,<field-n>]*)`

Works as aggregate or inline. If only one argument is passed than aggregates, otherwise executes, and returns, a UNION of the collections received as parameters. Works also with no collection values.

Available since: 1.0rc2

Example

```
select union(friends) from profile
```

```
select union(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

```sql
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 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()
asSet()	asString()	charAt()	convert()	exclude()
hash()	include()	indexOf()	javaType()	keys()
length()	normalize()	prefix()	remove()	removeAll()
right()	size()	subString()	trim()	toJSON()
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.

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.

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

```
selct 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

```
selct 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:

```
selct 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

```
selct tags.asList() from Friend
```

History

- 1.0rc2: First version

.asLong()

Transforms the field into a Long type.

Syntax: `<value>.asLong()`

Applies to the following types:

- any

Examples

```
selct 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

```
selct 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

```
selct 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.

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. 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

```

```

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`
- `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 $
- `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
```

Just plain OrientDB SQL, but with a few new items:

- `begin` -> begins a transaction
- `rollback` -> rollbacks an active transaction
- `commit [retry <times>]` -> commits an active transaction
- `let <variable> = <command>` -> executes a command and assigns it in the context as . That variable can be used in further commands by prefixing it with $
- `return <$variable>|<value>|null` -> returns a value instead of last command result (default)

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)

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 =se
    }
  ]
}
```

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
ORID last = new ORecordId();

List<ODocument> resultset = database.query(query, last);

while (!resultset.isEmpty()) {
    last = resultset.get(resultset.size() - 1).getIdentity();
    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 
for (List<ODocument> resultset = database.query(query); !resultset.isEmpty(); resultset = da
    ...
}
```

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", tr
```

Sequences and auto-increment

OrientDB doesn't support serial (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 WHERE name = 'mycounter' return after
INSERT INTO items SET id = $counter.value, qty = 10, price = 1000
COMMIT
```

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		Alter Property		Create Database (console only)	Revoke
Truncate Class		Drop Property		Drop Database (console only)	Create function
Truncate Cluster					
Truncate Record					

SQL - SELECT

Orient supports the SQL language to execute queries against the databas engine. Take a look at the operators and Functions. To learn the main differences in comparison to the SQL-92 standard, take a look at: OrientDB SQL.

Syntax

```
SELECT [<Projections>] [FROM <Target> [LET <Assignment>*]]
    [WHERE <Condition>*]
    [GROUP BY <Field>*]
    [ORDER BY <Fields>* [ASC|DESC] *]
    [SKIP <SkipRecords>]
    [LIMIT <MaxRecords>]
    [FETCHPLAN <FetchPlan>]
    [TIMEOUT <Timeout> [<STRATEGY>]
    [LOCK default|record]
    [PARALLEL]
```

- **Projections**, optionally, is the data you want to extract from the query as the result set. Look at Projections. Available since 0.9.25.
- **Target** can be a class, cluster, single RID, set of RIDs or index values sorted by ascending or descending key order (index values were added in 1.7.7). **Class** is the class name on which to execute the query. Similarly, specifying **cluster** with the `cluster:` prefix executes the query within that cluster only. You can fetch records not from a cluster but instead from an index using the following prefixes: `indexvalues:`, `indexvaluesasc:` or `indexvaluesdesc:`. If you are using `indexvalues:` or `indexvaluesasc:` prefix records will be sorted in ascending order of index keys. If you are using `indexvaluesdesc:` prefix records will be sorted in descending order of index keys. Use one or more RIDs to specify one or a small set of records. This is a useful in order to specify a starting point when navigating graphs.
- **WHERE** condition is common to the other SQL commands and is described in a dedicated section of the documentation.
- **LET** is the part that binds context variables to be used in projections, conditions or sub-queries
- **GROUP BY** is in accordance to the standard SQL syntax specifying the field to perform the grouping. The current release supports only 1 field.
- **ORDER BY** is in accordance to the standard SQL syntax specifying fields with an optional ASC or DESC (default is ASCending). If you are using a projection in your query, ensure the ORDER BY field is included in this projection.
- **SKIP** skips `<SkipRecords>` the specified number of records starting at the beginning of the result set. This is useful for Pagination when used in conjunction with `LIMIT`.
- **LIMIT** sets the maximum number of records returned by the query to `<MaxRecords>`. This is useful for Pagination when used in conjunction with SKIP.
- **FETCHPLAN** sets the fetchplan. Example: `FETCHPLAN out:3` to pre-fetch up to 3rd level under `out` field. Since v1.5.
- **TIMEOUT** sets the maximum timeout in milliseconds for the query. By default the

query has no timeout. If you don't specify the strategy, the default strategy `EXCEPTION` is used. Strategies are:
    - `RETURN`, truncate the result set returning the data collected up until the timeout
    - `EXCEPTION`, default one, throws an exception if the timeout has been reached

- **LOCK** manage the locking strategy. By default is "default", that means release the lock once the record is read, while "record" means to keep the record locked in exclusive mode in current transaction till the transaction has been finished by a commit or rollback operation.
- **PARALLEL** execute the query against X concurrent threads, where X is the number of processors/cores found on the host OS of the query (since 1.7). PARALLEL execution is useful on long running queries or queries that involve multiple clusters. On simple queries using PARALLEL could cause a slow down due to the overhead inherent with using multiple threads

NOTE: Starting from 1.0rc7 the `RANGE` keyword has been removed. To execute range queries use the `BETWEEN` operator against `@rid` as explained in Pagination.

Projections

In the standard SQL, projections are mandatory. In OrientDB if it's omitted, the entire record set is returned. It is the equivalent of the `*` keyword. Example:

```
SELECT FROM Account
```

With all projections except the wildcard "*", a new temporary document is created and the `@rid` and `@version` of the original record will not be included.

```
SELECT name, age FROM Account
```

The conventional naming for the returned document's fields are:

- the field name for plain fields `invoice -> invoice`
- the first field name for chained fields, like `invoice.customer.name -> invoice`
- the name of the function for functions, like `max(salary) -> max`

If the target field already exists, a progressive number is used as a prefix. Example:

```
SELECT max(incoming), max(cost) FROM Balance
```

Will return a document with the field `max` and `max2`.

To override the field name, use `AS`. Example:

```
SELECT max(incoming) AS max_incoming, max(cost) AS max_cost FROM Balance
```

By using the dollar (`$`) as a prefix, you can access context variables. Each time you run a command, OrientDB accesses the context to read and write variables. Here's an example to display the path and depth level of the traversal on all the movies, up to the 5th level of depth:

```
SELECT $path, $depth FROM ( TRAVERSE * FROM Movie WHERE $depth <= 5 )
```

Examples

Get all the records of type `Person` where the name starts with `Luk`:

```
select * from Person where name like 'Luk%'
```

or

```
select * from Person where name.left(3) = 'Luk'
```

or

```
select * from Person where name.substring(0,3) = 'Luk'
```

Get all the records of type `!AnimalType` where the collection `races` contains at least one entry where the first character of the name, ignoring the case, is equal to `e`:

```
select * from animaltype where races contains (name.toLowerCase().subString(0,1) = 'e')
```

Get all the records of type `!AnimalType` where the collection `races` contains at least one entry with name `European` or `Asiatic`:

```
select * from animaltype where races contains (name in ['European','Asiatic'])
```

Get all the records of type `Profile` where any field contains the word `danger`:

```
select from profile where any() like '%danger%'
```

Get any record at any level that has the word `danger`:

```
select from profile where any() traverse ( any() like '%danger%' )
```

Get all the records where up to the 3rd level of connections has some field that contains the word `danger` ignoring the case:

```
select from Profile where any() traverse( 0,3 ) ( any().toUpperCase().indexOf( 'danger' ) >
```

Order the result set by the `name` in descending order:

```
select from Profile order by name desc
```

Returns the total of records per city:

```
select sum(*) from Account group by city
```

Traverse record starting from a root node:

```
select from 11:4 where any() traverse(0,10) (address.city = 'Rome')
```

Query only a set of records:

```
select from [#10:3, #10:4, #10:5]
```

Select only three fields from Profile:

```
select nick, followings, followers from Profile
```

Select the `name` field in upper-case and the `country name` of the linked city of the address:

```
select name.toUppercase(), address.city.country.name from Profile
```

Order by record creation. Starting from 1.7.7, using the expression "order by @rid desc", allows OrientDB to open an Inverse cursor against clusters. This is extremely fast and

doesn't require classic ordering resources (RAM and CPU):

```
select from Profile order by @rid desc
```

LET block

The `LET` block contains the list of context variables to assign each time a record is evaluated. These values are destroyed once the query execution ends. Context variables can be used in projections, conditions and sub-queries.

Assign fields to reuse multiple times

OrientDB allows crossing relationships, but if in a single query you need to evaluate the same branch of nested relationship, it's definitely better using a context variable that refers to the full relationship.

Example:

```
SELECT FROM Profile
WHERE address.city.name like '%Saint%"' and
    ( address.city.country.name = 'Italy' or address.city.country.name = 'France' )
```

Using LET becomes shorter and faster, because the relationships are traversed only once:

```
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 the path till `address.city` is traversed only once.

Sub-query

LET block allows you to assign a context variable the result of a sub-query. Example:

```
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
```

Usage in projection

Context variables can be part of result set used in Projections. The example below displays the city name of the previous example:

```
SELECT $temp.name FROM Profile
LET $temp = address.city
WHERE $city.name like '%Saint%"' and
    ( $city.country.name = 'Italy' or $city.country.name = 'France' )
```

Conclusion

To know more about other SQL commands, take a look at SQL commands.

History

1.7.7

New targets `indexvalues:`, `indexvaluesasc:`, `indexvaluesdesc:` are added.

1.7

- Added **PARALLEL** keyword to execute the query against X concurrent threads, where X is the number of processors/core found on the os where the query is running (since 1.7). PARALLEL execution is useful on long running query or query that involve multiple clusters. On simple queries, using PARALLEL could cause a slow down because of the overhead on using multiple threads

SQL - INSERT

The **Insert** command creates a new record in the database. Records can be schema-less or conform to rules you specify 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>]
```

Where:

- `CONTENT`, JSON data as an option to set fields values
- `RETURN`, returns an expression instead of the number of inserted records. You can use any valid SQL expression. The most common use cases include:
  - `@rid` to return the record id of the new record
  - `@this` to return the entire new record
- `FROM`, inserts values from the resultset of a query. Since v1.7.

Examples

Insert a new record with name 'Jay' and surname 'Miner'

SQL-92 syntax:

```
insert into Profile (name, surname) values ('Jay', 'Miner' )
```

OrientDB abbreviated syntax:

```
insert into Profile SET name = 'Jay', surname = 'Miner'
```

JSON content syntax:

```
insert into Profile CONTENT {"name": "Jay", "surname" = "Miner"}
```

Insert a new record of type Profile, but in a different cluster than the default one

```
insert into Profile cluster profile_recent (name, surname) values ('Jay', 'Miner' )
```

```
insert into Profile cluster profile_recent set name = 'Jay', surname = 'Miner'
```

Insert several records at the same time

```
insert into Profile(name,surname) VALUES ('Jay','Miner'),('Frank','Hermier'),('Emily','Saut'
```

Insert a new record adding a relationship

```
insert into Employee (name, boss) values ('jack', #11:99 )
```

```
insert into Employee SET name = 'jack', boss = #11:99
```

Insert a new record adding a collection of relationship

```
insert into Profile (name, friends) values ('Luca', [#10:3, #10:4] )
```

```
insert into Profile SET name = 'Luca', friends =  [#10:3, #10:4]
```

Sub-selects

```
insert into Diver SET name = 'Luca', buddy = (select from Diver where name = 'Marko')
```

Sub-inserts

```
insert into Diver SET name = 'Luca', buddy = (insert into Diver name = 'Marko')
```

Insert in a different cluster

This inserts a new document in the cluster 'asiaemployee':

```
insert into cluster:asiaemployee (name) values ('Mattew')
```

But note that the document will have no class assigned. To create a document of a certain class but in a different cluster than the default one use:

```
insert into cluster:asiaemployee (@class, content) values('employee', 'Mattew')
```

That will insert the document of type 'employee' in the cluster 'asiaemployee'.

Insert a new record adding an embedded document

```
insert into Profile (name, address) values ('Luca', { "@type" : "d", "street" : "Melrose Ave
```

Insert from query

Copy records in another class

```
insert into GermanyClient from ( select from Client where country = 'Germany' )
```

Will insert all the records from Client where the country is "Germany".

Copy records in another class adding a field

```
insert into GermanyClient from ( select *, true as copied from Client where country = 'Germa
```

Will insert all the records from Client where the country is "Germany" and will add an additional field called "copied" with value true.

To know more about other SQL commands look at SQL commands.

SQL - UPDATE

Update one or more records in the current database. Remember that OrientDB can work also in schema-less mode, so you can create any field on-the-fly. Furthermore, OrientDB works on collections. This is the reason why OrientDB SQL has some extensions to handle 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>]
```

Where:

- **SET** updates the field
- **INCREMENT** increments the field by the value. If the record had 10 as a value and "INCREMENT value = 3" is executed, then the new value will be 13. This is useful for atomic updates of counters. Use negative numbers to decrement. INCREMENT can be used to implement sequences (autoincrement)
- **ADD**, adds a new item in collection fields
- **REMOVE**, removes an item in collection and maps fields
- **PUT**, puts an entry into map fields
- **CONTENT**, replaces the record content with a JSON
- **MERGE**, merges the record content with a JSON
- **LOCK** specifies how the record is locked between the load and the update. It can be a value between:
  - *DEFAULT*, no lock. In case of concurrent update, the MVCC throws an exception
  - *RECORD*, locks the record during the update

- **UPSERT** updates a record if it already exists, or inserts a new record if it does not, all in a single statement. This avoids the need to execute 2 commands, one for the query and a conditional insert/update. UPSERT requires a WHERE clause and a class target
- **RETURN** specifies what to return as `<returning>`. If `<returning-expression>` is specified (optional) and returning is BEFORE or AFTER, then the expression value is returned instead of record. `<returning>` can be a value between:
  - **COUNT**, the default, returns the number of updated records
  - **BEFORE**, returns the records before the update
  - **AFTER**, returns the records after the update

- WHERE, SQL-Where condition to select records to update
- LIMIT, sets the maximum number of records to update

- TIMEOUT, if any limits the update operation to a timeout

Note that RecordID must be prefixed with '#'. Example: #12:3.

To know more about conditions, take a look at WHERE conditions.

Examples

Example 1: Change the value of a field

```
> UPDATE Profile SET nick='Luca' WHERE nick IS NULL

Updated 2 record(s) in 0,008000 sec(s).
```

Example 2: Remove a field from all the records

```
> UPDATE Profile REMOVE nick
```

Example 3: Add a value into a collection

```
> UPDATE Account ADD addresses=#12:0
```

Example 4: Remove a value from a collection

```
> UPDATE Account REMOVE addresses=#12:0
```

Example 5: Put a map entry into a map

```
> UPDATE Account PUT addresses='Luca', #12:0
```

Example 6: Remove a value from a map

```
> UPDATE Account REMOVE addresses='Luca'
```

Example 7: Update an embedded document

Update command can take a JSON as value to update:

```
> UPDATE Account SET address={"street":"Melrose Avenue", "city":{"name":"Beverly Hills"}}
```

Example 8: Update the first 20 records that satisfy a condition

```
> UPDATE Profile SET nick='Luca' WHERE nick IS NULL LIMIT 20
```

Example 9: Update a record or insert if it does not already exist

```
> UPDATE Profile SET nick='Luca' UPSERT WHERE nick='Luca'
```

Example 10: Update a web counter, avoiding concurrent accesses

```
> UPDATE Counter INCREMENT viewes = 1 WHERE page='/downloads/' LOCK RECORD
```

Example 11: Usage of RETURN keyword

```
UPDATE #7:0 SET gender='male' RETURN AFTER @rid
UPDATE #7:0 SET gender='male' RETURN AFTER @version
UPDATE #7:0 SET gender='male' RETURN AFTER @this
UPDATE #7:0 INCREMENT Counter = 123 RETURN BEFORE $current.Counter
UPDATE #7:0 SET gender='male' RETURN AFTER $current.exclude("really_big_field")
UPDATE #7:0 ADD out_Edge = #12:1 RETURN AFTER $current.outE("Edge")
```

In case a single field is returned, the result is wrapped in a record storing value in "result" field (Just to avoid introducing new serialization – there is no primitive-values collection serialization in binary protocol). Additionally to that, useful fields like version and rid of original record is provided in corresponding fields. New syntax will allow optimizing client-server network traffic.

To know more about the SQL syntax used in Orient, take a look at: SQL-Query.

SQL - DELETE

The **Delete** command deletes one or more records from the database. The set of records involved are taken by the WHERE clause.

NOTE: Don't use SQL DELETE to remove Vertices or Edges but use the DELETE VERTEX and DELETE EDGE commands that assure the integrity of the graph.

Syntax

```
DELETE FROM <Class>|cluster:<cluster>|index:<index> [LOCK <default|record>] [RETURN <returni
  [WHERE <Condition>*] [LIMIT <MaxRecords>] [TIMEOUT <timeout>]
```

Where:

- **LOCK** specifies how the record is locked between the load and the delete. It can be a value between:
  - *DEFAULT*, no lock. In case of concurrent delete, the MVCC throws an exception
  - *RECORD*, locks the record during the delete

- **RETURN** specifies what to return. It can be a value between:
  - **COUNT**, the default, returns the number of deleted records
  - **BEFORE**, returns the records before the delete

- WHERE, SQL-Where condition to select records to update
- LIMIT, sets the maximum number of records to update
- TIMEOUT, if any limits the update operation to a timeout

Examples

Delete all the records with surname equals to 'unknown' ignoring the case:

```
delete from Profile where surname.toLowerCase() = 'unknown'
```

To know more about other SQL commands look at SQL commands.

SQL - ALTER CLASS

The **Alter Class** command alters a class in the schema.

Syntax

```
ALTER CLASS <class> <attribute-name> <attribute-value>
```

Where:

- **class** is the class name to change
- **attribute-name**, is the attribute name to alter. Supported attribute names are:
  - **NAME**, the class name. Accepts a string as value
  - **SHORTNAME**, the short name. Accepts a string as value. NULL to remove it
  - **SUPERCLASS**, the superclass name to assign. Accepts a string as value. NULL to remove it
  - **OVERSIZE**, the oversize factor. Accepts a decimal number as value
  - **ADDCLUSTER**, add a cluster to be part of the class. If the cluster doesn't exist, a physical cluster is created automatically. See also Create Cluster command. Adding clusters to classes is useful also to store records in distributed servers. Look at Distributed Sharding
  - **REMOVECLUSTER**, remove a cluster from a class. The cluster will be not deleted.
  - **STRICTMODE**, enable or disable the strict mode. With the strict mode enabled you work in schema-full mode and you can't add new properties in record if the class
  - **CLUSTERSELECTION** sets the strategy used on selecting the cluster where to create new records. On class creation the settings is inherited by database's cluster-selection property. For more information look also at Cluster Selection
  - **CUSTOM**, to set custom properties. Property name and value must be expressed using the syntax: "`<name>=<value>`" without spaces between name and value

- **attribute-value**, is the new attribute value to set

See also

- create class
- drop class
- alter cluster
- SQL commands
- Console commands

Examples

Change the name of the class 'Account':

```
ALTER CLASS Account NAME Seller
```

Change the oversize factor of the class 'Account':

```
ALTER CLASS Account OVERSIZE 2
```

Adds a cluster by name to a class. If the cluster didn't exist, it's created automatically:

```
ALTER CLASS Account ADDCLUSTER account2
```

Removes a cluster by id to a class without dropping the cluster:

```
ALTER CLASS Account REMOVECLUSTER 34
```

Add custom properties (in this case used in Record level security):

```
ALTER CLASS Post CUSTOM onCreate.fields=_allowRead,_allowUpdate
ALTER CLASS Post CUSTOM onCreate.identityType=role
```

Create a new cluster to the class and set the cluster-selection strategy as "balanced":

```
CREATE CLUSTER Employee_1
ALTER CLASS ADDCLUSTER Employee_1
ALTER CLASS CLUSTERSELECTION balanced
```

History

1.7

- Added support for CLUSTERSELECTION that sets the strategy used on selecting the cluster where to create new records

SQL - ALTER CLUSTER

The **Alter Cluster** command updates a cluster.

Syntax

```
ALTER CLUSTER <cluster-name>|<cluster-id> <attribute-name> <attribute-value>
```

Where:

- **cluster-name** name of the cluster to modify
- **cluster-id** id of the cluster to modify
- **attribute-name** between those supported:
  - **NAME** cluster's name
  - **STATUS** change the cluster's status. Allowed values: ONLINE, OFFLINE. By default clusters are ONLINE. To put cluster offline, change it status to OFFLINE. Once offline, the physical files of the cluster will be not open by OrientDB. This feature is useful when you want to archive old data elsewhere and restore when needed
  - **COMPRESSION** compression used between: nothing, snappy, gzip and any other compression registered in OCompressionFactory class. OrientDB calls the compress() method every time it saves a record to the storage, and uncompress() every time it loads a record from the storage. You can also use the OCompression interface to manage encryption
  - **USE_WAL** use the Journal (Write Ahead Log) when OrientDB operates against the cluster
  - **RECORD_GROW_FACTOR** grow factor to save more space on record creation. This is useful when you plan to update the record with additional information. Bigger record avoids defragmentation because OrientDB has not to find a new space in case of update with more data
  - **RECORD_OVERFLOW_GROW_FACTOR** like RECORD_GROW_FACTOR, but on update. When the size limit is reached this setting is considered to get more space (factor > 1)
  - **CONFLICTSTRATEGY**, (since 2.0) 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. If not defined a strategy at cluster level, the database configuration is taken (use ALTER DATABASE 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, throws an exception when versions are different
  - `content`, in case the version is different, it checks if the content is changed, otherwise use the highest version and avoid throwing exception

    - `automerge`, merges the changes

- **attribute-value** attribute's value to set

See also

- create cluster
- drop cluster
- alter class
- SQL commands
- Console commands

Examples

```
ALTER CLUSTER profile NAME profile2
```

```
ALTER CLUSTER 9 NAME profile2
```

```
ALTER CLUSTER V CONFLICTSTRATEGY automerge
```

Put a cluster offline

```
ALTER CLUSTER V_2012 STATUS OFFLINE
```

To know more about other SQL commands, take a look at SQL commands.

SQL - ALTER DATABASE

The **Alter Database** command update database settings.

Syntax

```
ALTER DATABASE <attribute-name> <attribute-value>
```

Where: **attribute-value** attribute's value to set and **attribute-name** between those supported:

- **STATUS** database's status between:
- **IMPORTING** to set importing 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 2.0) 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

See also

- Console Command Create Database
- Console Command Drop Database

Examples

Change the database type to "graph"

```
ALTER DATABASE TYPE graph
```

Use GraphDB created with releases before 1.4

Starting from v 1.4, OrientDB can use Lightweight Edges. After v2.0 this is disabled by default with new databases. To maintain the compatibility with OrientDB 1.4 or minor execute this commands:

```
alter database custom useLightweightEdges=false
alter database custom useClassForEdgeLabel=false
alter database custom useClassForVertexLabel=false
alter database custom useVertexFieldsForEdgeLabels=false
```

History

1.7

- Added support for CLUSTERSELECTION that sets the strategy used on selecting the cluster where to create new records
- Added **MINIMUMCLUSTERS** to pre-create X clusters every time a new class is created

SQL - ALTER PROPERTY

The **Alter Property** command alters a class's property in the schema.

Syntax

```
ALTER PROPERTY <class>.<property> <attribute-name> <attribute-value>
```

Where:

- **class** is the class owner of the property to change
- **property** is the property name to change
- **attribute-name**, is the attribute name to alter
- **attribute-value**, is the new attribute value to set

Supported attribute names are:

- **LINKEDCLASS**, the linked class name. Accepts a string as value. NULL to remove it
- **LINKEDTYPE**, the linked type name between those supported:Types. Accepts a string as value. NULL to remove it
- **MIN**, the minimum value as constraint. Accepts strings, numbers or dates as value. NULL to remove it
- **MANDATORY**, true if the property is mandatory. Accepts "true" or "false"
- **MAX**, the maximum value as constraint. Accepts strings, numbers or dates as value. NULL to remove it
- **NAME**, the property name. Accepts a string as value
- **NOTNULL**, the property can't be null. Accepts "true" or "false"
- **REGEXP** the regular expression as constraint. Accepts a string as value. NULL to remove it
- **TYPE**, the type between those supported:Types Accepts a string as value
- **COLLATE**, set the collate to define the strategy of comparison. By default is case sensitive. By setting it yo "ci", any comparison will be case-insensitive
- **CUSTOM** Set custom properties. Syntax is `<name> = <value>`. Example: stereotype = icon

Examples

Change the name of the property 'age' in class 'Account' in 'born'

```
ALTER PROPERTY Account.age NAME born
```

Set a property as mandatory

```
ALTER PROPERTY Account.age MANDATORY true
```

Set a regexp

```
ALTER PROPERTY Account.gender REGEXP [M|F]
```

Set a field as case insensitive to comparison

```
alter property Employee.name collate ci
```

Set a custom field on property

```
alter property Foo.bar1 custom stereotype = visible
```

To create a property use the Create Property command, to remove a property use the Drop Property command.

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

SQL - CREATE CLASS

The **Create Class** command creates a new class in the schema. *NOTE: If a cluster with the same name exists in the database will be used as default cluster.*

Syntax

```
CREATE CLASS <class> [EXTENDS <super-class>] [CLUSTER <clusterId>*]
```

Where:

- *class* is the class name to create. The first character must be alphabetic and others can be any alphanumeric characters plus underscore _ and dash -.
- *super-class*, optional, is the super-class to extend
- *clusterId* can be a list separated by comma (,)

By default OrientDB creates 1 cluster per class, but this can be changed by setting the property `minimumclusters` at database level.

Cluster selection strategy

OrientDB, by default, inherits the cluster selection by the database. By default is round-robin, but you can always change it after creation with alter class command. The 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

See also

- alter class
- drop class
- create cluster
- SQL commands
- Console commands

Examples

Create the class 'Account':

```
CREATE CLASS Account
```

Create the class 'Car' that extends 'Vehicle':

```
CREATE CLASS Car extends Vehicle
```

Create the class 'Car' with clusterId 10:

```
CREATE CLASS Car CLUSTER 10
```

Abstract class

Create the class 'Person' as ABSTRACT:

```
CREATE CLASS Person ABSTRACT
```

SQL - CREATE CLUSTER

The **Create Cluster** command creates a new cluster in database. Once created, the cluster can be used to save records by specifying its name during save. If you want to add the cluster to a class, use rather the Alter Class command using ADDCLUSTER property.

Syntax

```
CREATE CLUSTER <name> [POSITION <position>|append]
```

Where:

- *name* is the cluster name to create. The first character must be alphabetic and others can be any alphanumeric characters plus underscore _ and dash -.
- *position*, optional, is the position where to add the cluster. If omitted or it's equals to 'default' the cluster is appended at the end

Examples

Create the cluster 'Account':

```
CREATE CLUSTER account
```

To remove a cluster use the Drop Cluster command.

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

SQL - CREATE EDGE

This command creates a new Edge into the database. Edges, together with Vertices, are the main components of a Graph. OrientDB supports polymorphism on edges. The base class is "E" (before 1.4 was called "OGraphEdge"). Look also how to Create Vertex.

	***NOTE: While running as distributed, edge creation could be done in two steps (create+update). This could break some constraint defined at Edge's class level. To avoid this kind of problem disable the constrains in Edge's class.***

Syntax

```
CREATE EDGE <class> [CLUSTER <cluster>] FROM <rid>|(<query>)|[<rid>]* TO <rid>|(<query>)|[<r
                    [SET <field> = <expression>[,]*]|CONTENT {<JSON>}
                    [RETRY <retry> [WAIT <pauseBetweenRetriesInMs]]
```

Where:

- **class**, is the Edge's class name, or "E" if you don't use sub-types
- **cluster**, is the cluster name where to physically store the edge
- **JSON**, is the JSON content to set as record content, instead of field by field
- **retry**, is the number of retries in case of conflict (optimistic approach)
- **pauseBetweenRetriesInMs**, are the milliseconds of delay between retries

Examples

Create a new edge between two vertices of the base class 'E', namely OGraphEdge

```
create edge from #10:3 to #11:4
```

Create a new edge type and a new edge of the new type

```
create class E1 extends E
create edge E1 from #10:3 to #11:4
```

Create a new edge in a particular cluster

```
create edge E1 cluster EuropeEdges from #10:3 to #11:4
```

Create a new edge setting properties

```
create edge from #10:3 to #11:4 set brand = 'fiat'
```

Create a new edge of type E1 setting properties

```
create edge E1 from #10:3 to #11:4 set brand = 'fiat', name = 'wow'
```

Create new edges of type Watched between all the action Movies and me using sub-queries

```
create edge Watched from (select from account where name = 'Luca') to (select from movies wh
```

Create an edge with JSON content

```
create edge E from #22:33 to #22:55 content { "name" : "Jay", "surname" : "Miner" }
```

History and compatibility

- 1.1: first version
- 1.2: the support for query and collection of RIDs in FROM/TO
- 1.4: the command uses the Blueprints API under the hood, so if you're working in Java using the OGraphDatabase API you could experience in some difference how edges are managed. To force the command to work with the "old" API change the GraphDB settings described in Graph backward compatibility
- 2.0: New databases have Lightweight Edges disabled by default, so this command creates regular edges.

To know more about other SQL commands look at SQL commands.

SQL - CREATE FUNCTION

The **Create Function** command creates a new Server-Side function. Functions can be executed from SQL, HTTP and Java.

Syntax

```
CREATE FUNCTION <name> <code>
                [PARAMETERS [<comma-separated list of parameters' name>]]
                [IDEMPOTENT true|false]
                [LANGUAGE <language>]
```

Where:

- **name** is the function name as string
- **code** is the function code as string
- **PARAMETERS**, optional, are the function parameters bound to the execution heap
- **IDEMPOTENT**, optional, means the function doesn't change the database status. This is useful because IDEMPOTENT functions can be called by HTTP GET, otherwise HTTP POST. By default is FALSE.
- **language**, optional, is the language. By default is "Javascript".

See also

- Functions
- SQL commands
- Console commands

Examples

Create function 'test' in Javascript with no parameters:

```
CREATE FUNCTION test "print('\nTest!')"
```

Create function 'allUsersButAdmin' in SQL with no parameters:

```
CREATE FUNCTION allUsersButAdmin "select from ouser where name <> 'admin'" LANGUAGE SQL
```

SQL - CREATE INDEX

Creates a new index. To create an automatic index bound to a schema property use section "**ON**" of create index command or use as name the <**class.property**> notation. But assure to have created the schema for it before the index. See the example below.

Indexes can be:

- **UNIQUE**, doesn't allow duplicated
- **NOTUNIQUE**, allows duplicates
- **FULLTEXT**, by indexing any single word of the text. It's used in query with the operator CONTAINSTEXT

Syntax

```
CREATE INDEX <name> [ON <class-name> (prop-names)] <type> [<key-type>]
             METADATA [{<json-metadata>}]
```

Where:

- **name** logical name of index. Can be `<class>.<property>` to create an automatic index bound to a schema property. In this case **class** is the class of the schema and **property**, is the property created into the class. Notice that in another case index name can't contain '.' symbol
- **class-name** name of class that automatic index created for. Class with such name must already exist in database
- **prop-names** comma-separated list of properties that this automatic index is created for. Property with such name must already exist in schema. If property belongs to one of the Map types (LINKMAP, EMBEDDEDMAP) you can specify will be keys or values used for index generation. Use "by key" or "by value" expressions for that, if nothing will be specified keys will be used during index creation.
- **type**, between 'unique', 'notunique' and 'fulltext'
- **key-type**, is the type of key (Optional). On automatic indexes is auto-determined by reading the target schema property where the index is created. If not specified for manual indexes, at run-time during the first insertion the type will be auto determined by reading the type of the class. In case of creation composite index it is a comma separated list of types.
- **metadata** is a json representing all the additional metadata as key/value

If "**ON**" and **key-type** sections both exist database validate types of specified properties. And if types of properties not equals to types specified in **key-type** list, exception will be thrown.

List of key types can be used for creation manual composite indexes, but such indexes don't have fully support yet.

See also

- SQL Drop Index
- Indexes
- SQL commands

Examples

Examples of non-automatic index to store dates manually:

```
CREATE INDEX mostRecentRecords unique date
```

Examples of automatic index bound to the property "id" of class "User":

```
CREATE PROPERTY User.id BINARY
CREATE INDEX User.id UNIQUE
```

Examples of index for "thumbs" property of class "Movie".

```
CREATE INDEX thumbsAuthor ON Movie (thumbs) unique;
CREATE INDEX thumbsAuthor ON Movie (thumbs by key) unique;
CREATE INDEX thumbsValue ON Movie (thumbs by value) unique;
```

Composite index example:

```
CREATE PROPERTY Book.author STRING
CREATE PROPERTY Book.title STRING
CREATE PROPERTY Book.publicationYears EMBEDDEDLIST INTEGER
CREATE INDEX books ON Book (author, title, publicationYears) UNIQUE
```

Index by Edge's date range

You can create an index against the edge class if it's containing the begin/end date range of validity. This is a very common use case with historical graphs. Consider this File system example:

```
CREATE CLASS File EXTENDS V
CREATE CLASS Has EXTENDS E
```

```
CREATE PROPERTY Has.started DATETIME
CREATE PROPERTY Has.ended DATETIME
CREATE INDEX Has.started_ended ON Has (started, ended) NOTUNIQUE
```

And then you can retrieve all the edge that existed in 2014:

```
SELECT FROM Has Where started >= '2014-01-01 00:00:00.000' and ended < '2015-01-01 00:00:00.
```

To have the connected parent File:

```
SELECT outV() FROM Has Where started >= '2014-01-01 00:00:00.000' and ended < '2015-01-01 00
```

To have the connected children Files:

```
SELECT inV() FROM Has Where started >= '2014-01-01 00:00:00.000' and ended < '2015-01-01 00:
```

Null values

Indexes by default ignore null values. For such reason queries against NULL value that use indexes return no entries.

If you want to index also null values set `{ ignoreNullValues : false }` as metadata. Example:

```
CREATE INDEX addresses ON Employee (address) notunique
             METADATA {ignoreNullValues : false}
```

SQL - CREATE LINK

The **Create Link** transform two simple values in a link. This is very useful when you're importing data from a Relational database. In facts in the Relational world relationships are resolved as foreign keys.

This is not the way to create links in general, but a way to convert two values in two different classes in a link. To create a link in OrientDB look at Relationships. For more information about importing a Relational Database into OrientDB look at Import from RDBMS to Document Model.

Consider this example where the class "Post" has a relationship 1-N to "Comment":

```
Post 1 ---> * Comment
```

In a Relational database you'll have something like that:

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

Using OrientDB, instead, you have direct relationship as in your object model. So the navigation is from *Post* to *Comment* and not viceversa as for Relational model. For this reason you need to create a link as **INVERSE**.

Syntax

```
CREATE LINK <link-name> TYPE [<link-type>] FROM <source-class>.<source-property> TO
<destination-class>.<destination-property> [INVERSE]
```

Where:

- **link-name** is the name of the property for the link. If not expressed will be overwritten the *destination-property* field
- **link-type**, optional, is the type to use for the link. In case of inverse relationships (the most commons) you can specify LINKSET or LINKLIST for 1-N relationships
- **source-class**, is the source class
- **source-property**, is the source property
- **destination-class**, is the destination class
- **destination-property**, is the destination property
- **INVERSE**, tells to create the connection on the opposite direction. This is common when you've imported 1-N relationships from a RDBMS where they are mapped at the opposite direction

Examples

```
CREATE LINK comments TYPE LINKSET FROM comments.PostId TO posts.Id INVERSE
```

To know more about other SQL commands look at SQL.

SQL - CREATE PROPERTY

The **Create Property** command creates a new property in the schema. An existing class is required to perform this command.

Syntax

```
CREATE PROPERTY <class>.<property> <type> [<linked-type>|<linked-class>] [UNSAFE]
```

Where:

- **class** is the class of the property
- **property**, is the property created in the **class**
- **type**, the data type of the property. See Types. Valid options are:
  - **boolean**
  - **integer**
  - **short**
  - **long**
  - **float**
  - **double**
  - **date**
  - **string**
  - **binary**
  - **embedded**
  - **embeddedlist**, an ordered collection of items that supports duplicates. Optionally accepts the parameter *linked-type* or *linked-class* to specify the collection's content
  - **embeddedset**, an unordered collection of items that does not support duplicates. Optionally accepts the parameter *linked-type* or *linked-class* to specify the collection's content
  - **embeddedmap**, a map of key/value entries. Optionally accepts the parameter *linked-type* or *linked-class* to specify the map's value content
  - **link**
  - **linklist**, an ordered collection of items that supports duplicates. Optionally accepts the parameter *linked-class* to specify the linked record's class
  - **linkset**, an unordered collection of items that does not support duplicates. Optionally accepts the parameter *linked-class* to specify the linked record's class
  - **linkmap**, this is a map of key/ entries. Optionally accepts the parameter *linked-class* to specify the map's value record class
  - **byte**

- **linked-type**, the contained type in EMBEDDEDSET, EMBEDDEDLIST and EMBEDDEDMAP types (see above). See also Types. Valid options are:

    - **boolean**
    - **integer**
    - **short**
    - **long**
    - **float**
    - **double**
    - **date**
    - **string**
    - **binary**
    - **embedded**
    - **link**
    - **byte**

- **linked-class**, the contained class in containers (see above).
- `UNSAFE`, optional, avoid check on existent records. With millions of records this operation could take time. If you are sure the property is new, you can skip the check by using `UNSAFE`. Since 2.0.

Examples

Create the property 'name' of type 'STRING' in class 'User':

```
CREATE PROPERTY user.name STRING
```

Create a list of Strings as property 'tags' of type 'EMBEDDEDLIST' in class 'Profile'. The linked type is 'STRING':

```
CREATE PROPERTY profile.tags EMBEDDEDLIST STRING
```

Create the property 'friends' of type 'EMBEDDEDMAP' in class 'Profile'. The linked class is profile itself (circular references):

```
CREATE PROPERTY profile.friends EMBEDDEDMAP Profile
```

To remove a property use the SQL Drop Property command.

To learn more about other SQL commands look at SQL commands.

This is a command of the Orient console. To learn all available commands go to Console-Commands.

SQL - CREATE VERTEX

This command creates a new Vertex into the database. Vertices, together with Edges, are the main components of a Graph. OrientDB supports polymorphism on vertices. The base class is "V" (before 1.4 was called "OGraphVertex"). Look also how to Create Edges.

	***NOTE: While running as distributed, vertex creation could be done in two steps (create+update). This could break some constraint defined at Vertex's class level. To avoid this kind of problem disable the constrains in Vertex's class.***

Syntax

```
CREATE VERTEX [<class>] [CLUSTER <cluster>] [SET <field> = <expression>[,]*]
```

Examples

Create a new vertex of the base class 'V', namely OGraphVertex

```
create vertex
```

Create a new vertex type and a new vertex of the new type

```
create class V1 extends V
create vertex V1
```

Create a new vertex in a particular cluster

```
create vertex V1 cluster recent
```

Create a new vertex setting properties

```
create vertex set brand = 'fiat'
```

Create a new vertex of type V1 setting properties

```
create vertex V1 set brand = 'fiat', name = 'wow'
```

Create a vertex with JSON content

```
create vertex Employee content { "name" : "Jay", "surname" : "Miner" }
```

History and Compatibility

- 1.1: first version
- starting from v.1.4 the command uses the Blueprints API under the hood, so if you're working in Java using the OGraphDatabase API you could experience in some difference how edges are managed. To force the command to work with the "old" API change the GraphDB settings described in Graph backward compatibility

To know more about other SQL commands look at SQL commands.

SQL - MOVE VERTEX

This command moves a Vertex into another class or cluster.

Syntax

```
MOVE VERTEX <source> TO <destination> [SET [<field>=<value>]* [,]] [MERGE <JSON>]
```

Where:

- `source` are the vertices to move. This could be one of the following values:
  - A **single vertex** by RID. Example: `MOVE VERTEX #34:232 TO CLASS:Provider`
  - An **array of vertices** by RIDs. Example: `MOVE VERTEX [#34:232,#34:444] TO CLASS:Provider`
  - A **subquery** with vertices as result. All the returning vertices will be moved. Example: `MOVE VERTEX (SELECT FROM V WHERE city = 'Rome') TO CLASS:Provider`

- `destination` is the location where to move vertices. Can be one of the followings:
  - **Class**, by using the syntax `CLASS:<class-name>`. Use this to refactor your graph assigning a new class to vertices
  - **Cluster**, by using the syntax `CLUSTER:<cluster-name>`. Use this to move your vertices on different clusters in the same class. This is useful on Distributed Configuration where you can move vertices on other servers

- `SET` optional block contains the pairs of values to assign during the moving. The syntax is the same as SQL UPDATE. Example: `MOVE VERTEX (SELECT FROM V WHERE type = 'provider') TO CLASS:Provider SET movedOn = Date()`
- `MERGE` optional block gets a JSON containing the pairs of values to assign during the moving. The syntax is the same as SQL UPDATE. Example: `MOVE VERTEX (SELECT FROM V WHERE type = 'provider') TO CLASS:Provider MERGE { author : 'Jay Miner' }`

See also

- Create Vertex
- Create Edge

History and Compatibility

- 2.0: first version

Examples

Refactoring of graph by adding sub-types

It's very common the case when you start modeling your domain in a way, but then you need more flexibility. On this example we want to split all the "Person" vertices under 2 new sub-types called "Customer" and "Provider" respectively. At the end we declare Person as abstract class.

```
CREATE CLASS Customer EXTENDS Person
CREATE CLASS Provider EXTENDS Person
MOVE (SELECT FROM Person WHERE type = 'Customer') TO CLASS:Customer
MOVE (SELECT FROM Person WHERE type = 'Provider') TO CLASS:Provider
ALTER CLASS Person ABSTRACT TRUE
```

Move vertices on different servers

OrientDB allows you to scale up by just adding servers. As soon as you add a new server, OrientDB creates automatically a new cluster with the name of the class plus the node name. Example: "customer_europe". Partitioning is a best practice when you need to scale up, specially on writes. If you have a graph with "Customer" vertices and you want to move some vertices to other server you can move them to the cluster owned by the server where you want your vertices are moved.

With this example, we're moving all the customers that live in Italy, Germany or UK to the "customer_europe" cluster assigned to the node "Europe". In this way all the access to European customers will be faster to the applications connected to the European node:

```
MOVE (SELECT FROM Customer WHERE ['Italy', 'Germany', 'UK'] IN out('city').out('country') )
```

To know more about other SQL commands look at SQL commands.

SQL - DELETE EDGE

This command deletes one or more edges from the database. Use this command if you work against graphs. The "Delete edge" command takes care to remove all the cross references to the edge in both "in" and "out" vertices.

Syntax

```
DELETE EDGE <rid>|FROM <rid>|TO <rid>|[<class>] [WHERE <conditions>]> [LIMIT <MaxRecords>]
```

The WHERE clause is common to the other SQL commands.

History and Compatibility

- 1.1: first version
- 1.4: the command uses the Blueprints API under the hood, so if you're working in Java using the OGraphDatabase API you could experience in some difference how edges are managed. To force the command to work with the "old" API change the GraphDB settings described in Graph backward compatibility

Examples

Delete edges where date is a property which might exist in one of more edges between the two vertices:

```
DELETE EDGE from #11:101 TO #11:117 Where date >= "2012-01-15"
```

Deletes edges filtering also by Edge's class:

```
DELETE EDGE FROM #11:101 TO #11:117 WHERE @class = 'owns' and comment like "regex of forbidd
```

This is the faster alternative to `DELETE EDGE WHERE @class = 'owns' and date < "2011-11"`:

```
DELETE EDGE Owns WHERE date < "2011-11"
```

Deletes edges where in.price shows the condition on 'to vertex' for the edge

```
DELETE EDGE Owns WHERE date < "2011-11" and in.price >= 202.43
```

Deleting Edge using Java Code:

When User follow a company We create edge between User and company of type followCompany and CompanyFollowedBy class

```
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);
}
```

SQL - DELETE VERTEX

This command deletes one or more vertices from the database. Use this command if you work against graphs. The "Delete Vertex" (like the Delete Edge) command takes care to remove all the cross references to the vertices in all the edges involved.

Syntax

```
DELETE VERTEX <rid>|<class>|FROM (<subquery>) [WHERE <conditions>] [LIMIT <MaxRecords>>]
```

The WHERE clause is common to the other SQL commands.

History and Compatibility

- 1.1: first version
- starting from v.1.4 the command uses the Blueprints API under the hood, so if you're working in Java using the OGraphDatabase API you could experience in some difference how edges are managed. To force the command to work with the "old" API change the GraphDB settings described in Graph backward compatibility

Examples

Deletes the vertex, and disconnects all vertices pointing towards it:

```
DELETE VERTEX #10:231
```

Deletes all user accounts which are marked with an incoming edge of class BadBehaviorInForum:

```
DELETE VERTEX Account Where in.@Class = 'BadBehaviorInForum'
```

Deletes all those EmailMessages which are marked as spam by isSpam property

```
DELETE VERTEX EMailMessage Where isSpam = true
```

Deletes every vertex of class 'Attachment', which has an edge towards it of class 'HasAttachment', with a property 'date' of condition to be all (HasAttachment edges) which are older than 1990, and secondly, the vertex 'Email' which is connected to class Attachment has a condition on its property 'from' to be 'some...@example.com':

```
DELETE VERTEX Attachment Where in[@Class = 'HasAttachment'].date <= "1990" and in.out[@Class
```

SQL - DROP CLASS

The **Drop Class** command removes a class from the schema. *NOTE: Pay attention to maintain the schema coherent. For example avoid to remove classes that are super classes of others. The associated cluster won't be deleted.*

Syntax

```
DROP CLASS <class>
```

Where:

- **class** is the class of the schema

See also

- create class
- alter class
- alter cluster
- SQL commands
- Console commands

Examples

Remove the class 'Account':

```
DROP CLASS Account
```

SQL - DROP CLUSTER

The **Drop Cluster** command removes a cluster and all its content. This operation cannot be rollbacked.

Syntax

```
DROP CLUSTER <cluster-name>|<cluster-id>
```

Where:

- **cluster-name** is the cluster name as string
- **cluster-id** is the cluster id as integer

See also

- create cluster
- alter cluster
- drop class
- SQL commands
- Console commands

Examples

Remove the cluster 'Account':

```
DROP CLUSTER Account
```

SQL - DROP INDEX

The **Drop Index** command removes an index on a property defined in the schema.

Syntax

```
DROP INDEX <index-name>|<class>.<property>
```

Where:

- **class** is the class of the schema
- **property**, is the property created into the **class**

See also

- SQL Create Index
- Indexes
- SQL commands

Examples

```
DROP INDEX users.Id
```

SQL - DROP PROPERTY

The **Drop Property** command removes a property from the schema. This doesn't remove the property values in records, but just change the schema information. Records will continue to have the property values if any.

Syntax

```
DROP PROPERTY <class>.<property>
```

Where:

- **class** is the class of the schema
- **property**, is the property created into the **class**

Examples

Remove the property 'name' in class 'User':

```
DROP PROPERTY user.name
```

To create a new property use the Create Property command.

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

SQL - EXPLAIN

Profiles any command and returns back result of execution. This is useful to know why a query is slow. Use EXPLAIN as keyword before the command you want to profile.

Syntax

```
EXPLAIN <command>
```

- **command** is the command you want to profile.

Returns a document containing all the profiled metrics:

Metric	Description
elapsed	Time elapsed in seconds to execute the command. The precision is nanosecond
resultType	The result type. Can be 'collection', 'document' or 'number'
resultSize	The number of record retrieved in case the resultType is a 'collection'
recordReads	The number of records read from disk
documentReads	The number of documents read from disk. It could be different by recordReads if other kind of records are present in the target of the command. For example if you put in the same cluster documents and recordbytes you could skip many records. Much better to store different records in separate clusters in case of scan
documentAnalyzedCompatibleClass	The number of documents analyzed of the requested class of the query. It could be different by documentReads if records of different classes are present in the target of the command. For example if you put in the same cluster documents of class "Account" and "Invoice" you could skip many records of type "Invoice" if your're looking for Account instances. Much better to store records of different classes in separate clusters in case of scan
involvedIndexes	The indexes involved in the command
indexReads	The number of records read from the index

Examples

Non indexed query

```
orientdb> explain select from account

Profiled command
'{documentReads:1126,documentReadsCompatibleClass:1126,recordReads:1126,elapsed:209,resultTy
```

Indexed query

```
orientdb> explain select from profile where name = 'Luca'

Profiled command '{involvedIndexes:[1],indexReads:1,documentAnalyzedCompatibleClass:1,elapse
```

To know more about other SQL commands look at SQL commands.

SQL - FIND REFERENCES

SQL command to search all records that contains a link to a given record id in the entire database or a subset of specified class and cluster. Returns a set of record ids.

Syntax

```
FIND REFERENCES <rid|(<sub-query>)> [class-list]
```

Where:

- *rid* is the record id to search. If a sub-query is passed, then all the RIDs returned by the sub-query will be searched. Sub-query is available since 1.0rc9
- *class-list* list of specific class or cluster, separated by commas, you want to execute the search in.

Returns a list of document containing 2 fields:

- rid, as the original RID searched
- referredBy, as a Set of RIDs containing the collection of RID that reference the searched rid if any, otherwise the set is empty

Examples

Get all the records that contains a link to 5:0

```
find references 5:0
```

Result example:

```
RESULT:
+------+-----------------+
| rid  | referredBy      |
+------+-----------------+
| #5:0 | [#10:23, #30:4] |
+------+-----------------+
```

Get all the references to the record of the default cluster (available since 1.0rc9):

```
find references (select from cluster:default)
```

Get all the records in Profile and !AnimalType classes that contains a link to 5:0 :

```
find references 5:0 [Profile,AnimalType]
```

Get all the records in Profile cluster and !AnimalType class that contains a link to 5:0

```
find references 5:0 [cluster:Profile,AnimalType]
```

To know more about other SQL commands look at SQL SQL commands.

SQL - GRANT

The **Grant** command changes the permission of a role granting the access to one or more resources.

Syntax

```
GRANT <permission> ON <resource> TO <role>
```

Where:

- **permission** can be:
  - NONE, no permission
  - CREATE, to create the indicated resource
  - READ, to read the indicated resource
  - UPDATE, to update the indicated resource
  - DELETE, to delete the indicated resource
  - ALL, all permissions

- **resource**, the target resource where to change the permissions
  - database, as the access to the whole database
  - database.class, as the access to the records contained in a class. Use `**` to indicate all the classes
  - database.cluster, as the access to the records contained in a cluster. Use `**` to indicate all the clusters
  - database.query, as the ability to execute query (READ is enought)
  - database.command, as the ability to execute SQL commands. CREATE is for SQL-Insert, READ is for SQL SELECT, UPDATE for SQL-Update and DELETE is for SQL-Delete
  - database.config, as the ability to access to the configuration. Valid permissions are READ and UPDATE
  - database.hook.record, as the ability to set hooks
  - server.admin, as the ability to access to the server resources

- **role**, the role name

Examples

Grant the permission to *update* any records in *cluster Account* to the *role "backoffice"*.

```
GRANT update ON database.cluster.Account TO backoffice
```

To know more about other SQL commands look at SQL commands.

SQL - REBUILD INDEXES

The **Rebuild Index** command rebuilds an automatic index.

Syntax

```
REBUILD INDEX <index-name>
```

Where:

- **index-name** name of the index. Use * to rebuild all the automatic indices

See also

- SQL Create Index
- SQL Drop Index
- Indexes
- SQL commands

Examples

```
REBUILD INDEX Profile.nick
```

```
REBUILD INDEX *
```

SQL - REVOKE

The **Revoke** command change the permission of a role revoking the access to one or more resources.

Syntax

```
REVOKE <permission> ON <resource> FROM <role>
```

Where:

- **permission** can be:
  - NONE, no permission
  - CREATE, to create the indicated resource
  - READ, to read the indicated resource
  - UPDATE, to update the indicated resource
  - DELETE, to delete the indicated resource
  - ALL, all permissions

- **resource**, the target resource where to change the permissions
  - database, as the access to the whole database
  - database.class, as the access to the records contained in a class. Use `*` to indicate all the classes
  - database.cluster, as the access to the records contained in a cluster. Use `*` to indicate all the clusters
  - database.query, as the ability to execute query (READ is enought)
  - database.command, as the ability to execute SQL commands. CREATE is for SQL-Insert, READ is for SQL SELECT, UPDATE for SQL-Update and DELETE is for SQL-Delete
  - database.config, as the ability to access to the configuration. Valid permissions are READ and UPDATE
  - database.hook.record, as the ability to set hooks
  - server.admin, as the ability to access to the server resources

- **role**, the role name

Examples

Revoke the permission to *delete* any records in any *cluster* to the *role "backoffice"*.

```
REVOKE delete ON database.cluster.* TO backoffice
```

To know more about other SQL commands look at SQL commands.

SQL - TRAVERSE

Traverse is a special command that retrieves the connected records crossing the relationships. This command works not only with graph API but at document level. This means you can traverse relationships between invoice and customers without the need to model the domain using the Graph API.

To know more look at Java-Traverse page.

	In many cases SELECT can be used instead of TRAVERSE, resulting in faster and shorter query. Take a look at Should I use TRAVERSE or SELECT?

Syntax

```
TRAVERSE <[class.]field>|*|any()|all()
         [FROM <target>]
         [LET <Assignment>*]
         WHILE <condition>
         [LIMIT <max-records>]
         [STRATEGY <strategy>]
```

- **fields** are the list of fields you want to traverse
- **target** can be a class, one or more clusters, a single RID, a set of RIDs or another command like another TRAVERSE (as recursion) or a SELECT
- **LET** is the part that bind context variables to be used in projections, conditions or sub-queries
- **while** condition to continue the traversing while it's true. Usually it's used to limit the traversing depth by using `$depth` where x is the maximum level of depth you want to reach. **$depth** is the first context variable that reports the depth level during traversal. *NOTE: the old 'where' keyword is deprecated*
- **max-records** sets the maximum result the command can return
- **strategy**, to specify how to traverse the graph

Fields

Are the list of fields you want to traverse. If `*`, any() or all() are specified then all the fields are traversed. This could be costly so to optimize the traverse use the pertinent fields. You can also specify fields at class level. Polymorphism is supported, so by specifying Person.city and Customer class extends Person, you will traverse Customer instances too.

Field names are case-sensitive, classes not.

Target

Target can be:

- **Class** is the class name to browse all the record to be traversed. You can avoid to specify **class:** as prefix
- **Cluster** with the prefix 'cluster:' is the cluster name where to execute the query
- A set of RIDs inside square brackets to specify one or a small set of records. This is useful to navigate graphs starting from some root nodes

- A root record specifying its RID

Context

Traverse command uses the following variables in the context:

- **$parent**, to access to the parent's context if any. This is useful when the Traverse is called in a sub-query
- **$current**, current record iterated. To access to the upper level record in nested queries use $parent.$current
- **$depth**, as the current depth of nesting
- **$depth**, as the current depth of nesting
- **$path**, as the string representation of the current path. Example `#6:0.in.#5:0#.out`. You can also display it with -> `select $path from (traverse ** from V)`
- **$stack**, as the List of operation in the stack. Use it to access to the history of the traversal. It's a List> 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 all the records traversed as a `Set<ORID>`.

Examples

Traverse all the fields of a root record

Assuming #10:1234 is the RID of the record to start traversing:

```
traverse * from #10:1234
```

Social Network domain

In a social-network-like domain a profile is linked to all the friends. Below some commands.

Specify fields and depth level

Assuming #10:1234 is the RID of the record to start traversing get all the friends up to the third level of depth using the BREADTH_FIRST strategy:

```
traverse friends from #10:1234 while $depth <= 3 strategy BREADTH_FIRST
```

In case you want to filter per minimum depth create a predicate in the select. Example like before but excluding the first target vertex (#10:1234):

```
select from ( traverse friends from #10:1234 while $depth <= 3 ) where $depth >= 1
```

NOTE: You can also define the maximum depth in the SELECT clause but it's much more efficient to set it at the inner TRAVERSE statement because the returning record sets are already filtered by depth

Mix with select to have more power

Traverse command can be combined with SQL SELECT statement to filter the result set. Below the same example above but filtering by Rome as city:

```
select from ( traverse friends from #10:1234 while $depth <= 3 ) where city = 'Rome'
```

Another example to extract all the movies of actors that have worked, at least once, in any movie produced by J.J. Abrams:

```
select from (
  traverse Movie.actors, Actor.movies from (
    select from Movie where producer = "J.J. Abrams"
  ) while $depth <= 3
) where @class = 'Movie'
```

Display the current path

To return or use the current path in traversal refer to the **$path** variable:

```
select $path from ( traverse out from V while $depth <= 10 )
```

Should I use TRAVERSE or SELECT?

If traversing information, such as relationship names and depth level, are known at priori, please consider using SELECT instead of TRAVERSE. SELECT is faster on this case. Example:

This query traverses the "follow" relationship of Twitter accounts getting the 2nd level of friendship:

```
SELECT FROM (
  TRAVERSE out('follow') FROM TwitterAccounts WHERE $depth <= 2
) WHERE $depth = 2
```

But can be expressed also with SELECT and it's shorter and faster:

```
SELECT out('follow').out('follow') FROM TwitterAccounts
```

Using TRAVERSE with Graph model and API

Even if the TRAVERSE command can be used with any domain model, the place where is more used is the Graph-Database model.

Following this model all 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". So if you want to traverse in a direction you have to use the class name when declare the traversing fields. Below the directions:

- **OUTGOING**, use `V.out, E.in` because vertices are connected with the "out" field but the edge exits as "in" field.
- **INCOMING**, use `V.in, E.out` because vertices are connected with the "in" field but the edge enters as "out" field.

Example of traversing all the outgoing vertices found starting from the vertex with id #10:3434:

```
traverse V.out, E.in from #10:3434
```

So in a mailing-like domain to find all the messages sent in 1/1/2012 from the user 'Luca' assuming it's stored in the 'User' Vertex class and that messages are contained in the 'Message' Vertex class. Sent messages are stored as "out" connections of Edge class 'SentMessage':

```
select from (
  traverse V.out, E.in from (
    select from User where name = 'Luca'
  ) while $depth <= 2 and (@class = 'Message' || ( @class = 'SentMessage' and sentOn = '01/0
) where @class = 'Message'
```

Operator TRAVERSE

Before the introducing of TRAVERSE command OrientDB has the TRAVERSE operator but worked in the opposite way and it was applied in the WHERE condition.

TRAVERSE operator is deprecated. Please use the TRAVERSE command together with SELECT command to have much more power!

The syntax of the old TRAVERSE operator was:

```
SELECT FROM <target> WHERE <field> TRAVERSE[(<minDeep> [,<maxDeep> [,<fields>]])] (<conditio
```

	WARNING: THIS SYNTAX WILL NOT BE SUPPORTED ANYMORE IN v. 2.1

Where:

- **target** can be one of listed above
- **field** can be:
  - **out**, as the outgoing edges
  - **in**, as the incoming edges
  - **any attribute of the vertex**
  - **any()**, means any of the field considering also **in** and **out**
  - **all()**, means all the fields considering also **in** and **out**
- **minDeep** is the minimum deep level to start to apply the conditions. Usually is 0 for the root vertex or 1 for the just-outgoing vertexes
- **maxDeep**, optionally limits the maximum deep level to reach. -1 means infinite. Default is -1
- **fields**, optionally tells the field list to traverse. Default is any()
- **conditions** are the conditions to check for any traversed vertex. To know more about the query syntax see SQL syntax

Examples

Example of a query that returns all the vertices that have at least one friend (connected with out), up to the 3rd degree, that lives in Rome:

```
select from Profile where any() traverse(0,3) (city = 'Rome')
```

This can be rewritten using the most power TRAVERSE command:

```
select from Profile
let $temp = (
  select from (
    traverse * from $current while $depth <= 3
  )
  where city = 'Rome'
)
where $temp.size() > 0
```

Examples Of Graph Query.

Vertex edge Vertex User----->Friends----->User Label='f'

Query to Find the first level friends of User Whose record Id is #10:11

```
select distinct(in.lid) as lid,distinct(in.fid) as fid   from (traverse V.out, E.in from #10
```

2nd level friends of a user, to find that we have to just change the depth to 3

```
SELECT distinct(in.lid) as lid, distinct(in.fid) as fid FROM (
  TRAVERSE V.out, E.in FROM #10:11 WHILE $depth <=3
) WHERE @class='Friends'
```

To know more about other SQL commands look at SQL commands.

SQL - TRUNCATE CLASS

The **Truncate Class** command deletes the records of all the clusters defined as part of the class. By default every class has one cluster associated with the same name. This command acts at lower level then SQL Delete Command.

Syntax

```
TRUNCATE CLASS <class-name>
```

Where:

- **class-name** is the name of the class

Examples

Remove all the record of class "Profile":

```
TRUNCATE CLASS Profile
```

See also SQL Delete Command and SQL Truncate Cluster Command. To create a new class use the Create Class command.

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

SQL - TRUNCATE CLUSTER

The **Truncate Cluster** command deletes all the records of a cluster. This command acts at lower level then SQL Delete Command.

Syntax

```
TRUNCATE CLUSTER <cluster-name>
```

Where:

- **cluster-name** is the name of the cluster

Examples

Remove all the records in the cluster "Profile":

```
TRUNCATE CLUSTER Profile
```

See also SQL Delete Command and SQL Truncate Class commands.

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

SQL - TRUNCATE RECORD

The **Truncate Record** command truncates a record without loading it. Useful when the record is dirty in any way and can't be loaded correctly.

Syntax

```
TRUNCATE RECORD <rid>*
```

Where:

- **rid** RecordID to truncate. To truncate multiple records in one shot, list all the RecordIDs separated by comma inside squared brackets.

Returns

The number of records truncated.

Examples

Truncates the record **#20:3**:

```
TRUNCATE RECORD 20:3
```

Truncates 3 records all together:

```
TRUNCATE RECORD [20:0, 20:1, 20:2]
```

See also SQL Delete Command.

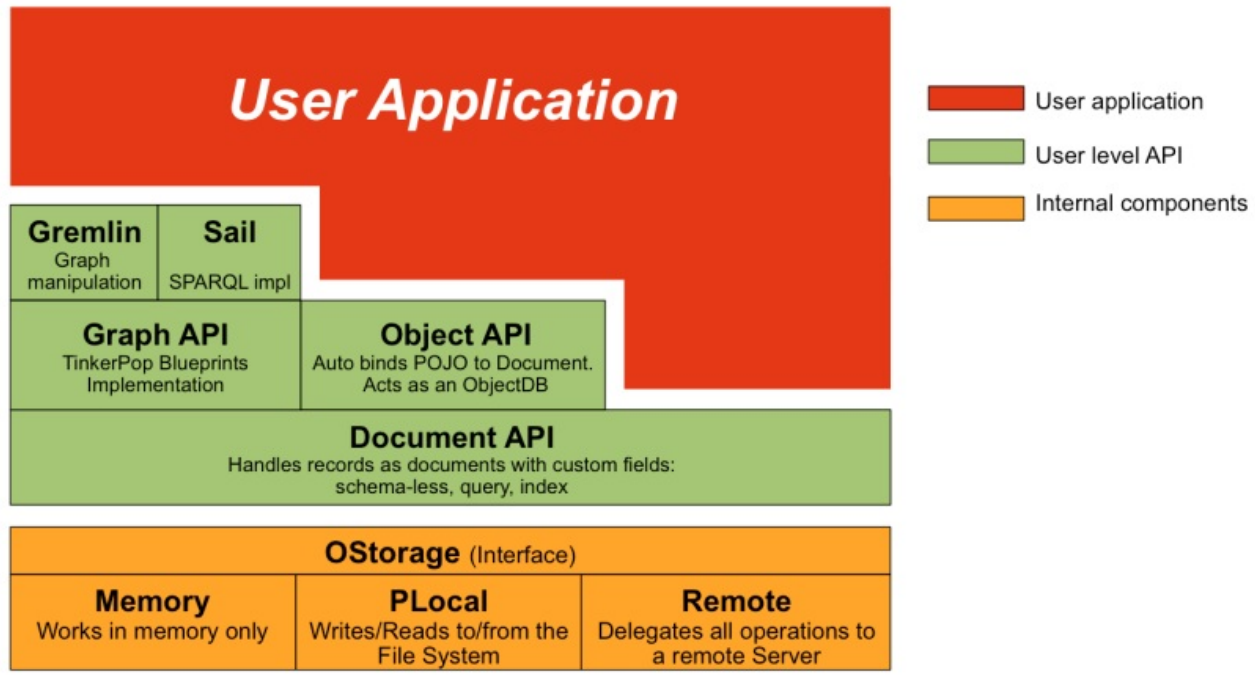
To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Java API

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

Architecture of 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 1.5. Please consider using Document or Graph API by writing an additional layer of mapping with your POJO.

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`	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`

Graph API

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

```
orient-commons-*.jar
orientdb-core-*.jar
blueprints-core-*.jar
orientdb-graphdb-*.jar
(blueprints-orient-graph-*.jar only for OrientDB < v1.7)
```

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.

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. In multi-threaded applications use one OrientGraph instance per thread.

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(1,1

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

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

Transactions

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.

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.

NOTE: To work against a graph always use transactional OrientGraph instances and never non-transactional ones to avoid graph corruption from multi-threaded changes.

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());
```

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: Starting from OrientDB v1.4.x (until 2.0, where the opposite is true) edges by default are stored as links not as records (i.e. useLightweightEdges=true by default). 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-Edge 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: https://github.com/orientechnologies/orientdb/wiki/Troubleshooting#why-i-cant-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", "Victor
```

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 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("c
```

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.

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	true

	fields and makes faster traversal by edge's label (class).	
blueprints.orientdb.lightweightEdges	Uses lightweight edges. This avoids creating a physical document per edge. Documents are created only when the Edges have properties.	true
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.

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));
```

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.

Transactions

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
        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();
        break;
    } catch( OTransactionException e ) {
        // SOMEONE HAVE UPDATE THE INVOICE VERTEX AT THE SAME TIME, RETRY IT
    }
}
```

Auto-retry

Starting with v.1.5, transactions are automaticaly retried if a timeout exception occurs. This happens in case of deadlocks or network latency. By default the AutoRetry setting is 10, but you can change it or disable it by setting it to 0, by calling:

```
((OTransactionOptimistic) graph.getRawGraph().getTransaction()).setAutoRetries( 0 );
```

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('bough') ) from Customer where name = 'Jay'"
    System.out.println("- Bought: " + v);
}
```

To execute an asynchronous query:

```
graph.command(
          new OSQLAsynchQuery<Vertex>("select from Member",
            new OCommandResultListener() {
              int resultCount =0;
              @Override
              public boolean result(Object iRecord) {
                resultCount++;
                Vertex doc = graph.getVertex( iRecord );
               return resultCount < 100;
              }
            } ).execute();
```

SQL commands

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').nam
```

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!');
```

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", "amigar
```

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.*

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);
```

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).setI
profile.createIndex("nickIdx", OClass.INDEX_TYPE.UNIQUE, "nick"); // Creates unique constrai

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("cla
```

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 major
- TinkerPop Blueprints 2.2.0 or major.

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:

```
orientdb> alter class V superclass orestricted

Class updated successfully

orientdb> 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:

```
orientdb> select from orole

---+---------+--------------------+--------------------+--------------------+--------------
  #| RID     |name                |mode                |rules               |inheritedRole
---+---------+--------------------+--------------------+--------------------+--------------
  0|     #4:0|admin               |1                   |{}                  |null
  1|     #4:1|reader              |0                   |{database=2, database.schema=2, data
  2|     #4:2|writer              |0                   |{database=2, database.schema=7, data
---+---------+--------------------+--------------------+--------------------+--------------

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):

```
orientdb> insert into ouser set name = 'luca', status = 'ACTIVE', password = 'luca', roles =

Inserted record 'OUser#5:4{name:luca,password:{SHA-256}D70F47790F689414789EEFF231703429C7F88

orientdb> insert into ouser set name = 'steve', status = 'ACTIVE', password = 'steve', roles

Inserted record 'OUser#5:3{name:steve,password:{SHA-256}F148389D080CFE85952998A8A367E2F7EAF3
```

Create a simple graph as user 'Luca'

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

```
orientdb> disconnect

Disconnecting from the database [blog]...OK

orientdb> 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:

```
orientdb> create vertex set label = 'food', name = 'Pizza'

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

orientdb> 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":

```
orientdb> 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:

```
orientdb> 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:

```
orientdb> disconnect

Disconnecting from the database [blog]...OK

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

orientdb> select from v

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

Ok, no vertices found. Try to create something:

```
orientdb> create vertex set label = 'car', name = 'Ferrari Modena'

Created vertex 'V#9:2{label:car,name:Ferrari Modena,_allow:[1]} v0' in 0,000000 sec(s).

orientdb> create vertex set label = 'driver', name = 'steve'

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

orientdb> 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:

```
orientdb> 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?

```
orientdb> create edge from #9:2 to #9:0 set label = 'security-test'

Error: com.orientechnologies.orient.core.exception.OCommandExecutionException: Error on exec
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:

```
orientdb> gremlin g.V

[v[#9:2], v[#9:3]]

Script executed in 0,448000 sec(s).
orientdb> 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
Release	1.0-SNAPSHOT	1.7M03	4.5.1
Product Web Site	http://www.orientdb.org	http://www.neo4j.org	http://www.spa technologies.c
License	Open Source Apache 2	Open Source GPL, Open Source AGPL and Commercial	Commercial
Query languages	Extended SQL, Gremlin	Cypher Gremlin	Not available, c via API
Transaction support	 ACID	 ACID	
Protocols	Embedded via Java API, remote as Binary and REST	Embedded via Java API and remote via REST	?
Replication	Multi-Master	Master-Slave	No
Custom types	 Supports custom types and 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
Release	1.0-SNAPSHOT	1.7M03
Product Web Site	http://www.orientdb.org	http://www.neo4j.org
Implementation details	OrientDB impl	Neo4j impl
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
Release	1.4	1.9.M05
Product Web Site	http://www.orientdb.org	http://www.neo4j.org
VertexTestSuite	**1,524.06**	1,595.27
EdgeTestSuite	**1,252.21**	1,253.73
GraphTestSuite	**1,664.75**	2,400.34
QueryTestSuite	306.58	**188.52**
IndexableGraphTestSuite	4,620.61	11,299.02
IndexTestSuite	**2,072.23**	5,239.92
TransactionalGraphTestSuite	**1,573.93**	3,579.50
KeyIndexableGraphTestSuite	**571.42**	845.84
GMLReaderTestSuite	778.08	**682.83**
GraphMLReaderTestSuite	**814.38**	864.70
GraphSONReaderTestSuite	**424.77**	480.81

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** from v1.4. **Lightweight Edges** are like regular edges, but they have no identity on database. Lightweight edges can be used only when:

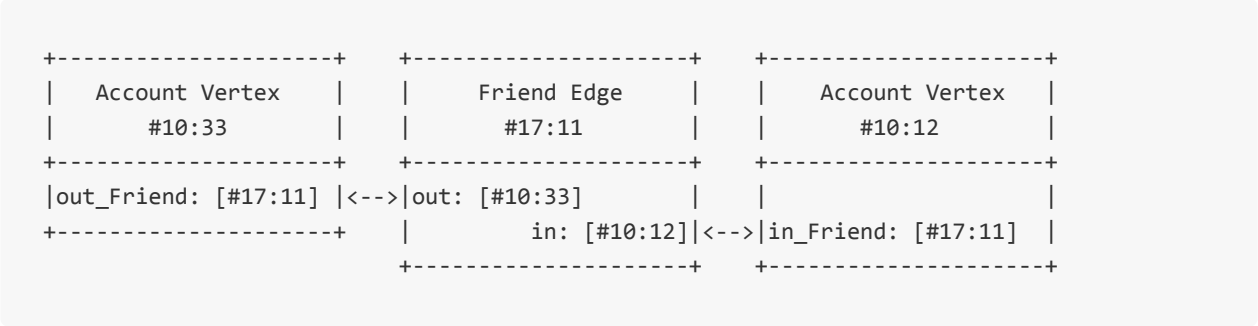
- no properties are defined on edge
- two vertices are connected by maximum 1 edge, so if you already have one edge between two vertices and you're creating a new edge between the same vertices, the second edge will be regular

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.



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:

```
orient-commons-*.jar
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", "

// 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",
```

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",
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.

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 n
List<ODocument> result = database.command(query).execute("Barack", "Obama");
```

Example of named parameters:

```
OSQLSynchQuery<ODocument> query = new OSQLSynchQuery<ODocument>("select from Profile where n
  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 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 referenced relationships

1-1 and N-1 referenced 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 referenced relationships

1-N and N-M referenced 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).setN
profile.createIndex("nickIdx", OClass.INDEX_TYPE.UNIQUE, "nick"); // Creates unique constrai

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 not updated 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	Co
Web Site	http://www.orientdb.org	http://www.mongodb.org	http://www
Supported models	Document and Graph	Document	Document
Transactions	Yes, ACID	No	Yes, ACID
Query languages	Extended SQL, Gremlin	Mongo Query Language	Non suppo

Object API

Note: The object database has been refactored since the release 1.0. If you use the previous one look at: Old Implementation ODatabaseObjectTx.

Requirements

To use the Object APi include the following jars in your classpath:

```
orient-commons-*.jar
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", "adm

// 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", "admi

// 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
OObjectDatabase db1 = new OObjectDatabaseTx("memory:petshop").create();

// OPEN A REMOTE DATABASE
OObjectDatabase 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:

```
OObjectDatabase 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 to store records. To work in schema-full mode take a look to the Schema APIs page.

Create a new object

The best practice to create a Java object is to use the OObjectDatabase.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
}

OObjectDatabase 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);
animal.setName( "Antoni" );
animal.setSurname( "Gaudi" );
db.save( person );

// CREATES A NEW PERSON FROM A PARAMETRIZED CONSTRUCTOR
Person person = db.newInstance(Person.class,  "Antoni");
animal.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.
  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);
```

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 OObjectDatabase 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 clas
db.getMetadata().getSchema().generateSchema("com.mycompany.myapp.mydomainpackage");  // Gene
```

Version 1.5

```
db.generateSchema(Foo.class); // Generates the schema for Foo class
db.generateSchema("com.mycompany.myapp.mydomainpackage"); // Generates the schema for all cl
```

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
db.getEntityManager().registerEntityClasses("com.mycompany.myapp.mydomainpackage"); // Gener
```

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", "adm

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", "admi

...

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.cl
```

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 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 ODatabaseObjectTx 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 USE
                     // 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 USE
                  // UNLESS IT'S NEEDED

  @Version
  private Object version; // DON'T CREATE GETTER/SETTER FOR IT TO PREVENT THE CHANGING BY TH
                       // 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 are managed. The standard SQL language is not enough to work with tree or graphs because it hasn't the concept of recursion. This is the reason why OrientDB provide a new command to traverse trees and graphs: TRAVERSE. Traversing is the operation that cross 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:

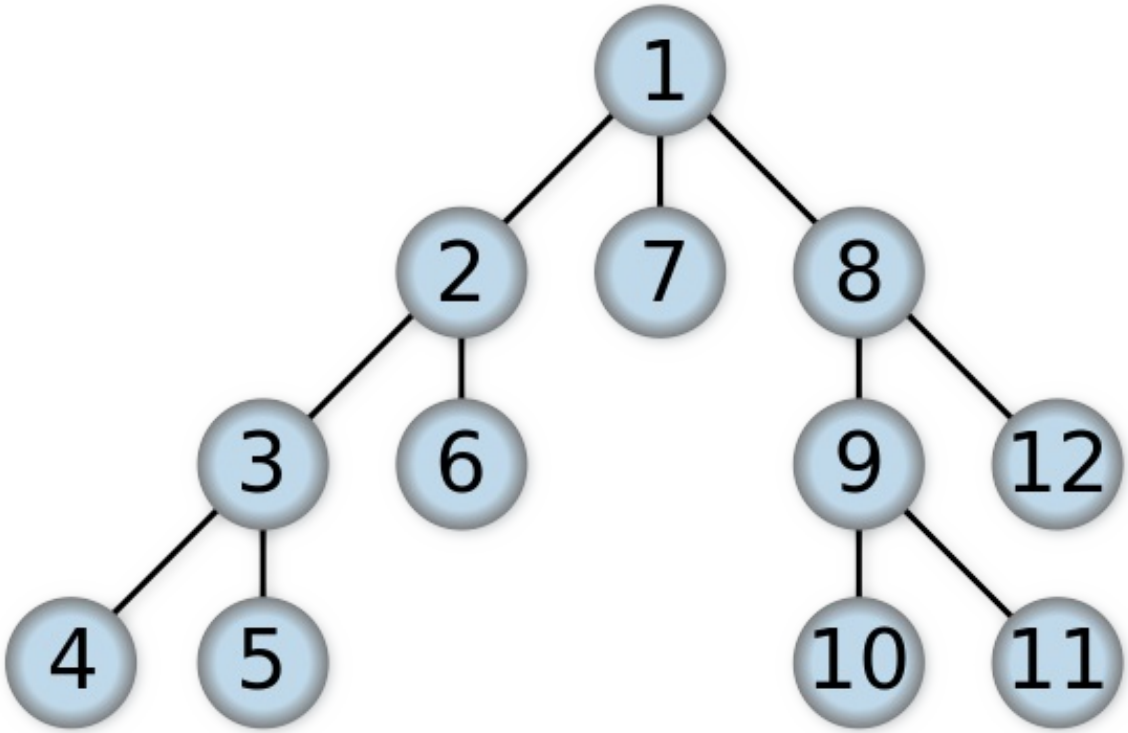
- **target**, as the starting point where to traverse records. Can be:
  - **class**
  - **cluster**
  - **set of records**, specifying its RecordID
  - **sub-command** that returns an `Iterable<OIdentifiable>`. You can nest multiple select and traverse all together

- **fields**, the fields to traverse. Use `*`, `any()` or `all()` to traverse all the fields of a document
- **limit**, the maximum number of records to retrieve
- **predicate**, as the predicate to execute against each document traversed. If the predicate returns true, then the document is returned, otherwise is skipped
- **strategy**, as the way the traverse go in deep:
  - *DEPTH_FIRST*, the default,
  - *BREADTH_FIRST*,

Traversing strategies

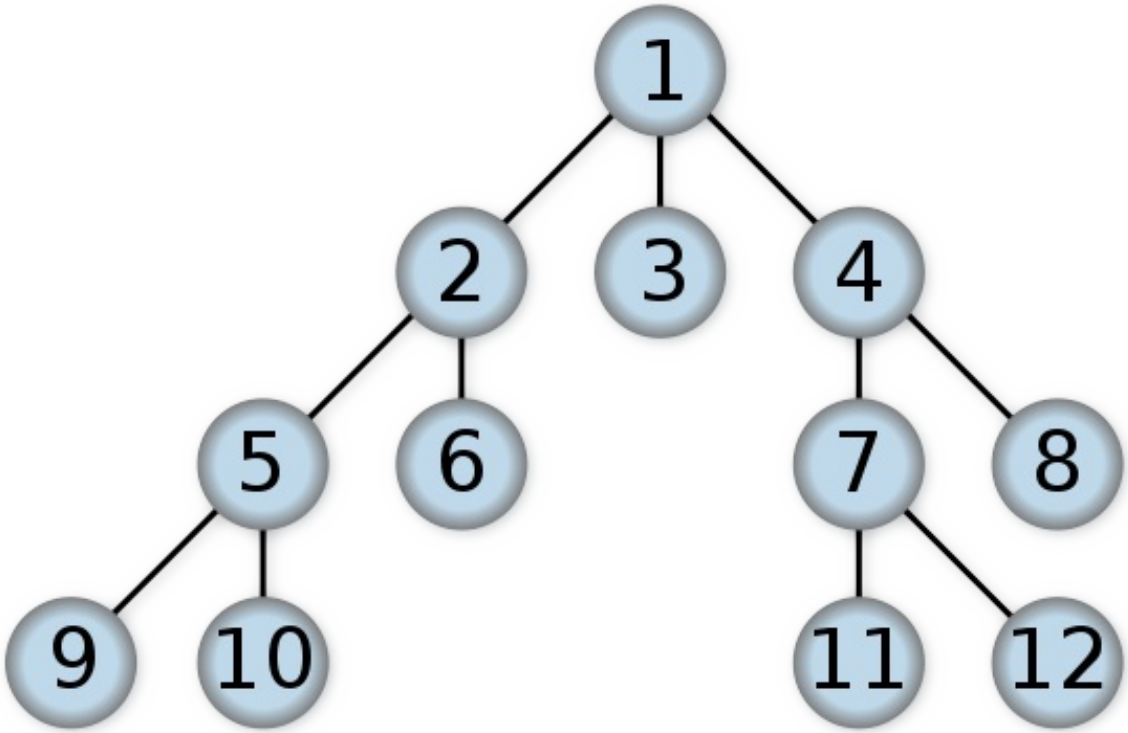
DEPTH_FIRST strategy

This is the default strategy used by OrientDB 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 *BREADTH_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 *Depth-First* strategy:



Context variables

During traversing 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

Below how to traverse using different approaches.

SQL Traverse

The simpler available way to execute a traversal is using the SQL Traverse command. Example to retrieve all the 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 $de
  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 boolean evaluate(ORecord<?> iRecord, 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);
}
```

Multi-Threading

OrientDB supports multi-threads access to the database. `ODatabase` *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*.

Multiple database instances points to the same storage by using the same URL. In this case Storage is thread-save and orchestrates requests from different `ODatabase*` instances.

```
ODatabaseDocument1------+
                        +----> OStorageLocal (url=local:/temp/db)
ODatabaseDocument2------+
```

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.save(rec1); // force saving in database1 no matter where the record came from

rec2.field("name", "Luke");
database2.save(rec2); // force saving in database2 no matter where the record came from
```

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:

```
ODatabaseRecordThreadLocal.INSTANCE.set( database );
```

Where database is the current database instance. Example:

```
ODocument rec1 = database1.newInstance();
ODocument rec2 = database2.newInstance();

ODatabaseRecordThreadLocal.INSTANCE.set( database1 );
rec1.field("name", "Luca");
rec1.save();

ODatabaseRecordThreadLocal.INSTANCE.set( database2 );
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().getUnd
  }
}
```

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 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
ODatabaseRecordThreadLocal.INSTANCE.set( db1 );
...
```

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.

To fix this problem you can:

- if your JVM is the only client is writing to the database then disabling the Level1 cache could be enough: http://code.google.com/p/orient/wiki/Caching#Level_1
- disable MVCC by setting the {db.mvcc} parameter to false: `java -Ddb.mvcc=false`
- If you're using the GraphDB api look at: concurrency

If you want to leave the MVCC and write code concurrency proof:

```
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 CURREN

ODocument doc2 = new ODocument("Provider");
doc2.field("name", "Chuck");
doc2.save(); // THIS IS BOUND TO DB2 BECAUSE IT'S THE CURRENT ONE

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

```
package com.orientechnologies.test;
import javax.servlet.*;

public class Example extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException
  {
    ODatabaseDocument database = ODatabaseDocumentPool.global().acquire("local:/temp/db", "a

    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 database from the pool and binds it in current ThreadLocal before to execute the Servlet code. Once returned the ThreadLocal is cleared and database released to the pool.

JaveEE Servlets

Create a Filter class

Create a database instance per request

In this example a new database 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) {
      ODatabaseDocument database = new ODatabaseDocumentTx("local:/temp/db").open("admin", "
      try{
        chain.doFilter(request, response);
      } finally {
        database.close();
      }
  }
}
```

Use the database pool

In this example the database pool is used.

```
package com.orientechnologies.test;
import javax.servlet.*;

public class OrientDBFilter implements Filter {

  public void doFilter(ServletRequest request, ServletResponse response,
          FilterChain chain) {
      ODatabaseDocument database = ODatabaseDocumentPool.global().acquire("local:/temp/db",
      try{
        chain.doFilter(request, response);
      } finally {
        database.close();
      }
  }

  public void destroy() {
      ODatabaseDocumentPool.global().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>
```

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
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
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
serverAdmin.existsDatabase("local");
```

JDBC Driver

OrientDB is a NoSQL DBMS that support a subset of SQL ad query language.

Include in your projects

```
<dependency>
  <groupId>com.orientechnologies</groupId>
  <artifactId>orientdb-jdbc</artifactId>
  <version>1.7</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:loc
```

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:loc
```

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/XMLSche
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/pe
    <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.reg
            <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", "adm

// 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.

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 **bin** directory:

```
> gremlin.bat

         \,,,/
         (o o)
-----oOOo-(_)-oOOo-----
```

Open the graph database

Before to play with Gremlin you need a valid **OrientGraph** instance that points to a OrientDB database. To know all the database types look at Storage types.

When you're working with a local or memory database if the database not exists it's created automatically. Using the remote connection you need to create the database on the target server before to use 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("local:/home/gremlin/db/demo");
==>orientgraph[local:/home/gremlin/db/demo]
```

Some useful links:

- All Gremlin methods
- All available steps

Working with local database

This is the most used mode. The console opens and locks the database for exclusive use. Doesn't require to start a OrientDB Server.

```
gremlin> g = new OrientGraph("local:/home/gremlin/db/demo");
==>orientgraph[local:/home/gremlin/db/demo]
```

Working with remote database

Open a database on a remote server. Assure the server is up and running. 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 cluster configuration (the database life is assured by the cluster itself) or just for test.

```
gremlin> g = new OrientGraph("memory:demo");
==>orientgraph[memory:demo]
```

Use the security

OrientDB supports the security by creating multiple users and roles to associate privileges. To know more look at Security. To open the graph database with a different user than default pass 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 the unique id will be displayed as 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 2 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. For saving 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 argument (with or without the prefix '#'). This example retrieves the first vertex created in the upon 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 it's very similar to [use the *e(id)* method passing the Concepts#RecordId RecordId as argument (with or without the prefix '#'). This example retrieves the first edge created in the upon 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 on traversal. Once you have a graph loaded in your database you can traverse it in many ways.

Basic Traversal

To display all the outgoing edges of the first vertex just created postpone the **.outE** at the vertex. Example:

```
gremlin> v1.outE
==>e[#6:0][#5:0-friend->#5:1]
```

And to display all the incoming edges of the second vertex created in the previous examples postpone 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 of 5:0 and the goes up to 5:1 where is the incoming edge.

For more information look at the Basic Traversal with Gremlin.

Filter results

This examples returns all the outgoing edges of all the vertices with label equals 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 to concatenate expressions to create more complex traversal in a single line:

```
v1.outE.inV
```

Of course this could be much more complex. Below an examples 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+=m
')")).execute(params);
```

GremlinPipeline

You can also use native Java GremlinPipeline like:

```
new GremlinPipeline(g.getVertex(1)).out("knows").property("name").filter(new PipeFunction<St
  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+=m
')")).execute(params);
```

There are more possibilities to define the output in 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 learned how to use Gremlin on top of OrientDB the best place to go in deep with this powerful language is the Gremlin WiKi.

Javascript

Starting from version 1.0, OrientDB supports server side scripting. All the JVM languages are supported. By default JavaScript is installed.

Scripts can be executed client and server side. For the client side the user must have the privilege of READ against `database.command` resource. For the server side the server must enable the scripting interpreter that are disabled by default for security reason.

See also

- SQL-batch

Usage

Via Java API

Execute a command like SQL but using the class `OCommandScript` passing 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 at client-side, the console, or server-side:

- Use `js` to execute the script at the **client-side** running it in the console
- use `jss` to execute the script at the **server-side**. This feature is disabled by default. To enable it look at Enable Server side scripting.

Since the semicolon `;` character is used in both console and JavaScript language to separate statements, how can we execute multiple commands in the console and JavaScript?

OrientDB console uses a reserved keyword `end` to switch from the JavaScript mode to the console mode.

Example:

```
orientdb> connect remote:localhost/demo admin admin; js for( i = 0; i < 10; i++ ){ db.query(
```

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 an example to display the result of a query server and client side.

1. connects to the remote server as `admin`
2. Execute a query and assign the result to the variable `r`, then display it server side and return it to be displayed client side too
3. Exit 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}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8
  1|     #4:1|reader              |{SHA-256}3D0941964AA3EBDCB00CCEF58B1BB399F9F898465E9886D5
  2|     #4:2|writer              |{SHA-256}B93006774CBDD4B299389A03AC3D88C3A76B460D538795BC
---+---------+--------------------+--------------------+--------------------+--------------
Script executed in 0,073000 sec(s). Returned 3 records

orientdb> exit
```

Batch mode

The same example above is execute in batch mode:

```
$ ./console.sh "connect remote:localhost/demo admin admin;jss var r = db.query('select from
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}8C6976E5B5410415BDE908BD4DEE15DFB167A9C873FC4BB8
  1|     #4:1|reader              |{SHA-256}3D0941964AA3EBDCB00CCEF58B1BB399F9F898465E9886D5
  2|     #4:2|writer              |{SHA-256}B93006774CBDD4B299389A03AC3D88C3A76B460D538795BC
---+---------+--------------------+--------------------+--------------------+--------------
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 ("tes
```

Create documents using wrapped Java API

```
orientdb> js new com.orientechnologies.orient.core.record.impl.ODocument('Profile').field('n

Client side script executed in 0,426000 sec(s). Value returned is: Profile#11:52{name:Luca}
```

Enable Server side scripting

For security reason server-side scripting is disabled by default on server. To enable it change the enable field to `true` in **orientdb-server-config.xml** file:

```
<!-- SERVER SIDE SCRIPT INTERPRETER. WARNING! THIS CAN BE A SECURITY HOLE: ENABLE IT ONLY IF
  <handler class="com.orientechnologies.orient.server.handler.OServerSideScriptInterpreter">
    <parameters>
      <parameter name="enabled" value="true" />
    </parameters>
  </handler>
```

NOTE: this will allow to 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 t
} else {
  commandResult = database.executeCommand('update Address set street = \'Via test 1\' where
}
database.close();
```

API

ODatabase object

ODatabase object requires server URL and database name:

Syntax: `new ODatabase(http://<host>:<port>/<databaseName>)`

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", "file
  ],

"clusters": [
    {"id": 0, "name": "internal", "type": "PHYSICAL", "records": 4, "size": 1048576, "filled
],

"txSegment": [
    {"totalLogs": 0, "size": 1000000, "filled": 0, "maxSize": "50mb", "file": "${STORAGE_PAT
  ], "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,
          }, {
          "name": "database.cluster.orole", "create": false, "read": true, "update": false, "d
          }, {
          "name": "database.cluster.ouser", "create": false, "read": true, "update": false, "d
          }, {
          "name": "database.class.*", "create": false, "read": true, "update": false, "delete"
          }, {
          "name": "database.cluster.*", "create": false, "read": true, "update": false, "delet
          }, {
          "name": "database.query", "create": false, "read": true, "update": false, "delete":
          }, {
          "name": "database.command", "create": false, "read": true, "update": false, "delete"
          }, {
          "name": "database.hook.record", "create": false, "read": true, "update": false, "del
          }]
      },
  ],

    "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\'',n
```

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\'',n
```

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\'',n
```

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
```

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
```

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,
        }, {
        "name": "database.cluster.orole", "create": false, "read": true, "update": false, "d
        }, {
        "name": "database.cluster.ouser", "create": false, "read": true, "update": false, "d
        }, {
        "name": "database.class.*", "create": false, "read": true, "update": false, "delete"
        }, {
        "name": "database.cluster.*", "create": false, "read": true, "update": false, "delet
        }, {
        "name": "database.query", "create": false, "read": true, "update": false, "delete":
        }, {
        "name": "database.command", "create": false, "read": true, "update": false, "delete"
        }, {
        "name": "database.hook.record", "create": false, "read": true, "update": false, "del
        }]
    }
  ]
}
```


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+XXX
```

This will put the result of the query `select from XXXX</code>` into the `<code>datajson` 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.

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

```
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: " +
}

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:
}
```

test.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.core.db.`object`.{ODatabaseObject, ODatabaseObjectPool, (

object Test {
    implicit def dbWrapper(db: OObjectDatabaseTx) = new {
        def queryBySql[T](sql:-String,-params:-AnyRef*.md): List[T] = {
            val params4java = params.toArray
            val results: java.util.List[T] = db.query(new OSQLSynchQuery[T](sql.md), params4
            results.asScala.toList
        }
    }

    def main(args: Array[String]) = {
        // ~~~~~~~~~~~~~~ create db ~~~~~~~~~~~~~~~~~~~
        var uri: String = "local: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".md)
        for (user <- users) {
            println(" - user: " + user)
        }

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

        // query by id
        val userById = db.queryBySql[User]("select-from-User-where-@rid-=-?",-new-ORecordId(
        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-(-ci
        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	**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 major length, 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

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": [
  {
    "id": 0,
    "name": "ORole",
    "clusters": [3],
    "defaultCluster": 3, "records": 0},
...
```

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>]]]` content: `<command-text>`

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.

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`

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: Number of operations executed.

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
}
```

DELETE - Document

Delete a document.

Syntax: `http://<server>:[<port>]/document/<database>/<record-id>`

Example

HTTP GET 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 major length, 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",
...
```

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 /connections/[<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 2.0-SNAPSHOT: **28**. 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 managment is implemented in two different way, one stateful another stateless this is choosed in the open/connect operation with a flag, the stateful is based on a Session-id the stateless is based on a Token

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.

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 mement 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)` . 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)`	
			Used for multiple text messages.	

strings	4	N	The format is: `(length:int)[(Nth-string:string)]`	`00020005Hello0007W`

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	
Server *(CONNECT Operations)*		
REQUEST_SHUTDOWN	1	Shut down s
REQUEST_CONNECT	2	Required init server comm
REQUEST_DB_OPEN	3	**Required ini** to the databa
REQUEST_DB_CREATE	4	Add a new d
REQUEST_DB_EXIST	6	Check if data
REQUEST_DB_DROP	7	Delete datab
REQUEST_CONFIG_GET	70	Get a configu
REQUEST_CONFIG_SET	71	Set a configu
REQUEST_CONFIG_LIST	72	Get a list of c
REQUEST_DB_LIST	74	Get a list of c
Database *(DB_OPEN Operations)*		
REQUEST_DB_CLOSE	5	Close a data
REQUEST_DB_SIZE	8	Get the size
REQUEST_DB_COUNTRECORDS	9	Get total nun database.
REQUEST_DATACLUSTER_ADD	10	Add a data c
REQUEST_DATACLUSTER_DROP	11	Delete a data
REQUEST_DATACLUSTER_COUNT	12	Get the total
REQUEST_DATACLUSTER_DATARANGE	13	Get the data
REQUEST_DATACLUSTER_COPY	14	Copy a data
REQUEST_DATACLUSTER_LH_CLUSTER_IS_USED	16	
REQUEST_RECORD_METADATA	29	Get metadat

REQUEST_RECORD_LOAD	30	Load a recor
REQUEST_RECORD_CREATE	31	Add a record
REQUEST_RECORD_UPDATE	32	
REQUEST_RECORD_DELETE	33	Delete a rec
REQUEST_RECORD_COPY	34	Copy a recor
REQUEST_RECORD_CLEAN_OUT	38	Clean out re
REQUEST_POSITIONS_FLOOR	39	Get the last r
REQUEST_COUNT *(DEPRECATED)*	40	See REQUEST_I
REQUEST_COMMAND	41	Execute a co
REQUEST_POSITIONS_CEILING	42	Get the first r
REQUEST_TX_COMMIT	60	Commit trans
REQUEST_DB_RELOAD	73	Reload datab
REQUEST_PUSH_RECORD	79	
REQUEST_PUSH_DISTRIB_CONFIG	80	
REQUEST_DB_COPY	90	
REQUEST_REPLICATION	91	
REQUEST_CLUSTER	92	
REQUEST_DB_TRANSFER	93	
REQUEST_DB_FREEZE	94	
REQUEST_DB_RELEASE	95	
REQUEST_DATACLUSTER_FREEZE	96	
REQUEST_DATACLUSTER_RELEASE	97	
REQUEST_CREATE_SBTREE_BONSAI	110	Creates an s remote serve
REQUEST_SBTREE_BONSAI_GET	111	Get value by
REQUEST_SBTREE_BONSAI_FIRST_KEY	112	Get first key
REQUEST_SBTREE_BONSAI_GET_ENTRIES_MAJOR	113	Gets the port specified one the specified
REQUEST_RIDBAG_GET_SIZE	114	Rid-bag spec does not sav Retrieves co

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

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;
```

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 'd
```

Example of 2 depth-levels exception:

```
(1)(com.orientechnologies.orient.core.exception.OStorageException)(Can't open the storage 'd
```

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. It returns the session id of the client.

```
Request: (driver-name:string)(driver-version:string)(protocol-version:short)(client-id:strin
Response: (session-id:int)(token:bytes)
```

Where:
request content:

- client's **driver-name** as string. Example: "OrientDB Java client"
- client's **driver-version** as string. Example: "1.0rc8-SNAPSHOT"
- client's **protocol-version** as short. Example: 7
- client's **client-id** as string. Can be null for clients. In clustered configuration is 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** as boolean, true if the client want to use a token based session otherwise false
- **user-name** as string. Example: "root"
- **user-password** as string. Example: "kdsjkkasjad" Typically the credentials are those of the OrientDB server administrator. This is not the same as the *admin* user for individual databases. It returns the Session-Id to being reused for all the next calls.

response content:

- **session-id** the new session id or a match id in case of token auth
- **token:bytes** the token bytes or empty(size = 0) if the client send token-session=false or the server not support the token based session

REQUEST_DB_OPEN

This is the first operation the client should call. It opens a database on the remote OrientDB Server. Returns the Session-Id to being reused for all the next calls and the list of configured clusters.

```
Request: (driver-name:string)(driver-version:string)(protocol-version:short)(client-id:strin
Response: (session-id:int)(token:bytes)(num-of-clusters:short)[(cluster-name:string)(cluster
```

Where: request detail:

- client's **driver-name** as string. Example: "OrientDB Java client"
- client's **driver-version** as string. Example: "1.0rc8-SNAPSHOT"
- client's **protocol-version** as short. Example: 7
- client's **client-id** as string. Can be null for clients. In clustered configuration is 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** as boolean, true if the client want to use a token based session otherwise false
- **database-name** as string. Example: "demo"
- **database-type** as string, can be 'document' or 'graph' (since version 8). Example: "document"
- **user-name** as string. Example: "admin"
- **user-password** as string. Example: "admin"
- **cluster-config** is always null unless you're running in a server clustered configuration.
- **orientdb-release** as string. Contains version of OrientDB release deployed on server and optionally build number. Example: "1.4.0-SNAPSHOT (build 13)"

response detail :

- **session-id** the new session id or a match id in case of token auth
- **token:bytes** the token bytes or empty(size = 0) if the client send token-session=false or the server not support the token based session
- **num-of-clusters:short** the size of cluster definition array composed by **cluster-name:string** and **cluster-id:short** and **cluster-config**
- **orientdb-release** the decription of the token release

REQUEST_DB_CREATE

Creates a database in the remote OrientDB server instance

```
Request: (database-name:string)(database-type:string)(storage-type:string)
Response: empty
```

Where:

- **database-name** as string. Example: "demo"
- **database-type** as string, can be 'document' or 'graph' (since version 8). Example: "document"
- **storage-type** can be one of the supported types:
- plocal, as a persistent database
- memory, as a volatile database

NB. 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 return is expected. The 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. It returns true (non-zero) or false (zero).

```
Request: (database-name:string) <-- before 1.0rc1 this was empty (server-storage-type:string
Response: (result:byte)
```

Where:

- **server-storage-type** can be one of the supported types:
- plocal as a persistent database
- memory, as a volatile database

REQUEST_DB_RELOAD

Reloads database information. Available since 1.0rc4.

```
Request: empty
Response:(num-of-clusters:short)[(cluster-name:string)(cluster-id:short)]
```

REQUEST_DB_DROP

Removes a database from the OrientDB Server instance. It returns nothing if the database has been deleted or throws a OStorageException if the database doesn't exists.

```
Request: (database-name:string)(server-storage-type:string - since 1.5-snapshot)
Response: empty
```

Where:

- **server-storage-type** can be one of the supported types:
- plocal as a persistent database
- memory, as a volatile database

REQUEST_DB_SIZE

Asks for the size of a database in the OrientDB Server instance.

```
Request: empty
Response: (size:long)
```

REQUEST_DB_COUNTRECORDS

Asks for the number of records in a database in the OrientDB Server instance.

```
Request: empty
Response: (count:long)
```

REQUEST_DATACLUSTER_ADD

Add a new data cluster.

```
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.

```
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.

```
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.

```
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

Load a record by RecordID, according to a fetch plan

```
Request: (cluster-id:short)(cluster-position:long)(fetch-plan:string)(ignore-cache:byte)(loa
Response: [(payload-status:byte)[(record-type:byte)(record-version:int)(record-content:bytes
```

Where:

- **fetch-plan**, the fetch plan to use or an empty string
- **ignore-cache**, tells if the cache must be ignored: 1 = ignore the cache, 0 = not ignore. since protocol v.9 (introduced in release 1.0rc9)
- **load-tombstones**, the flag which indicates whether information about deleted record should be loaded. The flag is applied only to autosharded storage and ignored otherwise.
- **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** is
- 'b': raw bytes
- 'f': flat data
- 'd': document

REQUEST_RECORD_CREATE

Create a new record. Returns the position in the cluster of the new record. New records can have version > 0 (since v1.0) in case the RID 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-
```

Where:

- **datasegment-id** the segment id to store the data (since version 10 - 1.0-SNAPSHOT). -1 Means default one. Removed since 2.0
- **record-type** is:
- 'b': raw bytes
- 'f': flat data
- 'd': document

and **mode** is:

- 0 = synchronous (default mode waits for the answer)
- 1 = asynchronous (don't need an answer)

The last part of response is referred to RidBag management. Take a look at the main page for more details.

REQUEST_RECORD_UPDATE

Update a record. Returns the new record's version.

```
Request: (cluster-id:short)(cluster-position:long)(update-content:boolean)(record-content:by
Response: (record-version:int)(count-of-collection-changes)[(uuid-most-sig-bits:long)(uuid-l
```

Where **record-type** is:

- 'b': raw bytes
- 'f': flat data
- 'd': document

and **record-version policy** is:

- '-1': Document update, version increment, no version control.
- '-2': Document update, no version control nor increment.
- '-3': Used internal in transaction rollback (version decrement).
- '>-1': Standard document update (version control).

and **mode** is:

- 0 = synchronous (default mode waits for the answer)
- 1 = asynchronous (don't need an answer)

and **update-content** is:

- true - content of record has been changed and content should be updated in storage
- false - the record was modified but its own content has not been changed. So related collections (e.g. rig-bags) have to be updated, but record version and content should not be.

The last part of response is referred to RidBag management. Take a look at the main page for more details.

REQUEST_RECORD_DELETE

Delete a record by its RecordID. During the optimistic transaction the record will be deleted only if the versions match. Returns true if has been deleted otherwise false.

```
Request: (cluster-id:short)(cluster-position:long)(record-version:int)(mode:byte)
Response: (payload-status:byte)
```

Where:

- **mode** is:
- 0 = synchronous (default mode waits for the answer)
- 1 = asynchronous (don't need an answer)
- **payload-status** returns 1 if the record has been deleted, otherwise 0. If the record didn't exist 0 is returned.

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
```

Where the request:

- **mode** can be 'a' for asynchronous mode and 's' for synchronous mode
- **command-payload-length** is the length of the class-name field plus the command-payload field
- **class-name** is the class name of the command implementation. There are short form for the most common commands:
  - **q** stands for query as idempotent command. It's like passing `com.orientechnologies.orient.core.sql.query.OSQLSynchQuery`
  - **c** stands for command as non-idempotent command (insert, update, etc). It's like passing `com.orientechnologies.orient.core.sql.OCommandSQL`
  - **s** stands for script. It's like passing `com.orientechnologies.orient.core.command.script.OCommandScript`. Script commands by using any supported server-side scripting like Javascript command. Since v1.0.
  - **any other values** is the class name. The command will be created via reflection using the default constructor and invoking the `fromStream()` method against it
- **command-payload** is the command's serialized payload (see Network-Binary-Protocol-Commands)

Response is different for synchronous and asynchronous request:

- **synchronous**:
- **synch-result-type** can be:
  - 'n', means null result
  - 'r', means single record returned
  - 'l', collection 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
- **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: (tx-id:int)(using-tx-log:byte)(tx-entry)*(0-byte indicating end-of-records)

 tx-entry: (operation-type:byte)(cluster-id:short)(cluster-position:long)(record-type:byte)(
  entry-content for CREATE: (record-content:bytes)
  entry-content for UPDATE: (version:record-version)(content-changed:boolean)(record-content
  entry-content for DELETE: (version:record-version)

Response: (created-record-count:int)[(client-specified-cluster-id:short)(client-specified-cl
```

Where:

- **tx-id** is the Transaction's Id
- **using-tx-log** tells if the server must use the Transaction Log to recover the transaction. 1 = true, 0 = false. Use always 1 (true) by default to assure consistency. *NOTE: Disabling the log could speed up execution of transaction, but can't be rollbacked in case of error. This could bring also at inconsistency in indexes as well, because in case of duplicated keys the rollback is not called to restore the index status.*
- **operation-type** can be:
  - 1, for **UPDATES**
  - 2, for **DELETES**
  - 3, for **CREATIONS**

- **record-content** depends on the operation type:
  - For **UPDATED** (1): `(original-record-version:int)(record-content:bytes)`
  - For **DELETED** (2): `(original-record-version:int)`
  - For **CREATED** (3): `(record-content:bytes)`

This response contains two parts: a map of 'temporary' client-generated record ids to 'real' server-provided record ids for each CREATED record, and a map of UPDATED record ids to update record-versions.

Look at Optimistic Transaction to know how temporary RecordIDs are managed.

The last part or response is referred to RidBag management. Take a look at the main page for more details.

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.

History

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 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 `@`. 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* `&` 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:
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.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: 23439223l
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	1002020303	Value in milliseconds ends with 'a'. Example: 1002020303a
Datetime	1002020303	Value in milliseconds ends with 't'. Example: 1002020303t
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: `<20, 30>`
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 no need of a external schema definition and with support for partial serialization/deserialization.

The whole record is structured in three main segments

```
+---------------+------------------+--------------+-------------+
| version:byte  | className:string | header:byte[] | data:byte[] |
+---------------+------------------+--------------+-------------+
```

Version

1 byte that contain the version of the current record serialization, to allow progressive serialization upgrade

Class Name

A String containing the name of the class of the record, if the record has no class will be just an empty string, the serialization of the string is the same of the String value

Header

The header contains the list of fields names of the current record with the association to the data location

```
+---------------------------+
| fields:field_definition[]  |
+---------------------------+
```

field definition

```
+----------------------------+------------------+-------------------------------+---------
| field_name_length|id:varint | field_name:byte[] | pointer_to_data_structure:int | data_typ
+----------------------------+------------------+-------------------------------+---------
```

field_name_length varint that describe the field, if positive is the size of the string that fallow next if negative is and id of current property referred in the schema, if is 0 mark the end of the header.
field_name the field name present only with **field_name_length** > 0
pointer_to_data a pointer to the data structure in the data segment that contains the field value or 0 if the field is null
data_type the field type id, the supported types are defined here OType present only with **field_name_length** > 0

Property ID

The property Id will be stored in **field_name_length** as negative value, for decode it should translated to positive value and decrased by 1: (**field_name_length** * -1) -1 == propertyId.

the relative property will be found in the schema, stored in the globalProperty list at the root of the document that rapresent the schema definition.

Data

The data segment is where the data is stored is composed by an array of data structure

```
+------------------------+
| data:data_structure[]  |
+------------------------+
```

each data structures content is depended to the field type, each type have it's own serialization structure

field_data serialization by type

SHORT,INTEGER,LONG

The Integer numbers will be serialized as variable size integer it use the same format of protobuf specified HERE
-64 < value < 64 1 byte
-8192 < value < 8192 2 byte
-1048576 < value < 1048576 3 byte
-134217728 < value < 134217728 4 byte
-17179869184 < value < 17179869184 5 byte

all the negative value are translated to positive using the ZigZag encoding

the algorithm can be also extended for longer values!

BYTE

The byte is stored as byte

BOOLEAN

The boolean is serialized as a byte: 0 = false 1 = 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

```
+---------------+
| float:byte[8] |
+---------------+
```

DATETIME

The date is converted to millisecond unix epoch and stored as the type LONG

DATE

The date is converted to second unix epoch,moved at midnight UTC+0, divided by 86400(seconds in a day) and stored as the type LONG

STRING

The string are stored as binary structure with UTF-8 encoding

```
+-------------+----------------+
| size:varInt | string:byte[]  |
+-------------+----------------+
```

size the number of the bytes in the string stored(not the length of the string) as variable size integer **string** the bytes of the string in UTF-8 encodings

BINARY

The BINARY store bytes in a row way on the storage

```
+--------------+----------------+
| size:varInt  | bytes:byte[]   |
+--------------+----------------+
```

size the number of the bytes to store **bytes** the row bytes

EMBEDDED

The embedded document is serialized calling the document serializer in recursive fashion, in the following structure

embedded document

```
+---------------------------+
```

```
| serialized_document:bytes[] |
+-----------------------------+
```

**serialized_document the bytes of the serialized document

EMBEDDEDLIST, EMBEDDEDSET

The embedded collections is stored as an array of bytes that contain the serialized document in the embedded mode.

```
+-------------+------------+-------------------+
|size:varInt  | type:Otype | items:item_data[] |
+-------------+------------+-------------------+
```

size the number of items in the list **type** the type of the types in the list or ANY if the type is unknown **items** an array of value serialized by type or if the type is ANY the item will have it's own structure.

the item_data structure is: +------------------+--------------+ l data_type:OType l data:byte[] l +------------------+--------------+ **data_type** the type of the data stored in the item. **data** the data stored with the format choose by the OType.

EMBEDDEDMAP

The link map allow to have as key the types: STRING,SHORT,INTEGER,LONG,BYTE,DATE,DECIMAL,DATETIME,DATA,FLOAT,DOUBLE the serialization of the map is divided in a header and a values

```
+--------------------------+------------------------+
| header:headerStructure    | values:valueStructure   |
+--------------------------+------------------------+
```

header structure

```
+--------------+------------------+
| keyType:byte | keyValue:byte[]  |
+--------------+------------------+
```

Current implementation convert all the keys to string **keyType** is the type of the key,

can be only one of the listed type. **keyValue** the value of the key serialized with the serializer of the type

value structure

```
+---------------+---------------+
|valueType:byte | value:byte[]  |
+---------------+---------------+
```

valueType the OType of the stored value **value** the value serialized with the serializer selected by OType

LINK

The link is stored as two 64 bit integer

```
+--------------+--------------+
|cluster:64int | record:64Int |
+--------------+--------------+
```

cluster orientdb cluster id **record** orientdb record id

LINKLIST, LINKSET

```
+-------------+---------------------+
| size:varint | collection:LINK[] |
+-------------+---------------------+
```

size the number of links in the collection **collection** an array of LINK each element is serialized as LINK type.

LINKMAP

The link map allow to have as key the types: STRING,SHORT,INTEGER,LONG,BYTE,DATE,DECIMAL,DATETIME,DATA,FLOAT,DOUBLE the serialization of the linkmap is a list of entry

```
+---------------------------+
| values:link_map_entry[]    |
```

```
+---------------------------+
```

link_map_entry structure

```
+--------------+------------------+------------+
| keyType:byte | keyValue:byte[]  | link:LINK  |
+--------------+------------------+------------+
```

keyType is the type of the key, can be only one of the listed type. **keyValue** the value of the key serialized with the serializer of the type **link** the link value store with the formant of a LINK

DECIMAL

The Decimal is converted to an integer and stored as scale and value (example "10234.546" is stored as scale "3" and value as:"10234546")

```
+---------------+-------------------+--------------+
| scale:byte[4] | valueSize:byte[4] | value:byte[] |
+---------------+-------------------+--------------+
```

scale an 4 byte integer that represent the scale of the value **valueSize** the length of the value bytes **value** the bytes that represent the value of the decimal in big-endian order.

LINKBAG

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 peremeters 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-paramete
```

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
```

language the language of the script present in the text field. All the others paramenters are serialized as the SQL Commands

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 Use case
- Use OrientDB as a Key/Value DBMS

Time Series Use Case

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

So when you've millions of record the best way to scale up linearly is avoid using indexes at all or as much as you can. But how to retrieve records in 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. Follows 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 flatten( 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 flatten( 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 flatten( 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 flatten( 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 flatten( logs ) from (
  select union( logs ) as logs from (
    traverse hour from (
     select flatten( 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 records to the closest Time Unit like the example above, but you could sum all the records in the same Day and put 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
)
```

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

```
update (
  select flatten( month[3].day[20] ) from Year where year = "2012"
) add logs = #23:45
```

Key Value Use Case

OrientDB can be used like 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 major length, 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 UN
```

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':'J
```

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" }
  ]
}
```

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.

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/orient-db.sh (or bin/orient-db.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.comma
      <parameters>
        <entry value="Cache-Control: no-cache, no-store, max-age=0, must-revalidate\r\nPragm
        <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 name=\"http\" implementation=\"com.orientechnologies.orient.server.network.p
   + "</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/relea
   + "<entry name=\"orientdb.config.file\" value=\"C:/work/dev/orientechnologies/orientdb/re
   + "<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
   + "<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 n
  user     : the user to connect with (this is NOT the same as root user in the configuratio
  password : the user to connect with (this is NOT the same as root password in the configur
```

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();
  }

}
```

Or 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();
  }

}
```

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();
  }
}
```

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.

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 iRequestT
    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<?>) iReco
    } 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 Except
    // 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, res

    // 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

    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, li

    } 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 Plugin

Java class implementation:

```
com.orientechnologies.orient.server.handler.OAutomaticBackup
```

Introduction

Configure an automatic backup of databases. 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
- **delay**: delay time. You can use different suffixes to specify different measures:
  - **ms** for milliseconds. Example 10000ms means 10 seconds
  - **s** for seconds. Example 10s means 10 seconds
  - **m** for minutes. Example 5m means 5 minutes
  - **h** for hours. Example 24h means every day
  - **d** for days. Example 1d means every day

- **target.directory**: target directory, the default is "backup"
- **target.fileName**: target file name configurable using the following variables between `${}`:
  - `${DBNAME}`, as the database name
  - `${DATE}`, as the current date following the format. For the complete syntax look at Java DateTime syntax

- **db.include**: database list to include. If empty means all the databases
- **db.exclude**: database list to exclude
- **bufferSize**: In memory buffer size to use in compression. Default is 1MB. Bigger means faster backup but more RAM used (Since 1.7)
- **compressionLevel**: Compression level of the resulting ZIP file. Default is maximum: 9. Set it lower if backup takes too much time (Since 1.7)

Default configuration in orientdb-server-config.xml

```
<!-- 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" />
    <parameter name="delay" value="4h" />
    <parameter name="target.directory" value="backup" />
    <parameter name="target.fileName" value="${DBNAME}-${DATE:yyyyMMddHHmmss}.zip" /><!-- ${
    <parameter name="db.include" value="" /><!-- DEFAULT: NO ONE, THAT MEANS ALL DATABASES.
    <parameter name="db.exclude" value="" /><!-- USE COMMA TO SEPARATE MULTIPLE DATABASE NAM
    <parameter name="compressionLevel" value="9"/>
    <parameter name="bufferSize" value="1048576"/>
  </parameters>
</handler>
```

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
enabled	true to turn on, false (default) is turned off	boolean	true
`profile.<name>.mail.smtp.host`	The SMTP host name or ip-address	string	smtp.gmail.com
`profile.<name>.mail.smtp.port`	The SMTP port	number	587
`profile.<name>.mail.smtp.auth`	Authenticate in SMTP	boolean	true
`profile.<name>.mail.smtp.starttls.enable`	Enable the starttls	boolean	true
`profile.<name>.mail.smtp.user`	The SMTP username	string	yoda@starwars.com
`profile.<name>.mail.from`	The source's email address	string	yoda@starwars.com
`profile.<name>.mail.smtp.password`	The SMTP password	string	UseTh3F0rc3
`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

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's properties:

Name	Description	Mandatory	Example	Sin
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.
cc	Carbon copy addresses separated by commas	No	cc: "first@mail.com", "second@mail.com"	1.2.
bcc	Blind Carbon Copy addresses separated by commas	No	bcc : "first@mail.com", "second@mail.com"	1.2.
subject	The subject of the message	No	subject : "This Email plugin rocks!"	1.2.
message	The message's content	Yes	message : "Hi, how are you mate?"	1.2.
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.
attachments	The files to attach	No	attachments : "tmp/2.eml"	1.2.

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 applica
    });
```

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 applica
  });
```

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

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>
```

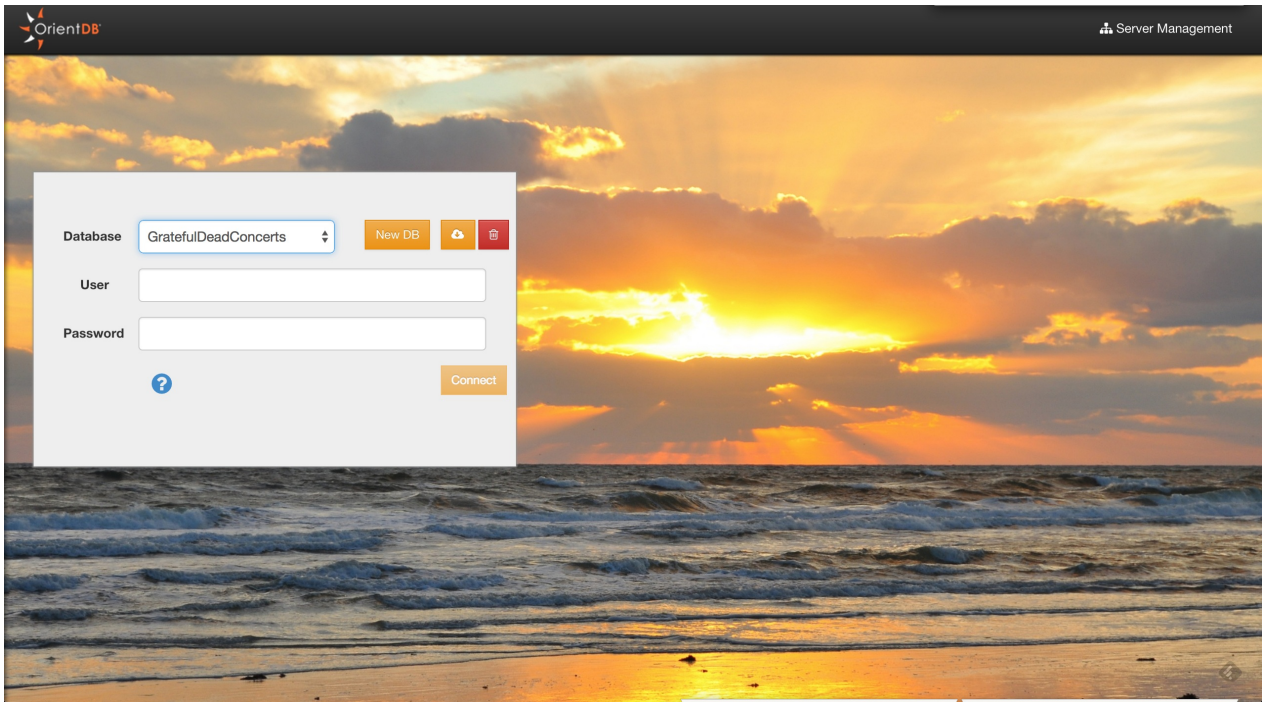
Studio Home page

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

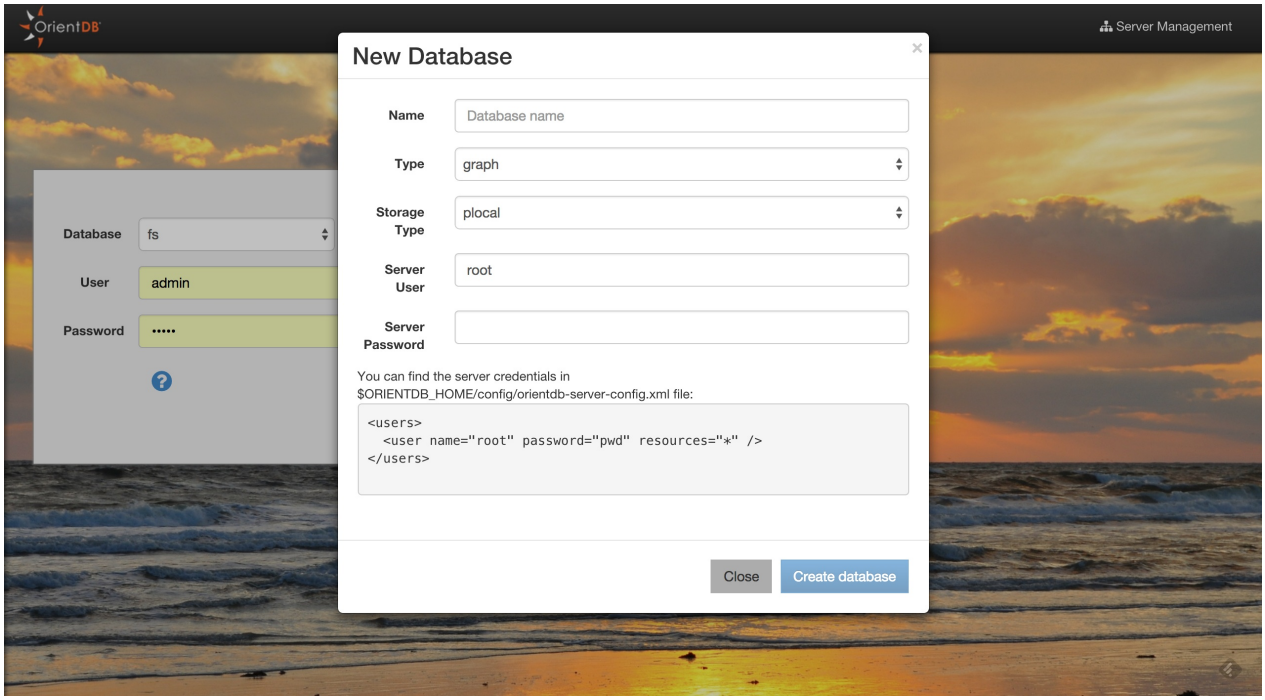
- Server User
- Server Password

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



Some information is needed to create a new database:

- Database name
- Database type (Document/Graph)
- Storage type (plocal/memory)
- Server user
- Server password

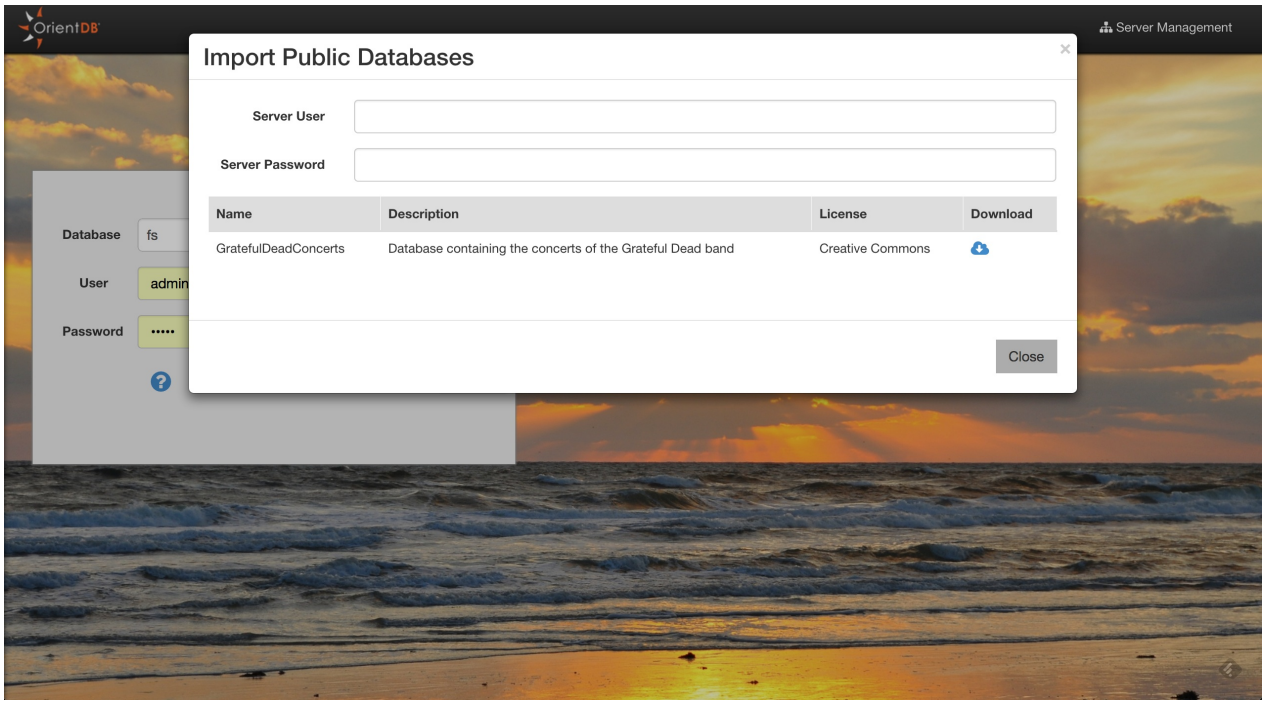
You can find the server credentials in the $ORIENTDB_HOME/config/orientdb-server-config.xml file:

```
<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.



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.

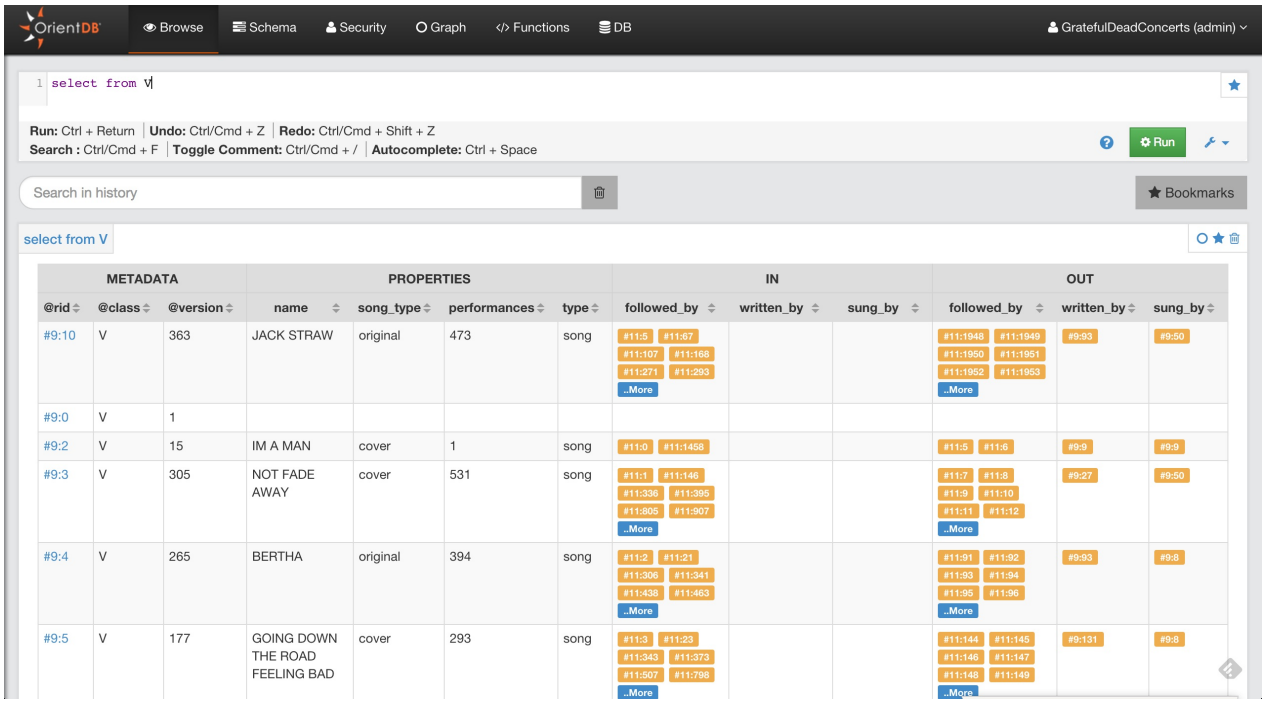
Execute a query

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:

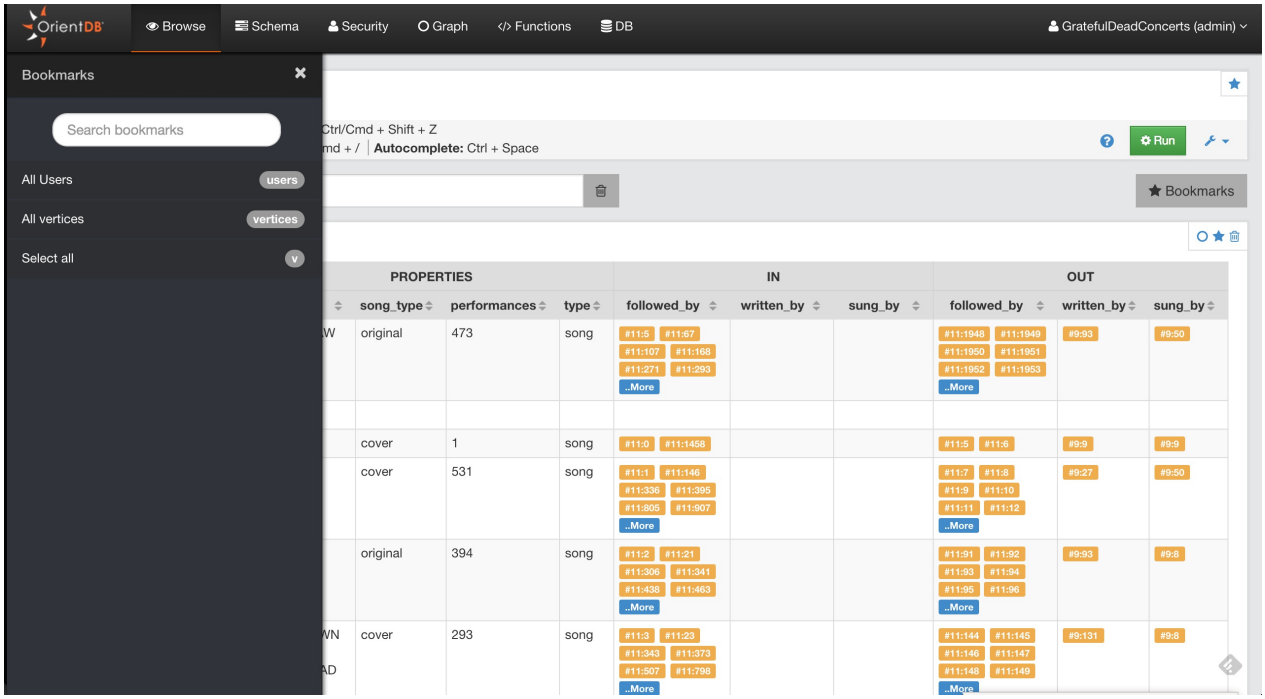
- **Ctrl + Return** to execute the query or just click the **Run** button
- **Ctrl/Cmd + Z** to undo changes
- **Ctrl/Cmd + Shift** + Z to redo changes
- **Ctrl/Cmd + F** to search in the editor
- **Ctrl/Cmd + /** to toggle a comment

> **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



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.

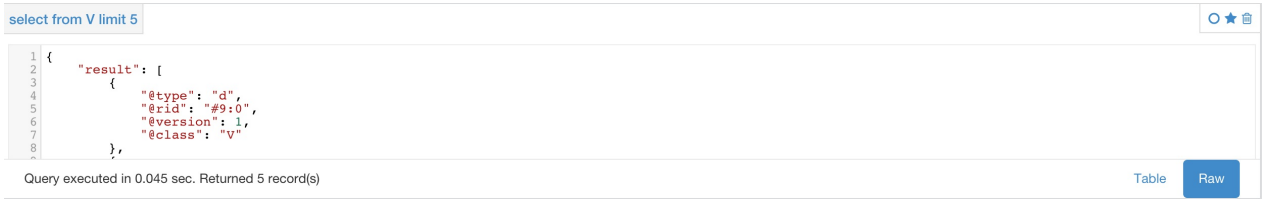


Studio saves the executed queries in the Local Storage of the browser, in the query settings, you can configure how many queries studio will keep in history. You can also search a previously executed query, delete all the queries from the history or delete a single query.

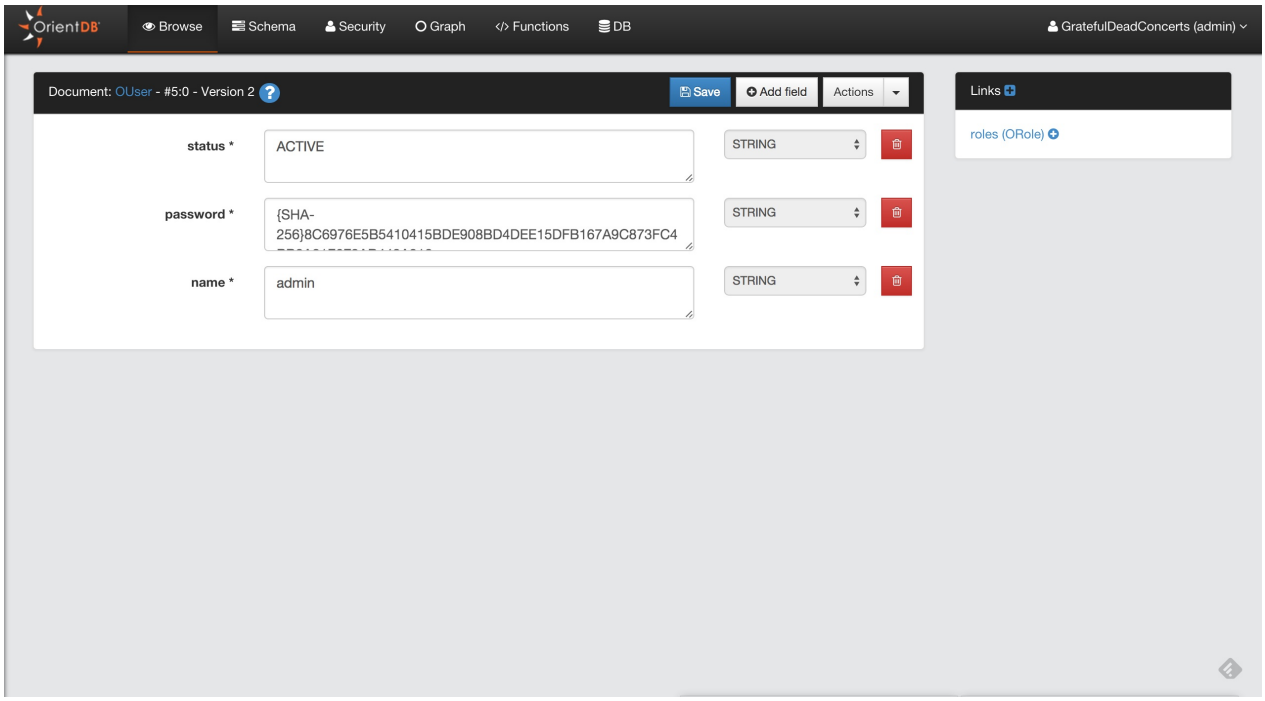
From Studio 2.0, you can send the result set of a query to the Graph Editor by clicking to the circle icon in the result set actions. This allows you to visualize your data graphically.

Look at the JSON output

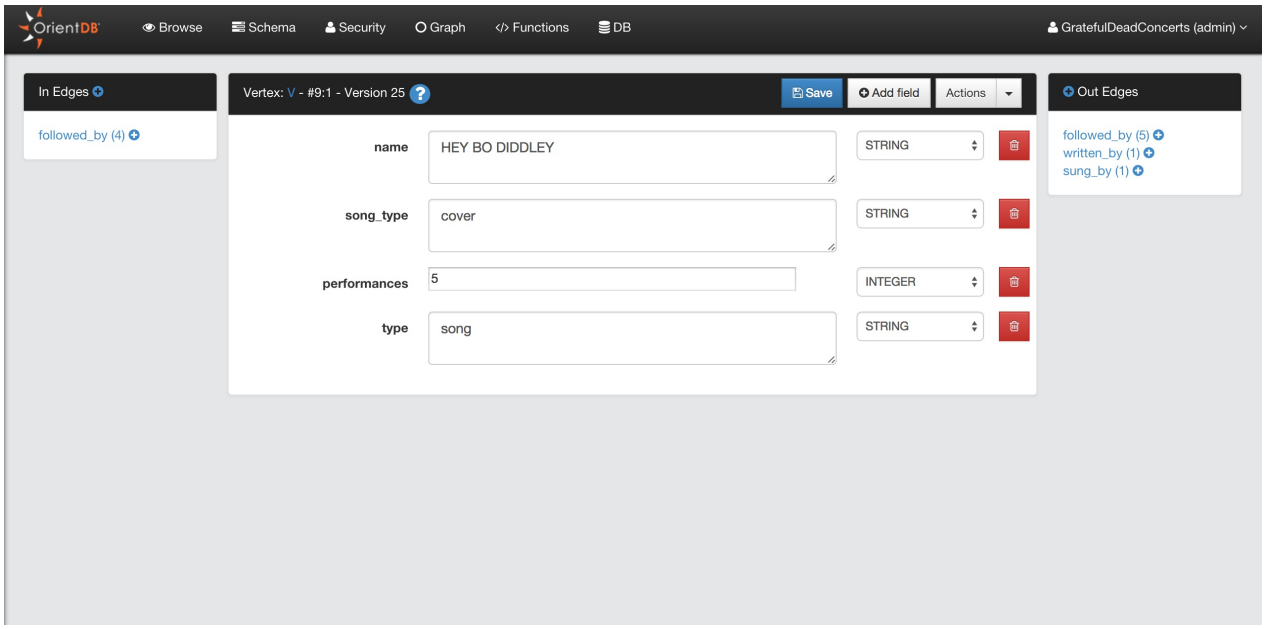
Studio speaks with the OrientDB Server using HTTP/RESt+JSON protocol. To see the output in JSON format, press the **RAW** tab.



Edit Document

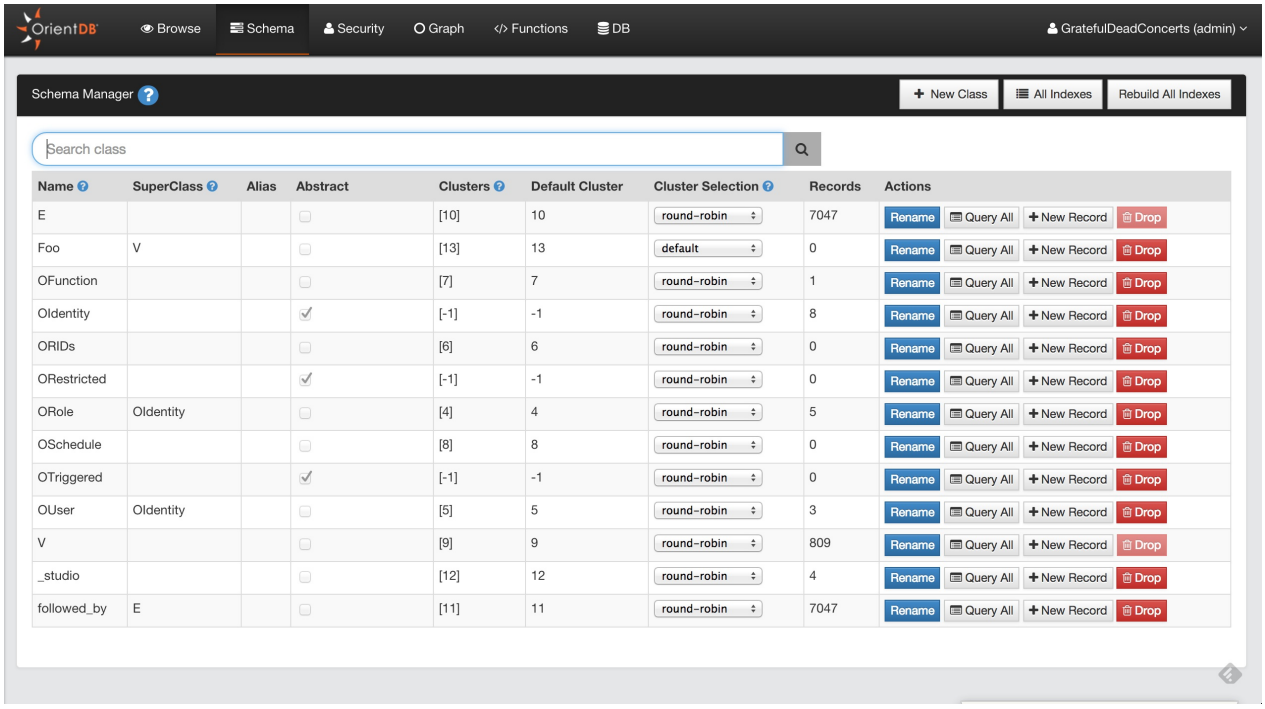


Edit Vertex


OrientDB
Browse
Schema
Security
Graph
Functions
DB
GratefulDeadConcerts (admin)
In Edges
followed_by (4)
Vertex: V - #9:1 - Version 25
Save
Add field
Actions
Out Edges
followed_by (5)
written_by (1)
sung_by (1)
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



Here you can :

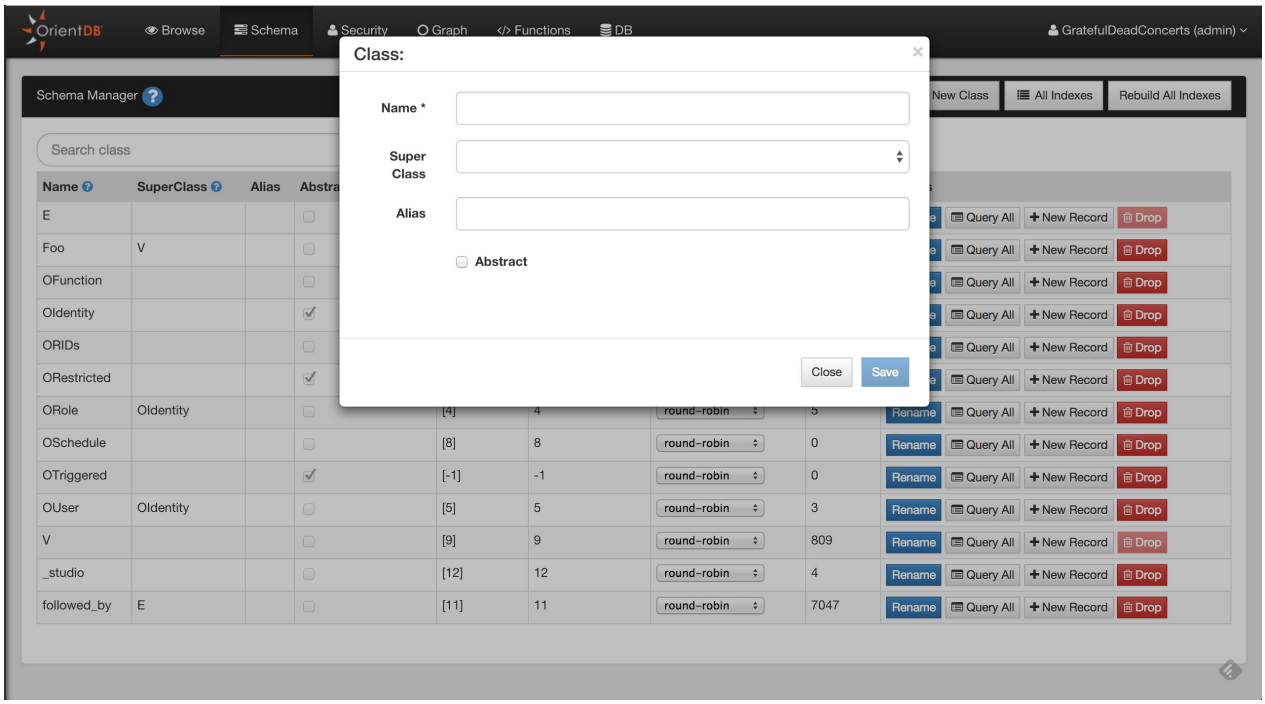
- Browse all the Classes of your database
- Create a new Class
- Rename/Drop a Class
- Change the cluster selection for a Class
- Edit a class by clicking on a class row in the table
- View all indexes created

Create a new Class

To create a new Class, just click the **New Class** button. Some information is required to create the new class.

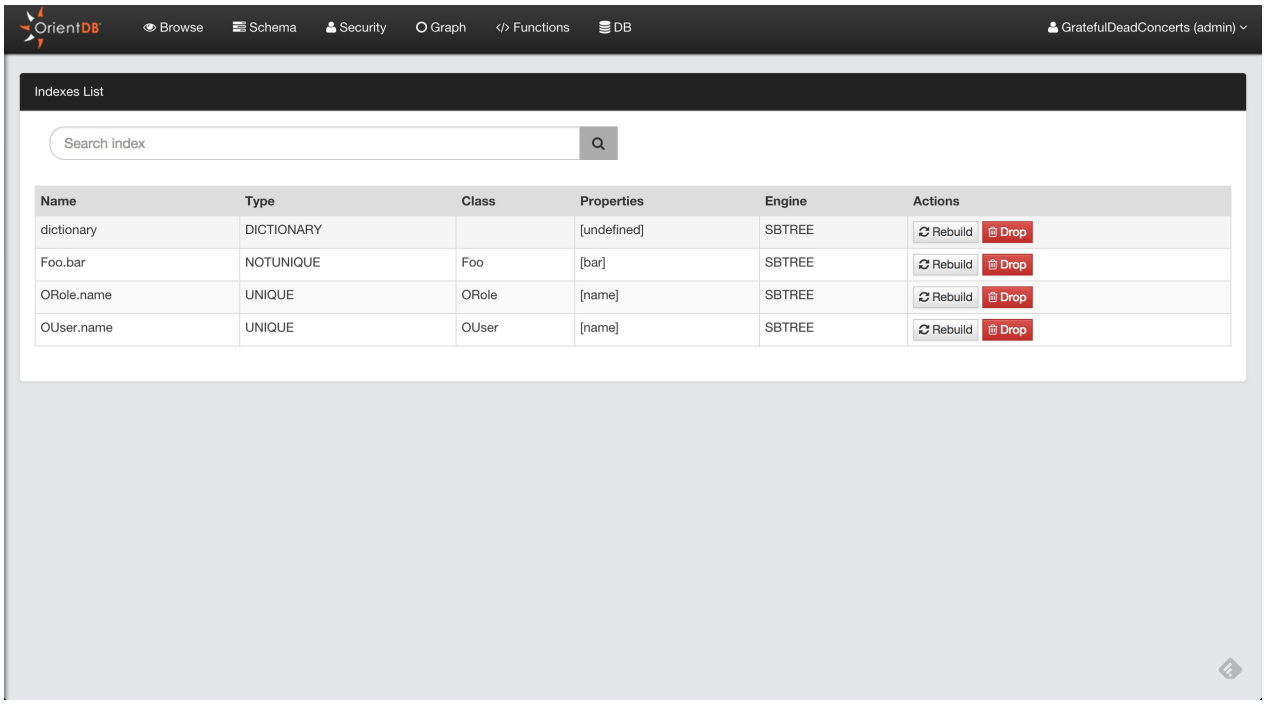
- Name
- SuperClass
- Alias (Optional)
- Abstract

Here you can find more information about Classes

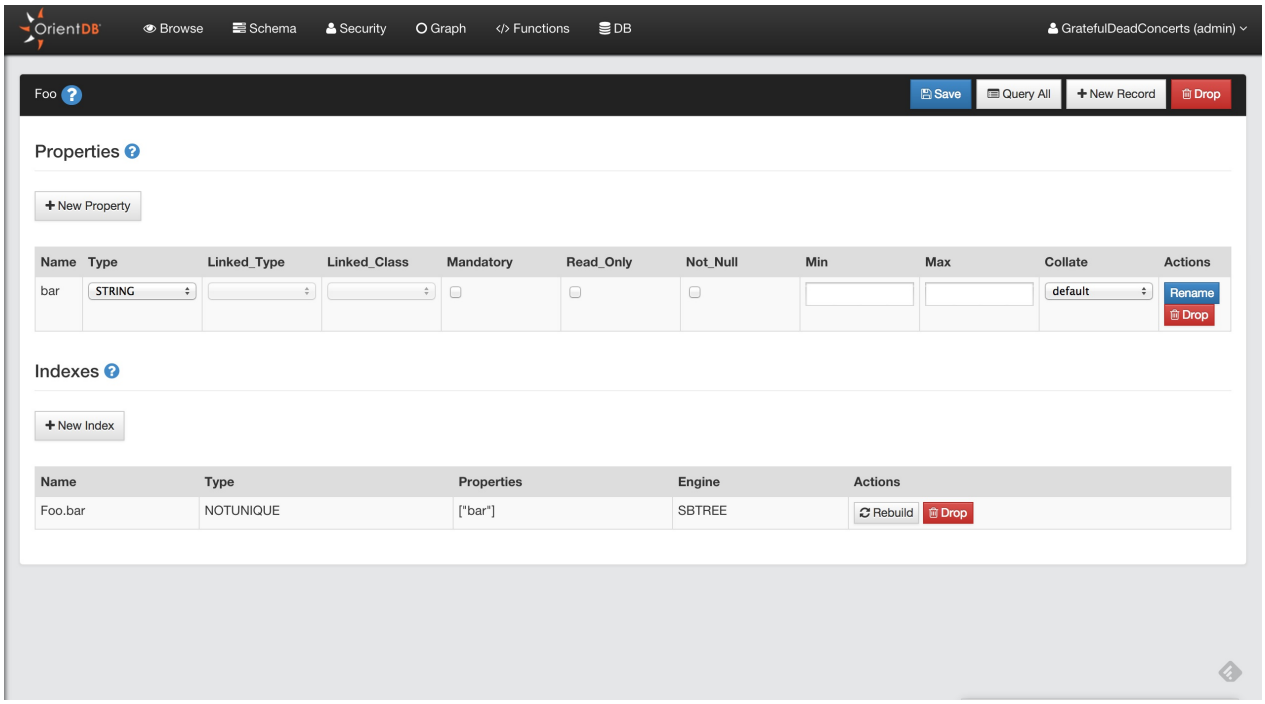


View all indexes

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.

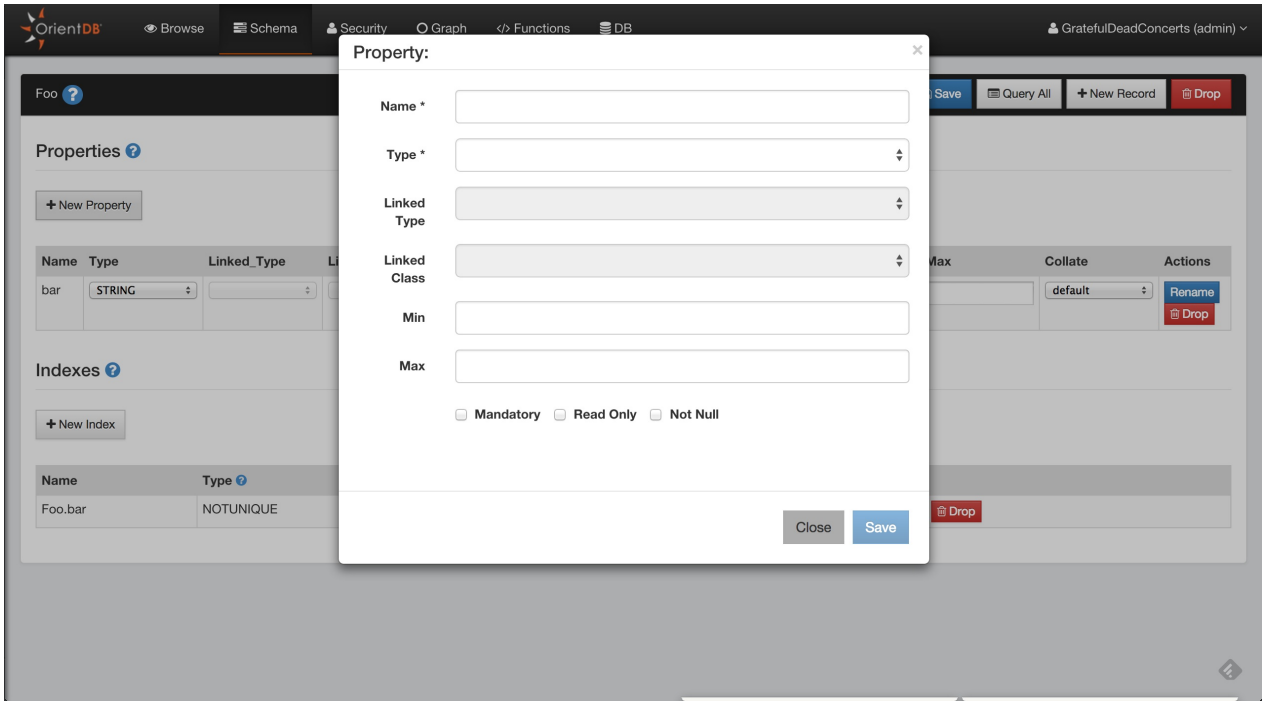


Class Edit


OrientDB
Browse
Schema
Security
Graph
Functions
DB
GratefulDeadConcerts (admin)
Foo
Save
Query All
New Record
Drop
Properties
New Property
Name
Type
Linked_Type
Linked_Class
Mandatory
Read_Only
Not_Null
Min
Max
Collate
Actions
bar
STRING
default
Rename
Drop
Indexes
New Index
Name
Type
Properties
Engine
Actions
Foo.bar
NOTUNIQUE
["bar"]
SBTREE
Rebuild
Drop

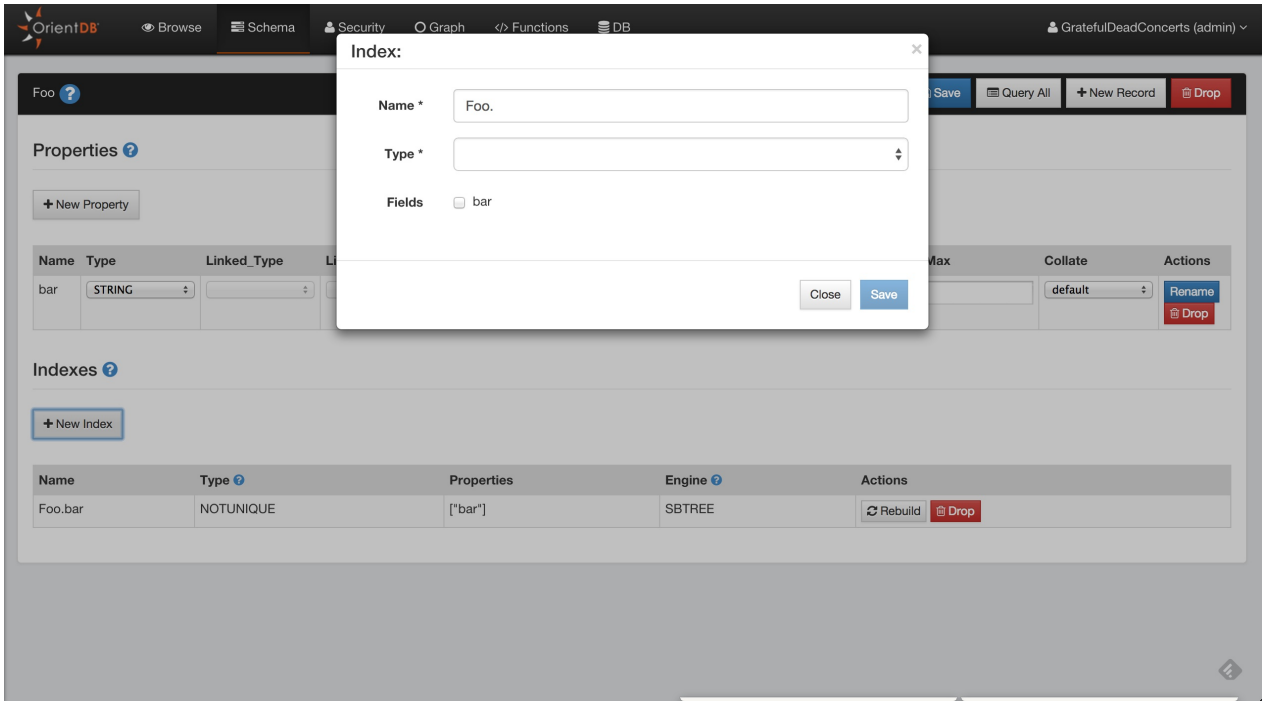
Property

Add Property



Indexes

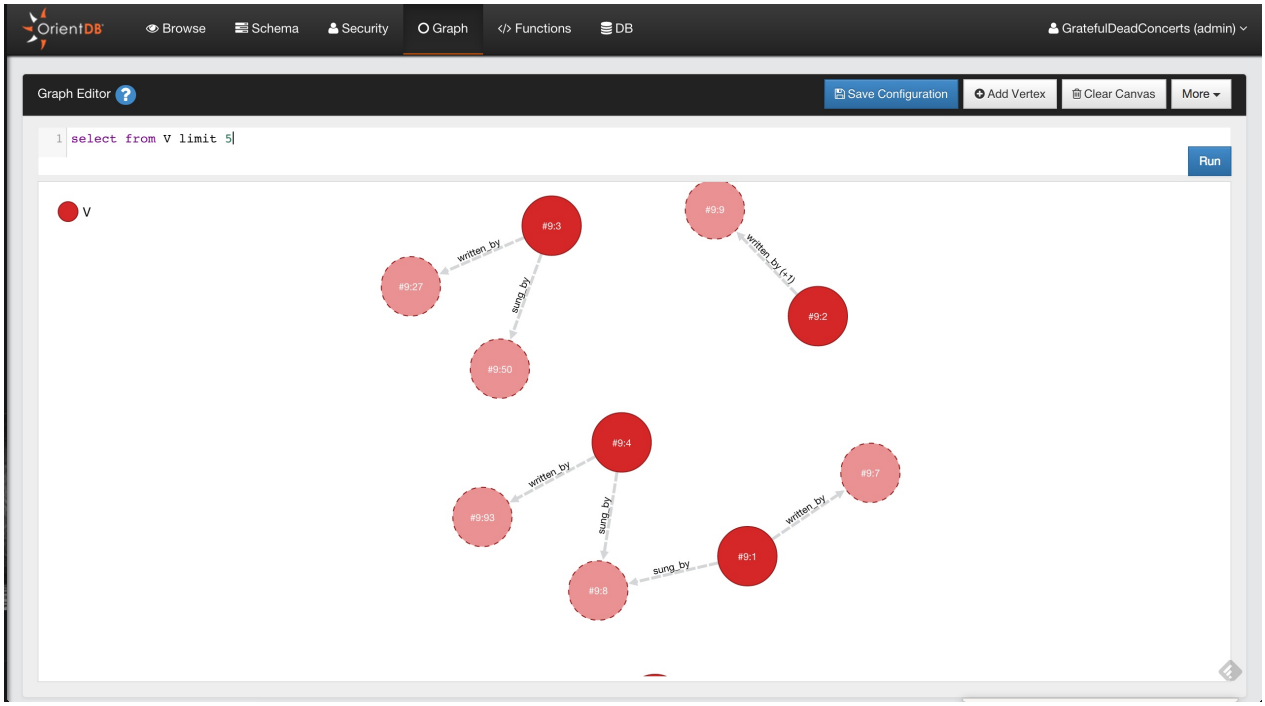
Create new index



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



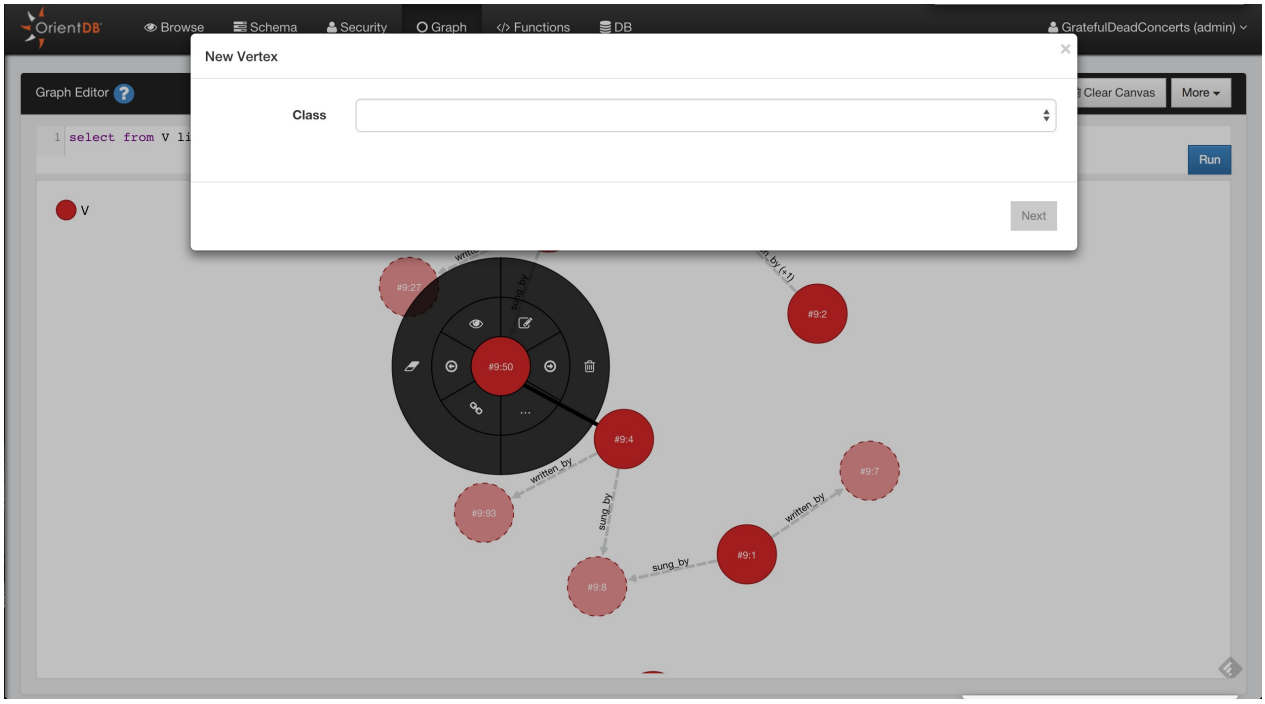
Supported operations in the Graph Editor are:

- 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

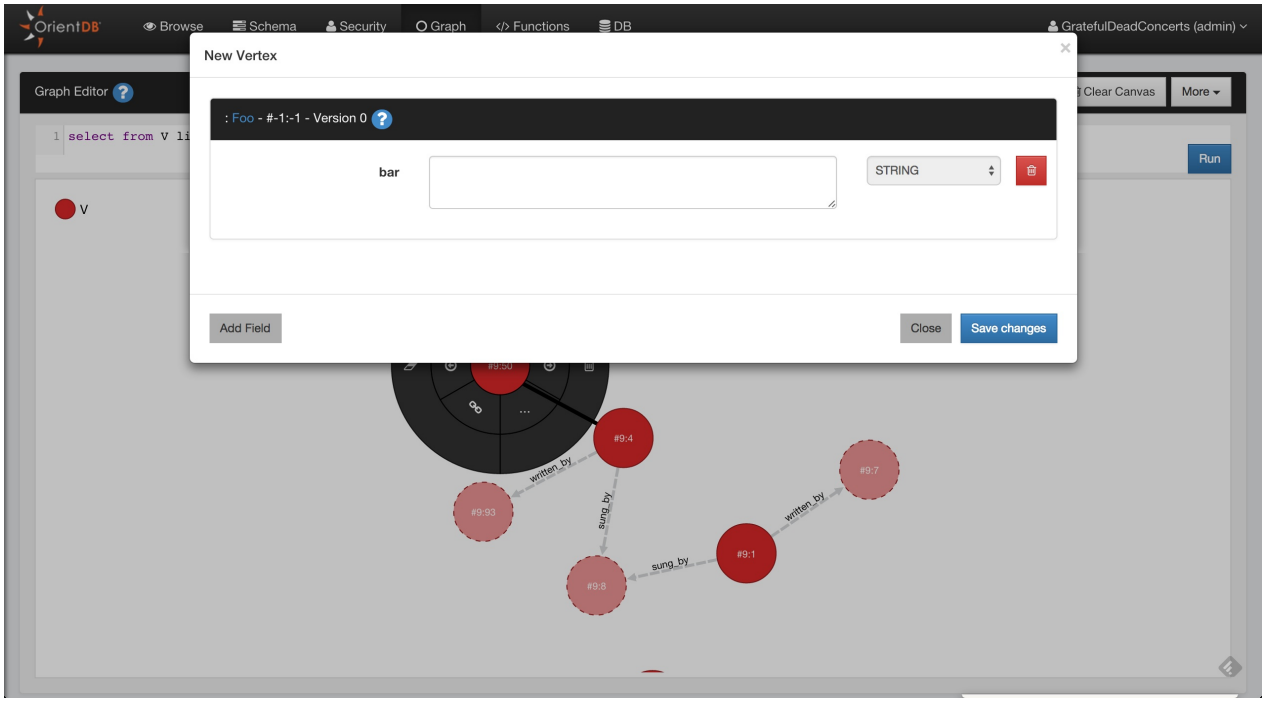
Add Vertices

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**



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

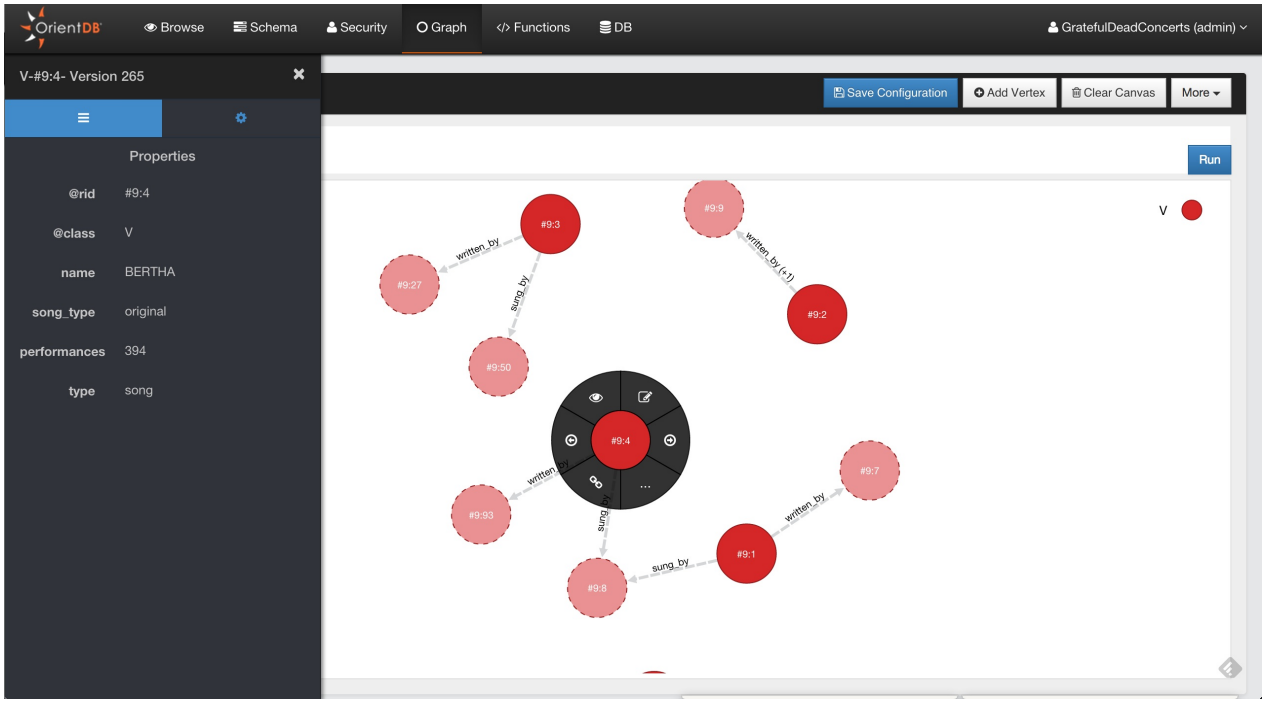
Open the circular menu , open the sub-menu by passing hover the mouse to the menu entry more (**...**) and then click the **eraser** icon.

Edit Vertices

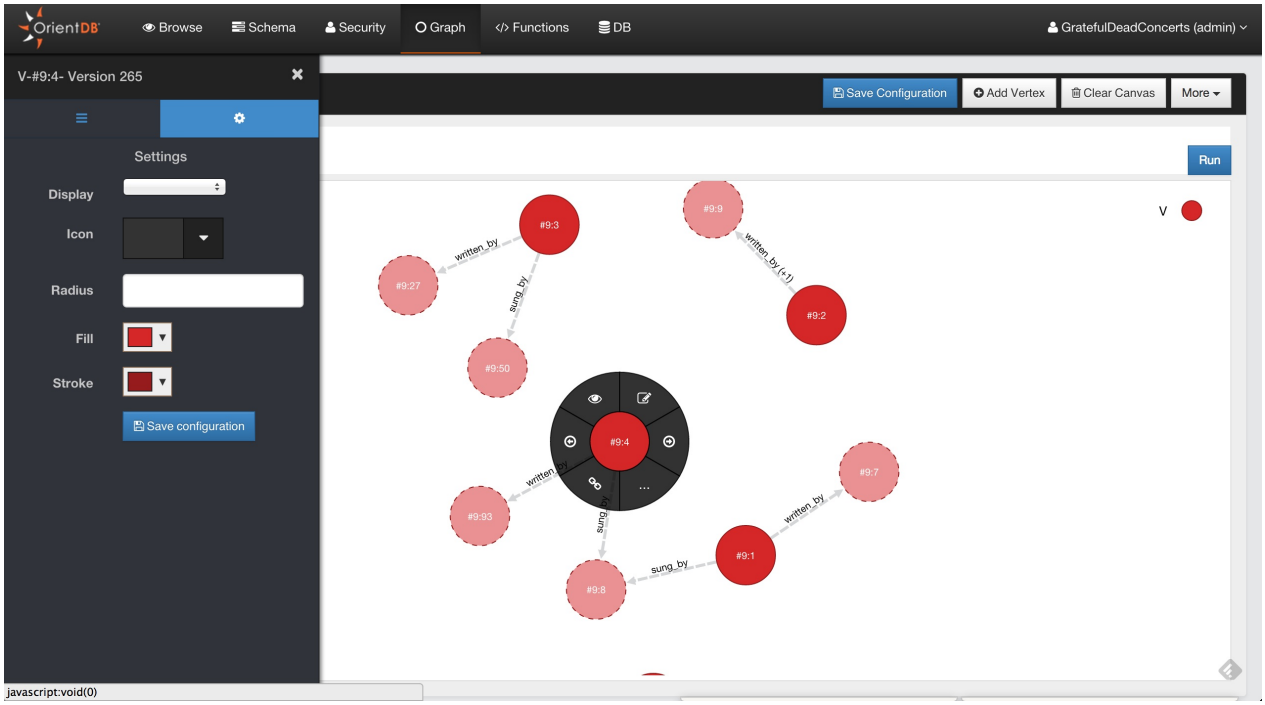
Open the circular menu and then click to the **edit** icon, Studio will open a popup where you can edit the vertex properties.

Inspect Vertices

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



Change the Rendering Configuration of Vertices



Navigating Relationships

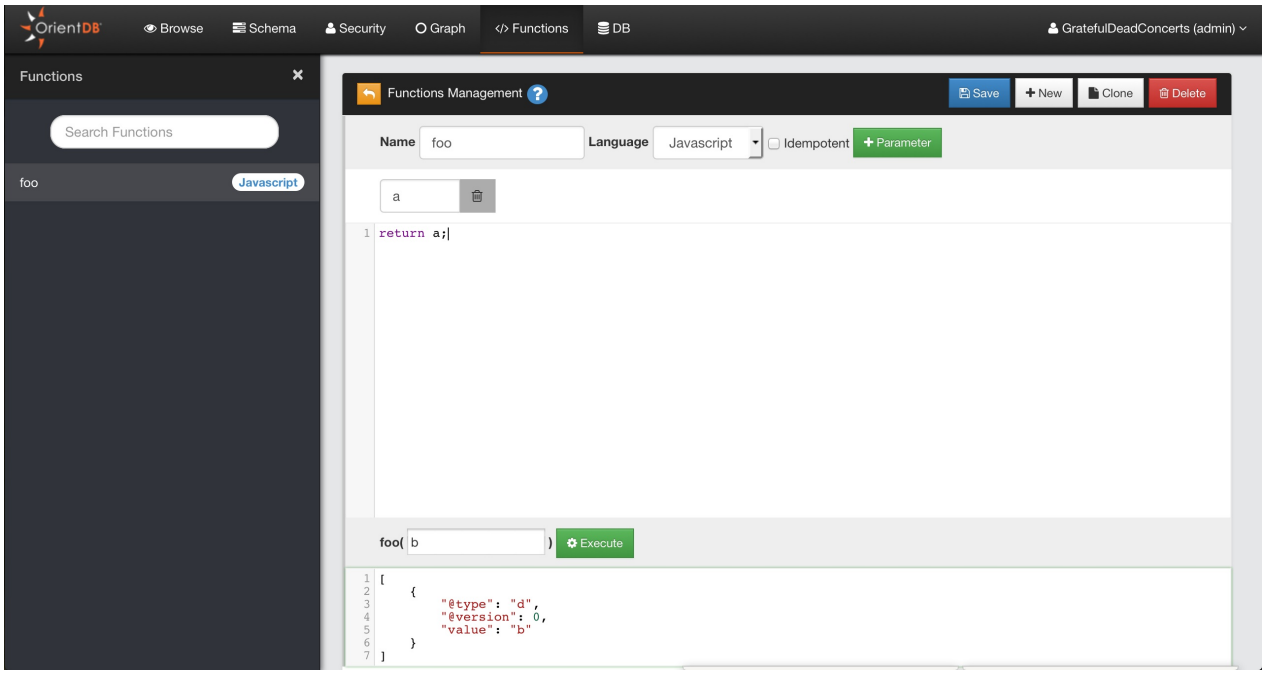
Create Edges between Vertices

Delete Edges between Vertices

Inspect Edges

Edit Edges

Functions


OrientDB
Browse
Schema
Security
Graph
Functions
DB
GratefulDeadConcerts (admin)
Functions
Search Functions
foo
Javascript
Functions Management
Save
New
Clone
Delete
Name
foo
Language
Javascript
Idempotent
Parameter
a
return a;
foo(b)
Execute
[
{
"@type": "d",
"@version": 0,
"value": "b"
}
]

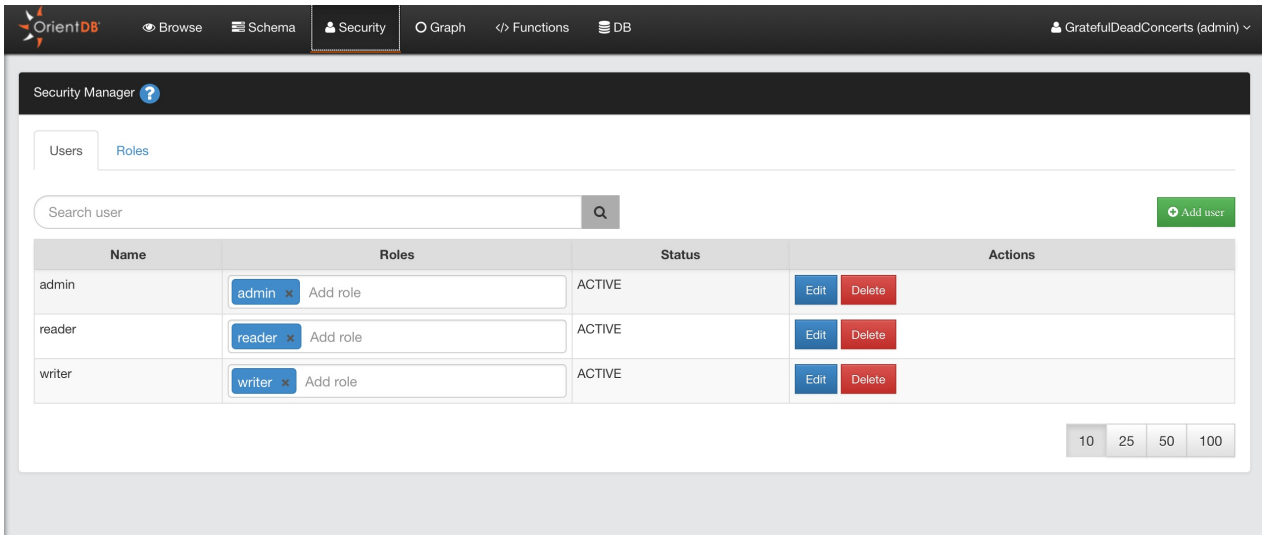
Security

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

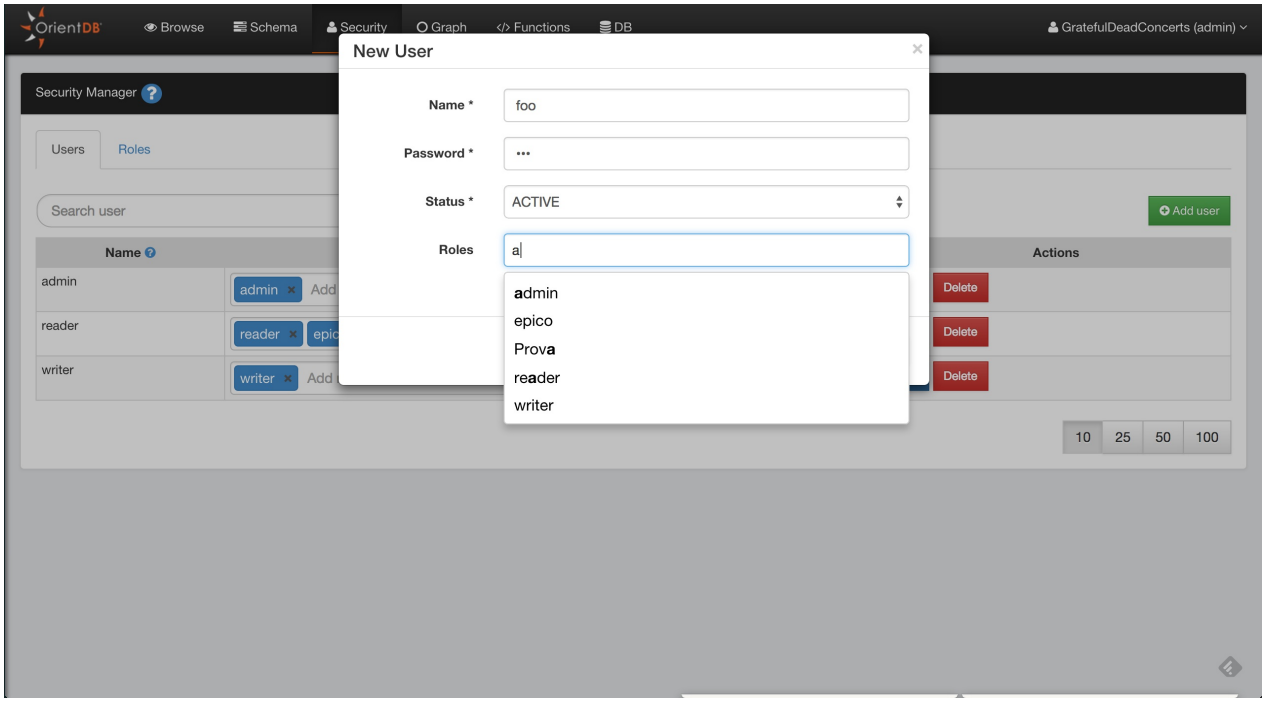
Here you can manage the database users:

- Search Users
- Add Users
- Delete Users
- Edit User: roles can be edited in-line, for name, status and password click the **Edit** button



Add Users

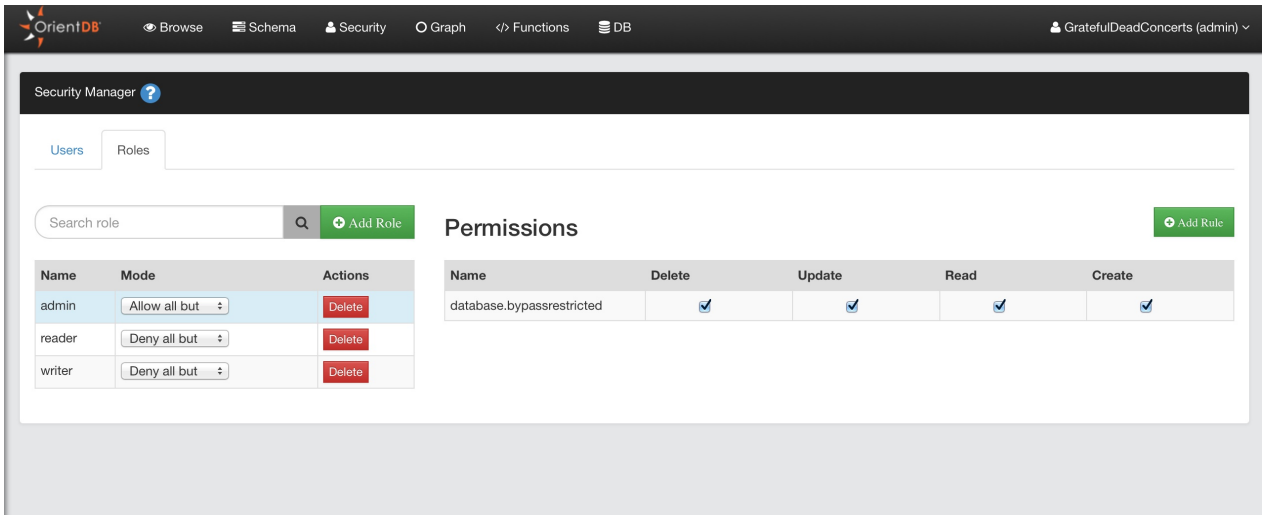
To add a new User, click the **Add User** button, complete the information for the new user (name, password, status, roles) and then save to add the new user to the database.



Roles

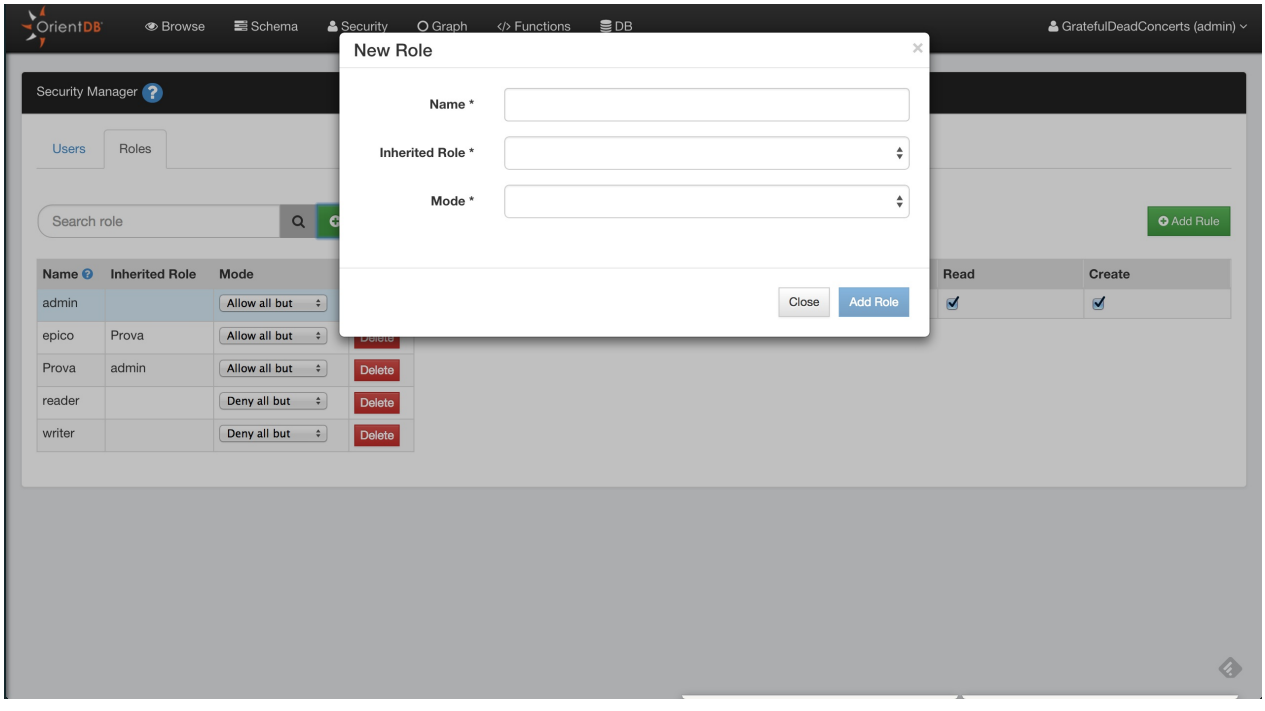
Here you can manage the database roles:

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



Add Role

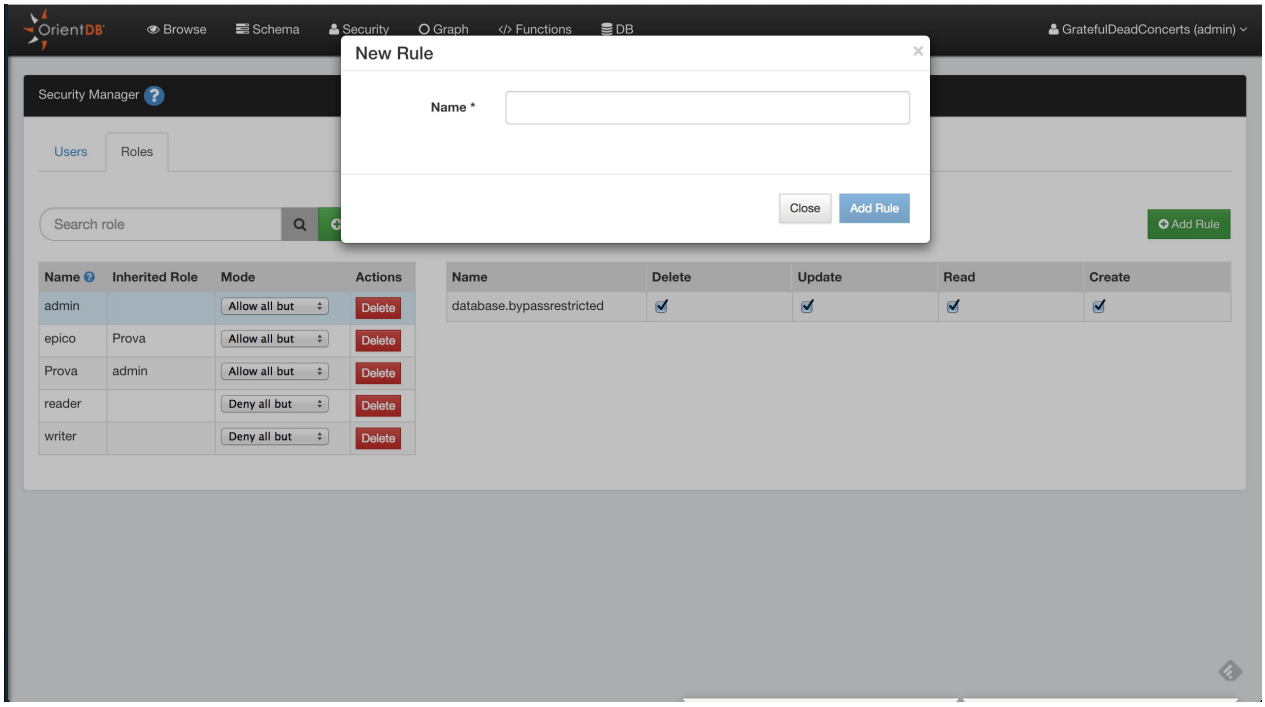
To add a new User, click the **Add Role** button, complete the information for the new role (name, parent role, mode) and then save to add the new role to the database.



Add Rule to a Role

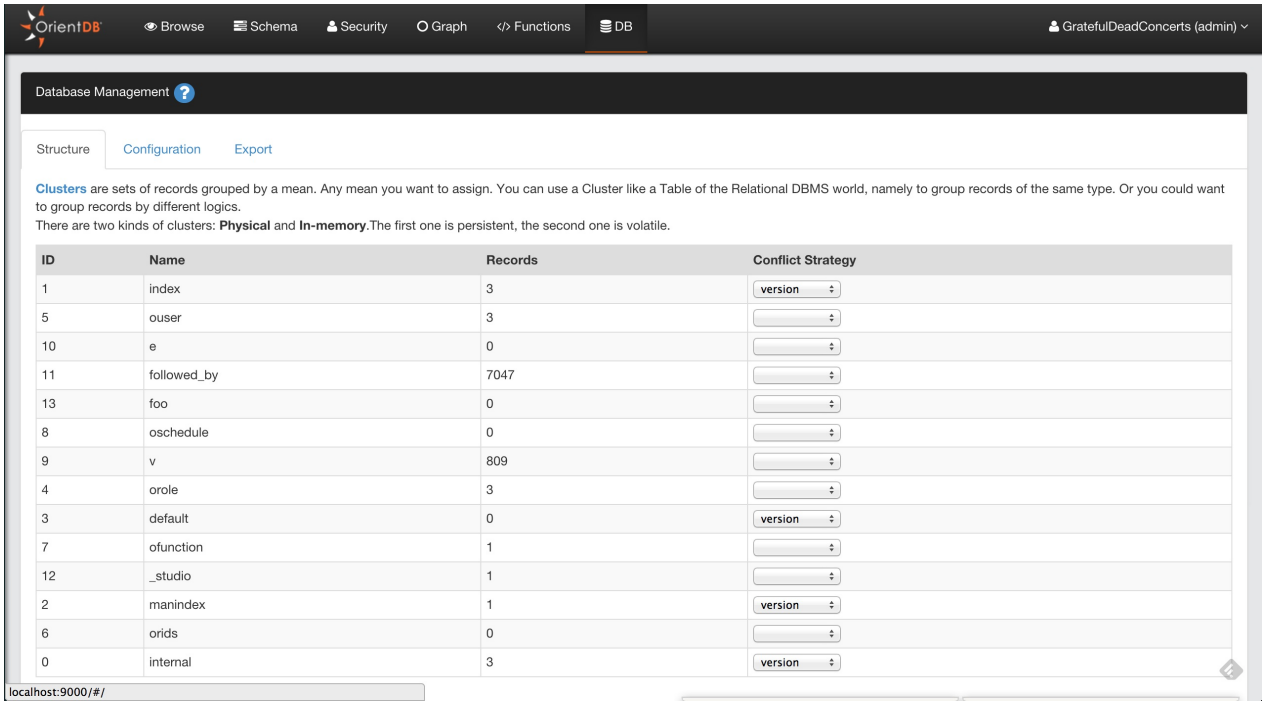
To add a new security rule for the selected role, click the *Add Rule* button. This will ask you the string of the resource that you want to secure. For a list of available resources, visit the official documentation here

Then you can configure the CRUD permissions on the newly created resource.

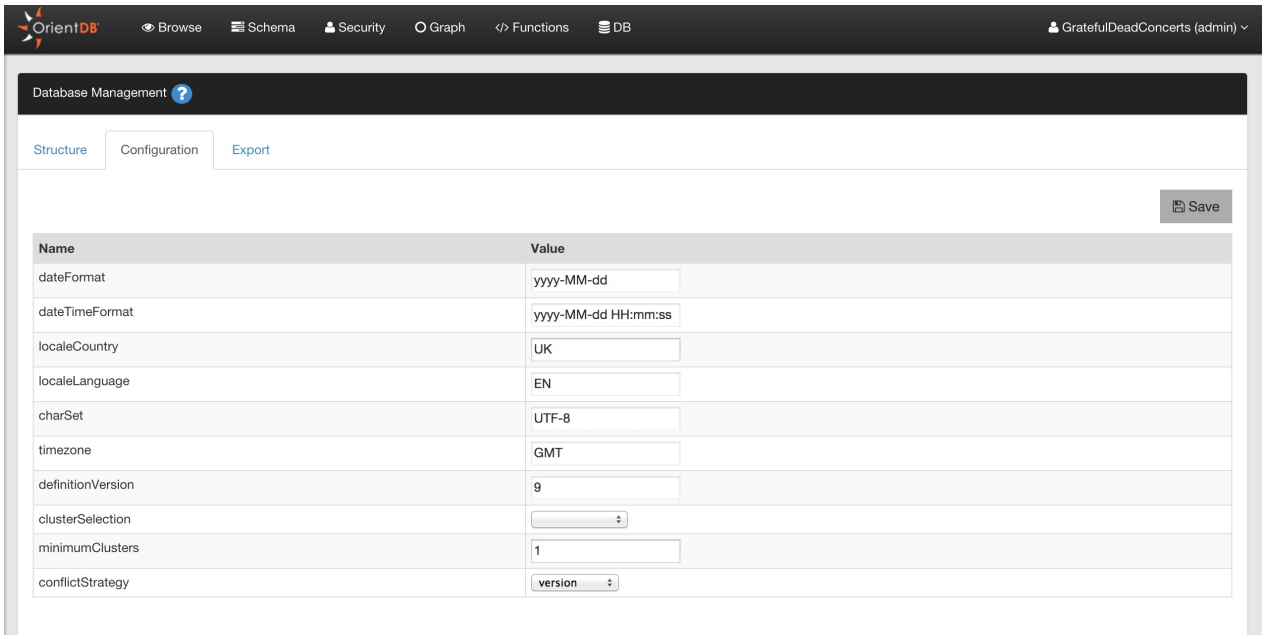


Database Management

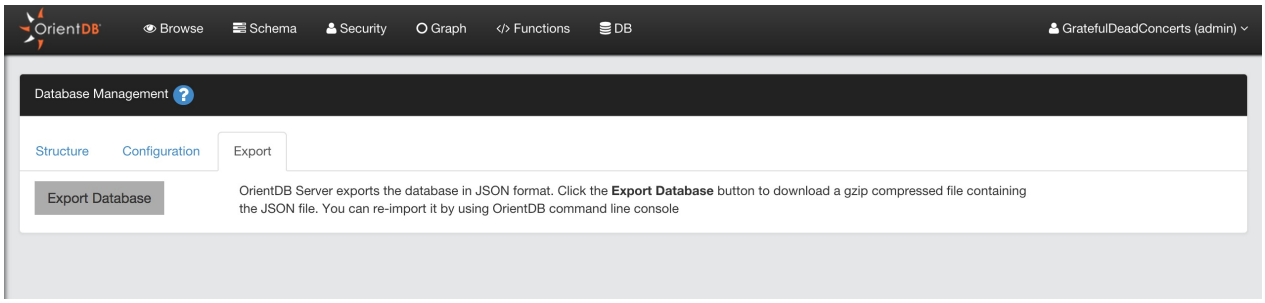
Structure



Configuration

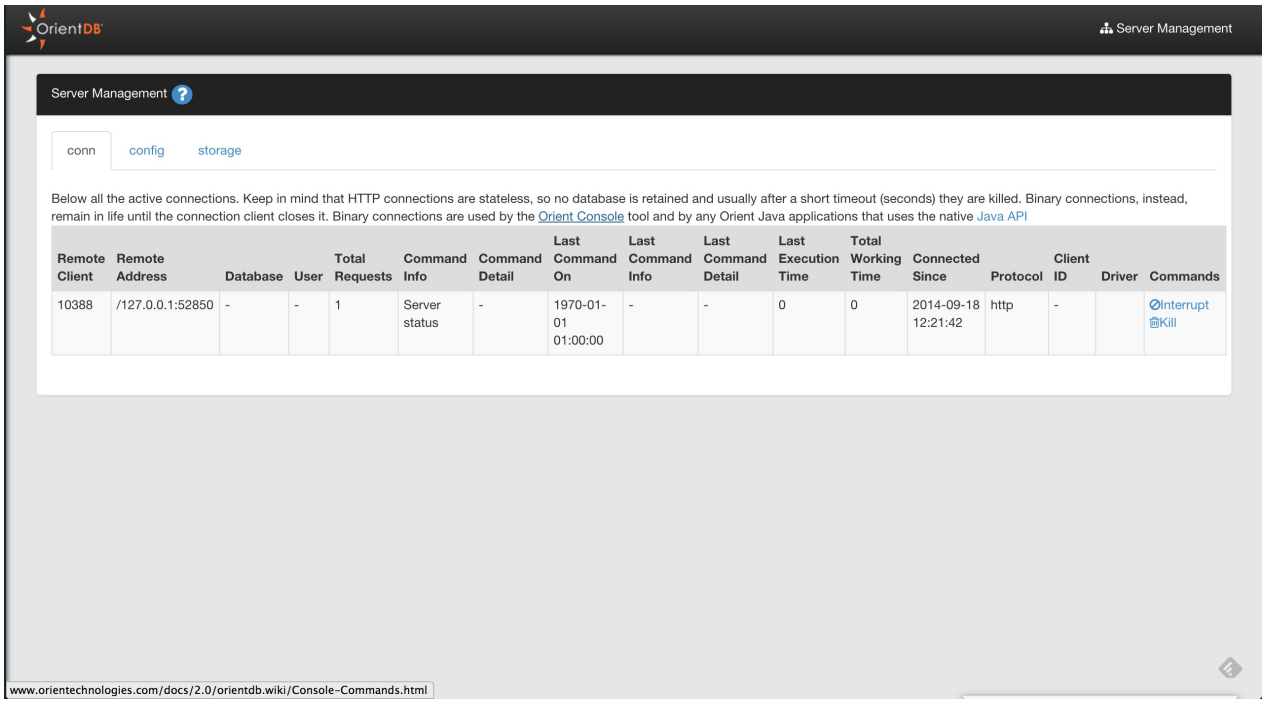


Export

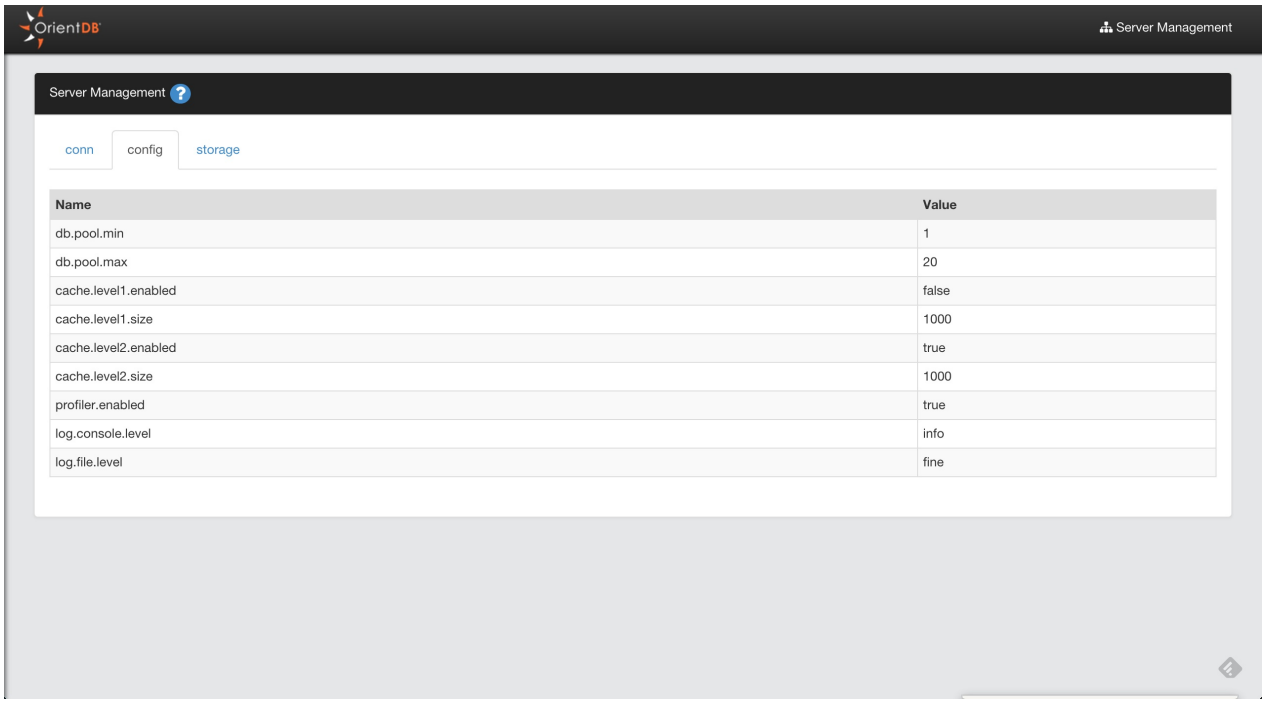

OrientDB
Browse
Schema
Security
Graph
Functions
DB
GratefulDeadConcerts (admin)
Database Management
Structure
Configuration
Export
Export Database
OrientDB Server exports the database in JSON format. Click the Export Database button to download a gzip compressed file containing the JSON file. You can re-import it by using OrientDB command line console

Server Management

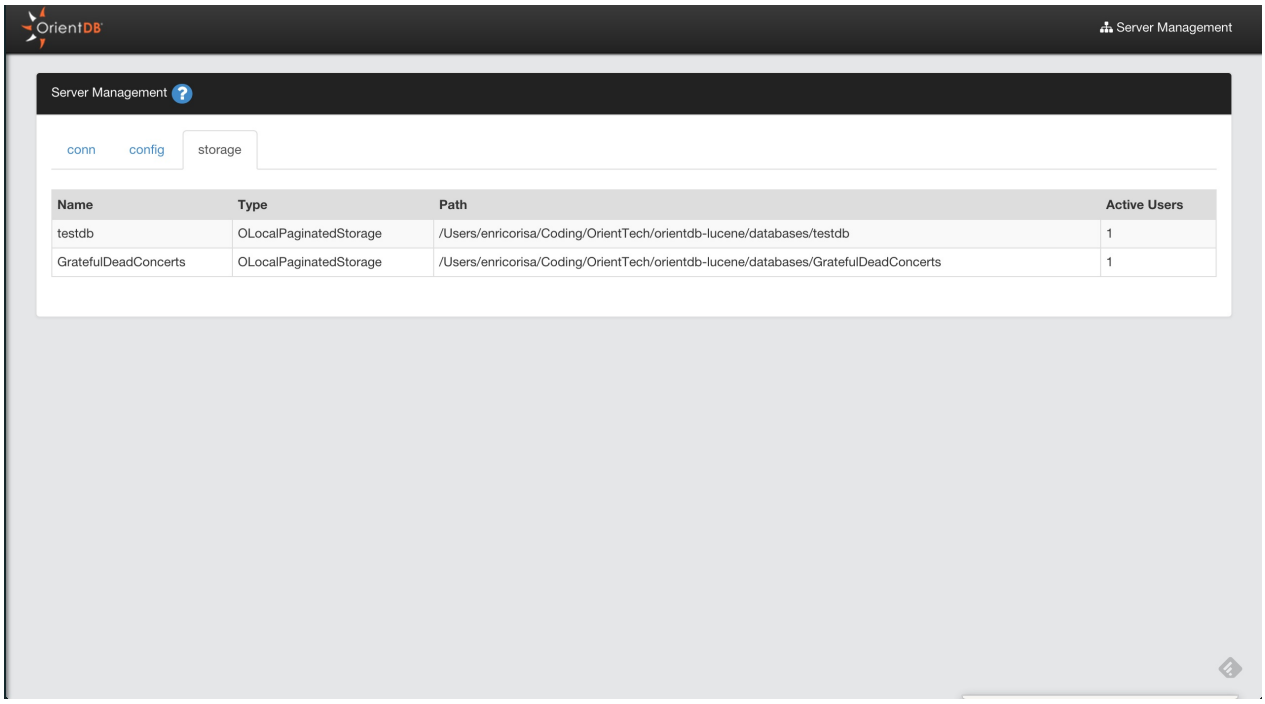
Connections



Configuration



Storage


OrientDB
Server Management
Server Management
conn
config
storage
Name
Type
Path
Active Users
testdb
OLocalPaginatedStorage
/Users/enricorisa/Coding/OrientTech/orientdb-lucene/databases/testdb
1
GratefulDeadConcerts
OLocalPaginatedStorage
/Users/enricorisa/Coding/OrientTech/orientdb-lucene/databases/GratefulDeadConcerts
1

Console Tool

The OrientDB Console is a Java Application made to work against OrientDB databases and Server instances.

Interactive mode

This is the default mode. Just launch the console by executing the script **bin/console.sh** (or **bin/console.bat** in MS Windows systems). Assure to have execution permission on it.

Once started the console is ready to accepts commands.

```
OrientDB console v.1.6.6 www.orientechnologies.com
Type 'help' to display all the commands supported.

orientdb>
```

To know all the supported commands look to commands.

Batch mode

To execute commands in batch mode run the **bin/console.sh** (or **bin/console.bat** in MS Windows systems) script passing all the commands separated with semicolon ";". Example:

```
> console.bat "connect remote:localhost/demo;select * from profile"
```

Or call the console script passing the name of the file in text format containing the list of commands to execute. Commands must be separated with semicolon ";". Example:

```
orientdb> console.bat commands.txt
```

In batch mode you can ignore errors to let the script to continue the execution by setting the "ignoreErrors" variable to true:

```
orientdb> set ignoreErrors true
```

Enable echo

When you run console commands in pipeline, you could need to display them. Enable "echo" of commands by setting it as property at the beginning:

```
orientdb> set echo true
```

Console commands

To know all the commands supported by the Orient console open it and type **help** or **?**.

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
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 use the release database to release it

get	Returns the value of a property	
grant	Grants a permission to a user	
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	
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	
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	
show holes	Displays the database's holes	

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

Extend the console with custom command

Edit the OConsoleDatabaseApp class and add a new method. There's an auto discovering system that put the new method between the available commands. To provide a description of the command use the annotations (look below). The command name must follow the Java code convention where to separate works just use the Camel-case.

So, for example, if you want to create the brand new "move cluster" command:

```
@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
  @ConsoleParameter(name = "target-path", description = "path of the new position where to m

  checkCurrentDatabase(); // THE DB MUST BE OPENED

  System.out.println("Moving cluster '" + iClusterName + "' to path " + iNewPath + "...");
}
```

If you type:

```
orientdb> help
```

Your new command will appear. And now try:

```
orientdb> move cluster foo /temp

Moving cluster 'foo' to path /temp...
```

Don't miss to contribute your command to the OrientDB Community! ;-)

Console - BACKUP

Executes a complete backup against the currently opened database. The backup file is compressed using the ZIP algorithm. To restore the database use the Restore Database command. Backup is much faster than Export Database. Look also to Export Database and Import Database commands. Backup can be done automatically by enabling the Automatic-Backup Server plugin.

NOTE: *Backup of remote databases is not supported in Community Edition, but only in Enterprise Edition. If you're using the Enterprise Edition look at Remote Backup.*

Syntax

```
backup database <output-file> [-compressionLevel=<compressionLevel>] [-bufferSize=<bufferSiz
```

Where:

- **output-file** is the output file path
- **compressionLevel** the compression level between 0 and 9. Default is 9. Since v1.7.
- **bufferSize** the compression buffer size. Default is 1MB. Since v1.7.

Example

```
orientdb> connect plocal:../databases/mydatabase admin admin
orientdb> backup database /backups/mydb.zip

Backuping current database to: database mydb.zip...

Backup executed in 0,52 seconds
```

Backup API

Backup can be executed in Java and any language on top of the JVM by using the method backup() against the database instance:

```
db.backup(out, options, callable, listener, compressionLevel, bufferSize);
```

Where:

- **out**: OutputStream used to write the backup content. Use a FileOutputStream to make the backup persistent on disk
- **options**: Backup options as Map object
- **callable**: Callback to execute when the database is locked iListener: Listener called for backup messages
- **compressionLevel**: ZIP Compression level between 0 (no compression) and 9 (maximum). The bigger is the compression, the smaller will be the final backup content, but will consume more CPU and time to execute
- **bufferSize**: Buffer size in bytes, the bigger is the buffer, the more efficient will be 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);
    }
  };

  OutputStream out = new FileOutputStream("/temp/mydb.zip");
  db.backup(out,null,null,listener,9,2048);
} finally {
   db.close();
}
```

See also

- Restore Database
- Export Database
- Import Database
- Console-Commands
- ODatabaseExport Java class

Console - BEGIN

OrientDB supports Transactions. To begin a new transaction use the **begin** command. Once a transaction is begun to make persistent the changes you have to call the commit command. To abort the changes call rollback command instead.

Syntax

```
begin
```

See also

- Transactions
- Console-Command-Commit
- Console-Command-Rollback
- Console-Commands

Example

```
orientdb> begin
Transaction 1 is running

orientdb> begin
Error: an active transaction is currently open (id=1). Commit or rollback before starting a

orientdb> insert into account (name) values ('tx test')

Inserted record 'Account#9:-2{name:tx test} v0' in 0,004000 sec(s).

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).
```

Until the commit all the new records will have a temporary RID with negative numbers.

Console - BROWSE CLASS

This command displays all the records of a class.

Syntax

```
browse class <class-name>
```

Where:

- class-name The name of the class

Example

```
> browse class City

---+--------+------------------
  #| REC ID |NAME
---+--------+------------------
  0|    -6:0|Rome
  1|    -6:1|London
  2|    -6:2|Honolulu
---+--------+------------------
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - BROWSE CLUSTER

This command displays all the records of a cluster.

Syntax

```
browse cluster <cluster-name>
```

Where:

- cluster-name The name of the cluster

Example

```
> browse cluster City

---+--------+------------------
  #| REC ID |NAME
---+--------+------------------
  0|    -6:0|Rome
  1|    -6:1|London
  2|    -6:2|Honolulu
---+--------+------------------
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CLASSES

Displays all the classes configured in the current database.

Syntax

```
classes
```

Example

```
> 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 |
-------------------------------------------------------------------------------+
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CLUSTERS

Displays all the clusters configured in the current database.

Syntax

```
clusters
```

Example

```
> 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 |
-----------------------------------------------------------+
```

See also

To create a new cluster in the current database use the command create cluster.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - COMMIT

OrientDB supports Transactions. To begin a new transaction use the **begin** command. Once a transaction is begun to make persistent the changes you have to call the commit command. To abort the changes call rollback command instead.

Syntax

```
commit
```

See also

- Transactions
- Console Command Begin
- Console Command Rollback
- Console Commands

Example

```
orientdb> begin
Transaction 2 is running

orientdb> begin
Error: an active transaction is currently open (id=2). Commit or rollback before starting a

orientdb> insert into account (name) values ('tx test')

Inserted record 'Account#9:-2{name:tx test} v0' in 0,000000 sec(s).

orientdb> commit
Transaction 2 has been committed in 4ms

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).

orientdb>
```

Until the commit all the new records will have a temporary RID with negative numbers.

Console - CONFIG

Displays the configuration where the opened database is located (local or remote)

Syntax

```
config
```

Example

```
> 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                           |
| network.timeout                   = 10000                          |
| file.mmap.forceDelay              = 500                            |
| profiler.enabled                  = false                          |
| network.socketBufferSize          = 32768                          |
| treemap.optimizeThreshold         = 50000                          |
| file.mmap.blockSize               = 300000                         |
| network.retryDelay                = 500                            |
| network.retry                     = 5                              |
| treemap.entryPoints               = 30                             |
+-----------------------------------+--------------------------------+
```

See also

To change a configuration value use the config set.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CONFIG GET

Returns the value of the requested configuration value.

Syntax

```
config get <config-name>
```

Where:

- config-name Name of the configuration

Example

```
> config get db.cache.enabled

Remote configuration: db.cache.enabled = false
```

See also

To display the entire configuration use the config.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CONFIG SET

Changes the value of a property.

Syntax

```
config set <config-name> <config-value>
```

Where:

- config-name Name of the configuration to change
- config-value Value to set

Example

```
> config set db.cache.enabled false

Remote configuration value changed correctly
```

See also

To know all the configuration values use the config. To read a configuration value use the config get.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CONNECT

Opens a database using a URL.

Syntax

```
connect <database-url> <user-name> <user-password>
```

Where:

- **database-url** The url of the database to connect in the format `<mode>:<path>`
- **user** The user name
- **user-password** The user password

Example: connect to a local database

To connect to a local database loading it directly into the console.

Example:

```
> connect plocal:../databases/GratefulDeadConcerts admin admin
```

Example: Connect to a remote database

To connect to a local or remote database by using a Orient Server.

Example:

```
> connect remote:127.0.0.1/GratefulDeadConcerts admin admin
```

```
> connect plocal:../databases/GratefulDeadConcerts admin

Connecting to database [plocal:../databases/GratefulDeadConcerts]...OK
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CREATE CLUSTER

Creates a new cluster in the current database. The cluster can be "physical" or "memory".

Syntax

```
create cluster <cluster-name> <cluster-type> <data-segment> <location> <position>
```

Where:

- **cluster-name** The name of the cluster to create
- **cluster-type** Cluster type: 'physical' or 'logical'
- **data-segment** Data segment to use. 'default' will use the default one
- **location** Location where to place the new cluster files, if appliable. use 'default' to leave into the database directory
- **position** 'append' to add as last cluster, otherwise the empty position to replace

Example

```
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
```

See also

To display all the cluster configured in the current database use the command clusters.

To delete a cluster use the command Drop Cluster.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CREATE DATABASE

Creates a new database.

Syntax

```
create database <database-url> [<user> <password> <storage-type> [<db-type>]]
```

Where:

- *database-url* The url of the database to create in the format '`<mode>:<path>`'
- *user* on remote database is the Server's administrator name
- *password* on remote database is the Server's administrator password
- *storage-type* The type of the storage between 'plocal' for disk-based database and 'memory' for in memory only database. Look at Storage types.
- *db-type* Optional, is the database type between "graph" (the default) and "document"

See also

- Console Command Drop Database
- SQL Alter Database

Example: create a local database

```
> create database plocal:/usr/local/orient/databases/demo/demo

Creating database [plocal:/usr/local/orient/databases/demo/demo]...
Connecting to database [plocal:/usr/local/orient/databases/demo/demo]...OK
Database created successfully.

Current database is: plocal:/usr/local/orient/databases/demo/demo
```

Example: create a remote database

```
> create database remote:localhost/trick root E30DD873203AAA245952278B4306D94E423CF91D569881

Creating database [remote:localhost/trick ]...
Connecting to database [remote:localhost/trick ]...OK
Database created successfully.

Current database is: remote:localhost/trick
```

Create a static database into the server configuration

To create a static database to use it from the server look at: Server pre-configured storages.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CREATE INDEX

The **SQL Create Index** command creates an index on a property defined in the schema.

Indexes can be:

- **unique**, doesn't allow duplicated
- **not unique**, allows duplicates
- **full-text**, by indexing any single word of the text. It's used in query with the operator CONTAINSTEXT

Syntax

```
CREATE INDEX <name> [ON <class-name> (prop-names)] <type> [<key-type>]
```

Where:

- **name** logical name of index. Can be `<class>.<property>` to create an automatic index bound to a schema property. In this case **class** is the class of the schema and **property**, is the property created into the class. Notice that in another case index name can't contain '.' symbol
- **class-name** name of class that automatic index created for. Class with such name must already exist in database
- **prop-names** comma-separated list of properties that this automatic index is created for. Property with such name must already exist in schema
- **type**, between 'unique', 'notunique' and 'fulltext'
- **key-type**, is the type of key (Optional). On automatic indexes is auto-determined by reading the target schema property where the index is created. If not specified for manual indexes, at run-time during the first insertion the type will be auto determined by reading the type of the class.

Examples

```
CREATE INDEX users.Id unique
```

For more information look at Create index command.

For complete index guide look at Index guide.

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CREATE LINK

The **Create Link** command creates links between two or more records of type Document. This is very useful when you're importing data from a Relational database. In facts in the Relational world relationships are resolved as foreign keys.

Consider this example where the class "Post" has a relationship 1-N to "Comment":

```
Post 1 ---> * Comment
```

In a Relational database you'll have something like that:

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

Using OrientDB, instead, you have direct relationship as in your object model. So the navigation is from *Post* to *Comment* and not viceversa as for Relational model. For this reason you need to create a link as **INVERSE**.

Syntax

```
CREATE LINK <link-name> FROM <source-class>.<source-property> TO <destination-class>.<destin
```

Where:

- *link-name* is the name of the property for the link. If not expressed will be overwritten the *destination-property* field
- *source-class*, is the source class
- *source-property*, is the source property
- *destination-class*, is the destination class
- *destination-property*, is the destination property

Examples

CREATE LINK comments FROM comments.!PostId To posts.Id INVERSE

To know more about other SQL commands look at SQL SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - CREATE PROPERTY

The **SQL Create Property** command creates a new property in the schema. You need to create the class before.

Syntax

```
CREATE PROPERTY <class>.<property> <type>
```

Where:

- *class* is the class of the schema
- *property*, is the property created into the *class*
- *type*, the type of the property. It can be:
  - *boolean*
  - *integer*
  - *short*
  - *long*
  - *float*
  - *double*
  - *date*
  - *string*
  - *binary*
  - *embedded*
  - *embeddedlist*, this is a container and needs the parameter *linked-type* or *linked-class*
  - *embeddedset*, this is a container and needs the parameter *linked-type* or *linked-class*
  - *embeddedmap*, this is a container and needs the parameter *linked-type* or *linked-class*
  - *link*
  - *linklist*, this is a container and needs the parameter *linked-type* or *linked-class*
  - *linkset*, this is a container and needs the parameter *linked-type* or *linked-class*
  - *linkmap*, this is a container and needs the parameter *linked-type* or *linked-class*
  - *byte*

- *linked-type*, the contained type in containers (see above). It can be:
  - *boolean*
  - *integer*
  - *short*
  - *long*
  - *float*
  - *double*
  - *date*

    - *string*
    - *binary*
    - *embedded*
    - *link*
    - *byte*

- *linked-class*, the contained class in containers (see above).

Examples

Create the property 'name' of type 'STRING' in class 'User':

```
CREATE PROPERTY user.name STRING
```

Create a list of Strings as property 'tags' of type 'EMBEDDEDLIST' in class 'Profile'. The linked type is 'STRING':

```
CREATE PROPERTY profile.tags EMBEDDEDLIST STRING
```

Create the property 'friends' of type 'EMBEDDEDMAP' in class 'Profile'. The linked class is profile itself (circular references):

```
CREATE PROPERTY profile.friends EMBEDDEDMAP Profile
```

To remove a property use the SQLRemoveProperty Remove Property command.

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - DECLARE INTENT

Declares an intent on current database. Intents are a way to tell to OrientDB what you're going to do.

Syntax

```
declare intent <intent-name>
```

Where:

- intent-name The name of the intent. "null" means remove the current intent. Supported ones are:
- massiveinsert
- massiveread

Example

```
> declare intent massiveinsert
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - DICTIONARY GET

Displays the value of the requested key loaded from the database dictionary.

Syntax

```
dictionary get <key>
```

Where:

- key The key to search

Example

```
> 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:Ob
,name:Natasha,surname:Obama,city:null}]
                name : Barack
             surname : Obama
                city : City@-6:2{name:Honolulu}
---------------------------------------------------
```

To know all the keys stored in the database dictionary use the dictionary keys command.

For complete index (and dictionary) guide look at Index guide.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - DICTIONARY KEYS

Displays all the keys stored in the database dictionary.

Syntax

```
dictionary keys
```

Example

```
> dictionary keys

Found 4 keys:
#0: key-148
#1: key-147
#2: key-146
#3: key-145
```

To load the associated record use the dictionary get `<key>`.

For complete index (and dictionary) guide look at Index guide.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - DISCTIONARY PUT

Associates in the database dictionary a record to a key to be found later using a dictionary get command.

Syntax

```
dictionary put <key> <record-id>
```

Where:

- key The key to bind
- record-id The record-id of the record to bind to the key passes

Example

```
> 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:Ob
,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 know all the keys stored in the database dictionary use the dictionary keys command.

For complete index (and dictionary) guide look at Index guide.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - DICTIONARY REMOVE

Removes the association from the database dictionary.

Syntax

```
dictionary remove <key>
```

Where:

- key The key to remove

Example

```
> 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:Ob
,name:Natasha,surname:Obama,city:null}]
                name : Barack
             surname : Obama
                city : City@-6:2{name:Honolulu}
----------------------------------------------------
```

To know all the keys stored in the database dictionary use the dictionary keys command.

For complete index (and dictionary) guide look at Index guide.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - DISCONNECT

Closes the current opened database.

Syntax

```
disconnect
```

Example

```
> disconnect

Disconnecting from the database [../databases/petshop/petshop]...OK
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - DISPLAYS RECORD

Displays the details of the record of the last result set returned. This command needs the relative position of the record in the result set.

Syntax

```
display [<number>](<number>.md)record])
```

Where:

- number The number of the record in the last result set

Example

```
> select * from person

---+--------+--------------------+--------------------+--------------------+---------------
  #| REC ID |PARENT              |CHILDREN            |NAME                |SURNAME
---+--------+--------------------+--------------------+--------------------+---------------
  0|     5:0|null                |null                |Giuseppe            |Garibaldi
  1|     5:1|5:0                 |null                |Napoleone           |Bonaparte
  2|     5:2|5:3                 |null                |Nicholas            |Churcill
  3|     5:3|5:2                 |null                |Winston             |Churcill
  4|     5:4|null                |[2]                 |Barack              |Obama
  5|     5:5|5:4                 |null                |Malia Ann           |Obama
  6|     5:6|5:4                 |null                |Natasha             |Obama
---+--------+--------------------+--------------------+--------------------+---------------
7 item(s) found. Query executed in 0.038 sec(s).

> display record 4
--------------------------------------------------
Class: Person   id: 5:5   v.0
--------------------------------------------------
              parent : Person@5:4{parent:null,children:[Person@5:5, Person@5:6],name:Barack,
            children : null
                name : Malia Ann
             surname : Obama
                city : null
--------------------------------------------------
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - DROP CLUSTER

The **Drop Cluster** command definitely deletes a cluster. This will delete the cluster, all its records and will clear all caches. *NOTE: Unless you've made backups there is no way to restore a deleted cluster.*

Syntax

```
DROP CLUSTER <cluster-name>
```

Where:

- **cluster-name** is the name of the cluster.

Examples

Delete the current local database:

```
DROP CLUSTER Person
```

deletes the cluster named 'Person' with all Person records.

To create a new cluster use the Create Cluster command.

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - DROP DATABASE

The **Drop Database** command definitely deletes a database. If a database is open and no database name is used, then the current database will be deleted. *NOTE: Unless you've made backups there is no way to restore a deleted database.*

Syntax

For database opened using "local" protocol:

```
DROP DATABASE
```

To remove a database hosted in a remote OrientDB Server you need the credential to do it at the target OrientDB server:

```
DROP DATABASE <database-name> <server-username> <server-userpassword>
```

Where:

- **database-name** is the name of database. If not specified means the current database if it's opened
- **server-username** is the name of the server's user with privileges to drop the database
- **server-userpassword** is the password of the server's user

See also

- Console Command Create Database
- SQL Alter Database

Examples

Delete the current local database:

```
DROP DATABASE
```

Delete the remote database "demo" hosted on localhost:

```
DROP DATABASE remote:localhost/demo root 5B1A917B20C78ECAA219E37CFDDA6598D4D62CE68DD82E5B05D
```

To create a new database use the Create Database command.

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - EXPORT

Exports the current opened database to a file. The exported file is in JSON format using the Export-Format. By default the file is compressed using the GZIP algorithm. The Export/Import commands allow to migrate the database between different releases of OrientDB without loosing data. If you receive an error about the database version, export the database using the same version of OrientDB that has generated the database.

Export doesn't lock your database, but browses it. This means that concurrent operation can be executed during the export, but the exported database couldn't be the exact replica when you issued the command because concurrent updates could occurs. If you need a snapshot of database at a point in a time, please use Backup.

Once exported, use the Import to restore it. The database will be imported and will be ready to be used. Look also to Backup Database and Restore Database commands.

Syntax

By default the export command exports the full database, but there are some flags to disable some parts.

```
export database <output-file>
      [-excludeAll]
      [-includeClass=<class-name>*]
      [-excludeClass=<class-name>*]
      [-includeCluster=<cluster-name>*]
      [-excludeCluster=<cluster-name>*]
      [-includeInfo=<true|false>]
      [-includeClusterDefinitions=<true|false>]
      [-includeSchema=<true|false>]
      [-includeSecurity=<true|false>]
      [-includeRecords=<true|false>]
      [-includeIndexDefinitions=<true|false>]
      [-includeManualIndexes=<true|false>]
      [-compressionLevel=<0-9>]
      [-compressionBuffer=<bufferSize>]
```

Where:

- **output-file** is the output file path
- **-excludeAll** exclude everything. This is useful to export only few things. Instead of exclude all the feature it's much easier exclude all, and include what you're interested. Example: "-excludeAll -includeSchema" to export the schema only. Available since v1.7.
- **-includeClass** includes few classes to export. Class names must be separated by spaces
- **-excludeClass** excludes few classes to export. Class names must be separated by spaces
- **-includeCluster** includes few clusters to export. Cluster names must be separated by spaces
- **-excludeCluster** excludes few clusters to export. Cluster names must be separated by spaces
- **-includeInfo** includes or not database's information
- **-includeClusterDefinitions** includes or not definitions of clusters
- **-includeSchema** includes or not the database's schema
- **-includeSecurity** includes or not database's security
- **-includeRecords** includes or not record contents
- **-includeIndexDefinitions** includes or not database's index definition
- **-includeManualIndexes** includes or not manual index contents

- **-compressionLevel** set the compression level between 0 (=no compression) and 9 (maximum compression). Default is 1 (since 1.7.6)
- **-compressionBuffer** Set the buffer size in bytes used by compression. By default is 16Kb (since 1.7.6)

Examples

Export the entire database

```
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 database's functions only

```
orientdb> export database functions.gz -includeClass=OFunction
                 -includeInfo=false
                 -includeClusterDefinitions=false
                 -includeSchema=false
                 -includeIndexDefinitions=false
                 -includeManualIndexes=false
```

Export API

Export command can be used in Java and any language on top of the JVM by using the class ODatabaseExport. 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();
}
```

See also

- Export File Format
- Import Database
- Backup Database
- Restore Database
- Console Commands
- ODatabaseExport Java class

Console - EXPORT RECORD

This command exports the current record in the format requested. The format must be between the supported ones. In case of error the supported format list will be displayed.

Syntax

```
export record <format>
```

Example

```
> export record json
{
  'parent': null,
  'children': [5:5, 5:6],
  'name': 'Barack',
  'surname': 'Obama',
  'city': -6:2
}
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - FREEZE DATABASE

Flushes all cached content to the disk storage and allows to perform only read commands. Database will be "frozen" till release database command will not been executed.

This command requires presence of server administration rights and can be executed only on remote DBs. If you would like to freeze/release local DB use methods `ODatabase.freeze()` and `ODatabase.release()` directly from OrientDB API.

This command is very useful in case you would like to do "live" database backups. You can "freeze" database, do file system snapshot, "release" database, copy snapshot anywhere you want. Using such approach you can perform backup in short term.

Syntax

```
FREEZE DATABASE
```

See also

- release database, to release the frozen database
- SQL commands
- Console-Commands

Examples

Freezes the current database:

```
FREEZE DATABASE
```

Console - GET

Returns the value of the requested property

Syntax

```
get <property-name>
```

Where:

- property-name Name of the property

Example

```
> get limit

limit = 20
```

See also

To know all the properties setted use the properties.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - GRANT

The **SQL Grant** command changes the permission of a role granting the access to one or more resources.

Syntax

```
GRANT <permission> ON <resource> TO <role>
```

Where:

- **permission** can be:
- NONE, no permission
- CREATE, to create the indicated resource
- READ, to read the indicated resource
- UPDATE, to update the indicated resource
- DELETE, to delete the indicated resource
- ALL, all permissions
- **resource**, the target resource where to change the permissions
- database, as the access to the whole database
- database.class, as the access to the records contained in a class. Use `**` to indicate all the classes
- database.cluster, as the access to the records contained in a cluster. Use `**` to indicate all the clusters
- database.query, as the ability to execute query (READ is enought)
- database.command, as the ability to execute SQL commands. CREATE is for SQL-Insert, READ is for SQL SELECT, UPDATE for SQL-Update and DELETE is for SQL-Delete
- database.config, as the ability to access to the configuration. Valid permissions are READ and UPDATE
- database.hook.record, as the ability to set hooks
- server.admin, as the ability to access to the server resources
- **role**, the role name

Examples

Grant the permission to *update* any records in *cluster Account* to the *role "backoffice"*.

```
GRANT update ON database.cluster.Account TO backoffice
```

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - IMPORT

Imports a database to the current one opened. The input file is in JSON format using the Export-Format generated by Console Command Export tool. By default the file is compressed using the GZIP algorithm. The Export/Import commands allow to migrate between engine releases without loosing data. Use the Export command to generate the JSON file to import. Look also to Backup Database and Restore Database commands.

Syntax

```
import database <input-file> [-preserveClusterIDs = <true|false>]
                             [-merge = <true|false>]
                             [-migrateLinks = <true|false>]
                             [-rebuildIndexes = <true|false>]
```

Where:

- **input-file**: input file path
- **-preserveClusterIDs**: allows to keep the same cluster ids during import. Valid only for plocal storage. Import tool usually creates temporary clusters to keep cluster ids the same, but this approach fails some times so if you use plocal storage it is recommended to set this parameter during DB import.
- **-merge**: merges the database to import into the current one. Security classes (ORole, OUser and OIdentity) are always preserved. By default is false that means overwrite current database (but security classes). Since 1.6.1.
- **-migrateLinks**: Migrate links after import. This is needed to update all the references to the new RID. By default is true that means all the links are updated. Set it to false to speed up the importing only when -merge=true and if you're very sure no other existent records are linking the records you're importing. Since 1.6.1.
- **-rebuildIndexes**: Rebuild indexes after import. By default is true that means all the indexes are rebuilt. Set it to false to speed up the importing only when you're very sure indexes aren't impacted from import. Since 1.6.1.

See also

- Export File Format
- Export Database
- Backup Database
- Restore Database
- Console-Commands
- ODatabaseImport Java class

Example

```
> 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
```

Troubleshooting

If during the importing you experience that "Imported cluster 'XXX' has id=6 different from the original: 5" means that your database was created with an ancient version of OrientDB:

```
- Creating cluster 'company'...Error on database import happened just before line 16, column
com.orientechnologies.orient.core.exception.OConfigurationException: Imported cluster 'compa
        at com.orientechnologies.orient.core.db.tool.ODatabaseImport.importClusters(ODatabas
        at com.orientechnologies.orient.core.db.tool.ODatabaseImport.importDatabase(ODatabas
```

To fix it just drop the ORIDs class before to import the database:

```
orientdb> drop class ORIDs
orientdb> import database ...
```

Import API

Import command can be used in Java and any language on top of the JVM by using the class ODatabaseImport. 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);
    }
  };

  ODatabaseImport import = new ODatabaseImport(db, "/temp/export/export.json.gz", listener);
  import.importDatabase();
  import.close();
} finally {
  db.close();
}
```

Console - INFO

Displays all the information about the current database.

Syntax

```
info
```

Example

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

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - INFO CLASS

Displays all the information about the selected class

Syntax

```
info class <class-name>
```

Example

```
> info class profile

Class.................: Profile (id=4)
Default cluster......: profile (id=10)
Supported cluster ids: [10]
Properties:
-------------------------------+----+-------------+-------------------------------+---------
 NAME                          | ID | TYPE        | LINKED TYPE/CLASS             | INDEX
-------------------------------+----+-------------+-------------------------------+---------
 lastAccessOn                  |  5 | DATETIME    | null                          |
 registeredOn                  |  4 | DATETIME    | null                          |
 nick                          |  3 | STRING      | null                          |
 name                          |  2 | STRING      | null                          | NOTUNIQU
 surname                       |  1 | STRING      | null                          |
 photo                         |  0 | TRANSIENT   | null                          |
-------------------------------+----+-------------+-------------------------------+---------
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - insert

Insert

Insert a new record into the current database. Remember that Orient can work also in schema-less mode, so you can create any field at-the-fly.

Syntax

```
insert into <class|cluster:<cluster>> (<field-name>*) values ( <field-value> )
```

Example

nsert a new record with name 'Jay' and surname 'Miner':

```
> insert into Profile (name, surname) values ('Jay', 'Miner' )

Inserted record in 0,060000 sec(s).
```

Insert a new record adding a relationship:

```
insert into Employee (name, boss) values ('jack', 11:99 )
```

Insert a new record adding a collection of relationship:

```
insert into Profile (name, friends) values ('Luca', [10:3, 10:4] )
```

To know more about the SQL syntax used in Orient take a look to: SQL-Query.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - LOAD RECORD

Loads a record by its record-id from the current database.

Syntax

```
load record <record-id>
```

Where:

- record-id The unique Record Id of the record to load. If you don't have the Record Id execute a query first

Example

```
> 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,
            children : null
                name : Malia Ann
             surname : Obama
                city : null
----------------------------------------------------
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - PROFILER

Controls the Profiler.

Syntax

```
profiler on|off|dump|reset
```

Where:

- on Turn on the profiler and start recording
- off Turn off the profiler and stop recording
- dump Dump profiler's data
- reset Reset profiler's data

Example

```
orientdb> profiler on
orientdb> profiler dump
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - PROPERTIES

Returns all the properties setted.

Syntax

```
properties
```

Example

```
> properties

PROPERTIES:
+---------------------+----------------------+
| NAME                | VALUE                |
+---------------------+----------------------+
| limit               = 20                   |
+---------------------+----------------------+
```

See also

To change a property value use the set.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - RELEASE DATABASE

Switches database from "frozen" state (where only read operations are allowed) to normal mode.

Execution of this command requires presence of server administration rights.

This command is very usefull in case you would like to do "live" database backups. You can "freeze" database, do file system snapshot, "release" database, copy snapshot anywhere you want. Using such approach you can perform backup in short term.

Syntax

```
RELEASE DATABASE
```

See also

- Freeze Database, to freeze a database
- SQL commands
- Console-Commands

Examples

Release the current database:

```
RELEASE DATABASE
```

Console - RELOAD RECORD

Reloads a record by its record-id from the current database ignoring the cache. This is useful when external applications change the record and you need to see latest update.

Syntax

```
reload record <record-id>
```

Where:

- record-id The unique Record Id of the record to reload. If you don't have the Record Id execute a query first

Example

```
> 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,
            children : null
                name : Malia Ann
             surname : Obama
                city : null
--------------------------------------------------
```

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - RESTORE

Executes a restore of current opened database. The backup file is created using the Backup Database command. Look also to Export Database and Import Database commands.

Syntax

```
restore database <backup-file>
```

Where:

- *backup-file* is the backup input file path to restore

Example

```
orientdb> restore database /backups/mydb.zip

Restore executed in 6,33 seconds
```

Restore API

Restore can be executed in Java and any language on top of the JVM by using the method restore() against the database instance:

```
db.restore(in, options, callable, listener);
```

Where:

- **in**: InputStream used to read the backup content. Use a FileInputStream to read the backup content from disk
- **options**: Backup options as Map object
- **callable**: Callback to execute when the database is locked iListener: Listener called for backup messages
- **compressionLevel**: ZIP Compression level between 0 (no compression) and 9 (maximum). The bigger is the compression, the smaller will be the final backup content, but will consume more CPU and time to execute
- **bufferSize**: Buffer size in bytes, the bigger is the buffer, the more efficient will be 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();
}
```

See also

- Backup Database
- Export Database
- Import Database
- Console-Commands

Console - REVOKE

The **SQL Revoke** command changes the permission of a role revoking the access to one or more resources.

Syntax

```
REVOKE <permission> ON <resource> FROM <role>
```

Where:

- **permission** can be:
- NONE, no permission
- CREATE, to create the indicated resource
- READ, to read the indicated resource
- UPDATE, to update the indicated resource
- DELETE, to delete the indicated resource
- ALL, all permissions
- **resource**, the target resource where to change the permissions
- database, as the access to the whole database
- database.class, as the access to the records contained in a class. Use `**` to indicate all the classes
- database.cluster, as the access to the records contained in a cluster. Use `**` to indicate all the clusters
- database.query, as the ability to execute query (READ is enought)
- database.command, as the ability to execute SQL commands. CREATE is for SQL-Insert, READ is for SQL SELECT, UPDATE for SQL-Update and DELETE is for SQL-Delete
- database.config, as the ability to access to the configuration. Valid permissions are READ and UPDATE
- database.hook.record, as the ability to set hooks
- server.admin, as the ability to access to the server resources
- **role**, the role name

Examples

Revoke the permission to *delete* any records in any *cluster* to the *role "backoffice"*.

```
REVOKE delete ON database.cluster.* TO backoffice
```

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Console - ROLLBACK

OrientDB supports Transactions. Once a transaction is begun you can abort changes in transactions by using the **rollback** command.

Syntax

```
rollback
```

See also

- Transactions
- Console Command Commit
- Console Command Rollback
- Console Commands

Example

```
orientdb> begin
Transaction 1 is running

orientdb> begin
Error: an active transaction is currently open (id=1). Commit or rollback before starting a

orientdb> insert into account (name) values ('tx test')

Inserted record 'Account#9:-2{name:tx test} v0' in 0,004000 sec(s).

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).

orientdb> rollback
Transaction 1 has been rollbacked in 4ms

orientdb> select from account where name like 'tx%'

0 item(s) found. Query executed in 0.037 sec(s).
```

Console - SET

Changes the value of a property.

Syntax

```
set <property-name> <property-value>
```

Where:

- property-name Name of the property
- property-value Value to set

Example

```
> set limit 100

Previous value was: 20

limit = 100
```

See also

To know all the properties setted use the properties. To read the property value use the get.

This is a command of the Orient console. To know all the commands go to Console-Commands.

Operations

This is the main page for DBA and DevOps.

Tuning

- Performance Tuning

In & Out

Install

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 by providing Enterprise features such as:
  - Query Profiler
  - Distributed Clustering configuration
  - Metrics Recording
  - Live Monitoring with configurable Alerts

An Enterprise Edition license is included without charge if you purchase Support.

Prerequisites

Both editions run on every operating system that has an implementation of the Java Virtual Machine (JVM), for example:

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

This means the only requirement for using OrientDB is to have Java version 1.6 or higher installed.

Download Binaries

The easiest and fastest way to start using OrientDB is to download binaries from the Official OrientDB Download Page.

Compile Your Own Community Edition

Alternatively, you can clone the Community Edition project from GitHub and compile it. This allows you access to the latest functionality without waiting for a distribution binary. To build the Community Edition, you must first install the Apache Ant tool and follow these steps:

```
> git clone git@github.com:orientechnologies/orientdb.git
> cd orientdb
> ant clean install
```

After the compilation, all the binaries are placed under the `../releases` directory.

Change Permissions

The Mac OS X, Linux, and UNIX based operating systems typically require you to change the permissions to execute scripts. The following command will apply the necessary permissions for these scripts in the `bin` directory of the OrientDB distribution:

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

Use inside of OSGi container

OrientDB uses a ConcurrentLinkedHashMap implementation provided by https://code.google.com/p/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
```

Other Resources

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

- Install as service on Unix, Linux and MacOSX
- Install as service on Windows

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

orientdb-docker

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. Checkout this project to a local folder cding to it
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 my own image 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
```

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/orient
docker run -d --name orientdb_databases -v /opt/orientdb/databases nesrait/btsync /opt/
docker run -d --name orientdb_backup -v /opt/orientdb/backup nesrait/btsync /opt/orient
```

4. Wait until all files have magically appeared inside your BTSync data volumes:

```
docker run --rm -i -t --volumes-from orientdb_config --volumes-from orientdb_databases
```

5. Finally you're ready to start your OrientDB server

```
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/
```

- 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
```

Install as Service on Unix/Linux

OrientDB is shipped with the script ***$ORIENTDB_HOME/bin/orientdb.sh*** that can be used to run OrientDB like a daemon. It supports the following parameters:

- start
- stop
- status

Before to install it as service open the file and change the following lines:

```
ORIENTDB_DIR="YOUR_ORIENTDB_INSTALLATION_PATH"
ORIENTDB_USER="USER_YOU_WANT_ORIENTDB_RUN_WITH"
`
```

By setting the installation path and the user as stated, save the script, and deploy like other scripts for the other daemon.

Different Unix, Linux and MacOSX distribution uses different ways to manage the start/stop process at the system bootstrap/shutdown.

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 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 shutted 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 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 the problem 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 that 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 GraphEd 1.0rc5 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_SETTINGS=-Dprofiler.enabled=true -Dcache.level1.enabled=false -Dcache.level2.st
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
--StartClass=com.orientechnologies.orient.server.OServerMain --StopClass=com.orientechnologi
--Classpath="%ORIENTDB_HOME%\lib\*" --JvmOptions "-Djava.util.logging.config.file="%LOG_FILE
--StartMode=jvm --StartPath="%ORIENTDB_HOME%\bin" --StopMode=jvm --StopPath="%ORIENTDB_HOME%

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	The class invoked in the */bin/shutdown.bat* script

	to be called is the `main` method	
--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

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
- Native Graph Database performance tuning (Deprecated)

Configuration

To tune OrientDB look at the Configuration settings.

Platforms

- Performance analysis on ZFS

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:+AggressiveOpts -XX:CompileThreshold=200
```

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 t
```

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.

Keep the database small

The smaller the database you have, the bigger are the number of records you can cache in memory. Furthermore small database means faster seek in filesystem and minor loading time from disk/network. In order to keep your database small follow the following suggestions:

Keep field names short

OrientDB is schema-less that means field names are stored with the values too. So if you call a field "out" instead of "outVertices" you saves 8 characters, namely 8 bytes per record. Applying this to millions of records allows you to save several Megabytes.

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

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Ser 64bi
`environment.dumpCfgAtStartup`	true	false	true	false
`environment.concurrent`	true	true	true	true

Memory

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Server 64bit
`memory.optimizeThreshold`	0.85	0.85	0.85	0.85

Storage

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Serv 64bit

`storage.keepOpen`	true	true	true	true
`storage.record.lockTimeout`	5000	5000	5000	5000

Cache

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Server 64bit
`cache.level1.enabled`	true	true	false	false
`cache.level1.size`	-1	-1	0	0
`cache.level2.enabled`	true	true	false	false
`cache.level2.size`	-1	-1	0	0

Database

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Server 64bit
`db.mvcc`	true	true	true	true
`object.saveOnlyDirty`	false	false	false	false
`nonTX.recordUpdate.synch`	false	false	false	false

Transactions

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Server 64bit	Allo in
`tx.useLog`	true	true	true	true	true fals
`tx.log.fileType`	classic	classic	classic	classic	'cla or 'mm

`tx.log.synch`	false	false	false	false	true fals
`tx.commit.synch`	false	false	true	true	true fals

TinkerPop Blueprints

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Server 64bit
`blueprints.graph.txMode`	0	0	0	0

Index

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	D
`index.auto.rebuildAfterNotSoftClose`	true	true	true	tr

MVRB Tree (index and dictionary)

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	De

`mvrbtree.lazyUpdates`	20000	20000	1	1
`mvrbtree.nodePageSize`	128	128	128	12
`mvrbtree.loadFactor`	0.7f	0.7f	0.7f	0.
`mvrbtree.optimizeThreshold`	100000	100000	100000	10
`mvrbtree.entryPoints`	16	16	16	16
`mvrbtree.optimizeEntryPointsFactor`	1.0f	1.0f	1.0f	1.
`mvrbtree.ridBinaryThreshold`	8	8	8	8
`mvrbtree.ridNodePageSize`	16	16	16	16
`mvrbtree.ridNodeSaveMemory`	False	False	False	Fa

Lazy Collections

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Server 64bit
`lazyset.workOnStream`	true	true	false	false

File (I/O)

Parameter	Def. 32bit	Def. 64bit	Def. Server 32bit
`file.lock`	false	false	false
`file.defrag.strategy`	0	0	0
`file.defrag.holeMaxDistance`	32768 (32Kb)	32768 (32Kb)	32768 (32Kb)
`file.mmap.useOldManager`	false	false	false
`file.mmap.lockMemory`	true	true	true

`file.mmap.strategy`	0	0	0
`file.mmap.blockSize`	1048576 (1Mb)	1048576 (1Mb)	1048576 (1Mb)
`file.mmap.bufferSize`	8192 (8Kb)	8192 (8Kb)	8192 (8Kb)
`file.mmap.maxMemory`	134Mb	(maxOsMem - maxProcessHeapMem) / 2	like Def. 32 bit
`file.mmap.overlapStrategy`	2	2	2

`file.mmap.forceDelay`	500 (0.5sec)	500 (0.5sec)	500 (0.5sec)
`file.mmap.forceRetry`	20	20	20

JNA

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Serv 64bit
`jna.disable.system.library`	true	true	true	true

Networking (I/O)

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	D
`network.socketBufferSize`	32768	32768	32768	3
`network.lockTimeout`	15000 (15secs)	15000 (15secs)	15000 (15secs)	1 (
`network.socketTimeout`	10000 (10secs)	10000 (10secs)	10000 (10secs)	1 (
`network.retry`	5	5	5	5

`network.retryDelay`	500 (0.5sec)	500 (0.5sec)	500 (0.5sec)	5 (
`network.binary.maxLength`	100000 (100Kb)	100000 (100Kb)	100000 (100Kb)	1 (
`network.binary.readResponse.maxTime`	30	30	30	3
`network.binary.debug`	false	false	false	fa
`network.http.maxLength`	100000 (100Kb)	100000 (100Kb)	100000 (100Kb)	1 (
`network.http.charset`	utf-8	utf-8	utf-8	u
`network.http.sessionExpireTimeout`	300 (5min)	300 (5min)	300 (5min)	3

Profiler

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Serv 64bit
`profiler.enabled`	false	false	true	true
`profiler.autoDump.interval`	0	0	0	0
`profiler.autoDump.reset`	true	true	true	true
`profiler.config`	null	null	null	null

Log

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Server 64bit	A
`log.console.level`	info	info	info	info	fi in w er
`log.file.level`	fine	fine	fine	fine	fi in w er

Client

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	Def.Server 64bit

`client.channel.minPool`	1	1	1	1
`client.channel.maxPool`	5	5	5	5

Server

Parameter	Def.32bit	Def.64bit	Def.Serv 32bit
`server.channel.cleanDelay`	5000	5000	5000
`server.log.dumpClientExceptionLevel`	FINE	FINE	FINE
`server.log.dumpClientExceptionFullStackTrace`	false	false	false
`server.cache.staticFile`	false	false	false

Distributed cluster

Parameter	Def.32bit	Def.64bit	Def.Server 32bit	De
`distributed.async.timeDelay`	0	0	0	0
`distributed.sync.maxRecordsBuffer`	100	100	100	100

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 = (OStoragePhysicalClusterConfiguration
```

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 at all. 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("cla
```

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:

```
db.declareIntent( new OIntentMassiveInsert() );

ODocument doc = new ODocument(db);
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=t
```

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": 10000
"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": 500
"db.demo.index.ClassIndexTestCompositeEmbeddedMapByKey.optimizationThreshold": 1
"db.demo.index.ClassIndexTestCompositeEmbeddedMapByValue.entryPointSize": 64,
"db.demo.index.ClassIndexTestCompositeEmbeddedMapByValue.items": 0,
"db.demo.index.ClassIndexTestCompositeEmbeddedMapByValue.maxUpdateBeforeSave": 5
"db.demo.index.ClassIndexTestCompositeEmbeddedMapByValue.optimizationThreshold":
"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": 100
"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": 10
"db.demo.index.ClassIndexTestPropertyByValueEmbeddedMap.entryPointSize": 64,
"db.demo.index.ClassIndexTestPropertyByValueEmbeddedMap.items": 0,
"db.demo.index.ClassIndexTestPropertyByValueEmbeddedMap.maxUpdateBeforeSave": 50
"db.demo.index.ClassIndexTestPropertyByValueEmbeddedMap.optimizationThreshold":
"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": 1000
"db.demo.index.ClassIndexTestPropertyLinkedMapByValue.entryPointSize": 64,
"db.demo.index.ClassIndexTestPropertyLinkedMapByValue.items": 0,
"db.demo.index.ClassIndexTestPropertyLinkedMapByValue.maxUpdateBeforeSave": 5000
"db.demo.index.ClassIndexTestPropertyLinkedMapByValue.optimizationThreshold": 10
"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"
"db.demo.index.SQLSelectCompositeIndexDirectSearchTestIndex.optimizationThreshol
"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": 500
"db.demo.index.TransactionUniqueIndexWithDotTest.label.optimizationThreshold": 1
"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"
"db.demo.index.classIndexManagerTestIndexValueAndCollection.optimizationThreshol
"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": 100
"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": 10000
"db.demo.index.sqlCreateIndexEmbeddedMapByValueIndex.entryPointSize": 64,
"db.demo.index.sqlCreateIndexEmbeddedMapByValueIndex.items": 0,
"db.demo.index.sqlCreateIndexEmbeddedMapByValueIndex.maxUpdateBeforeSave": 5000,
"db.demo.index.sqlCreateIndexEmbeddedMapByValueIndex.optimizationThreshold": 100
"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": 10000
"db.demo.index.sqlSelectIndexReuseTestEmbeddedListTwoProp8.entryPointSize": 64,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedListTwoProp8.items": 0,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedListTwoProp8.maxUpdateBeforeSave":
"db.demo.index.sqlSelectIndexReuseTestEmbeddedListTwoProp8.optimizationThreshold
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKey.entryPointSize": 64,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKey.items": 0,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKey.maxUpdateBeforeSave": 500
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKey.optimizationThreshold": 1
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKeyProp8.entryPointSize": 64,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKeyProp8.items": 0,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKeyProp8.maxUpdateBeforeSave"
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByKeyProp8.optimizationThreshol
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValue.entryPointSize": 64,
```

```
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValue.items": 0,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValue.maxUpdateBeforeSave": 5
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValue.optimizationThreshold":
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValueProp8.entryPointSize": 6
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValueProp8.items": 0,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValueProp8.maxUpdateBeforeSav
"db.demo.index.sqlSelectIndexReuseTestEmbeddedMapByValueProp8.optimizationThresh
"db.demo.index.sqlSelectIndexReuseTestEmbeddedSetProp8.entryPointSize": 64,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedSetProp8.items": 0,
"db.demo.index.sqlSelectIndexReuseTestEmbeddedSetProp8.maxUpdateBeforeSave": 500
"db.demo.index.sqlSelectIndexReuseTestEmbeddedSetProp8.optimizationThreshold": 1
"db.demo.index.sqlSelectIndexReuseTestProp9EmbeddedSetProp8.entryPointSize": 64,
"db.demo.index.sqlSelectIndexReuseTestProp9EmbeddedSetProp8.items": 0,
"db.demo.index.sqlSelectIndexReuseTestProp9EmbeddedSetProp8.maxUpdateBeforeSave"
"db.demo.index.sqlSelectIndexReuseTestProp9EmbeddedSetProp8.optimizationThreshol
"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
        }
    }
}
```

ETL

The OrientDB-ETL module is an amazing tool to move data from and to OrientDB by executing an ETL process. It's super easy to use. OrientDB ETL is based on the following principles:

- one configuration file in JSON format
- one Extractor is allowed to extract data from a source
- one Loader is allowed to load data to a destination
- multiple Transformers that transform data in pipeline. They receive something in input, do something, return something as output that will be processed as input by the next component

How ETL works

```
EXTRACTOR => TRANSFORMERS[] => LOADER
```

Example of a process that extract from a CSV file, apply some change, lookup if the record has already been created and then store the record as document against OrientDB database:

```
+-----------+----------------------+-----------+
|           |              PIPELINE             |
+ EXTRACTOR +----------------------+-----------+
|           |     TRANSFORMERS     |  LOADER   |
+-----------+----------------------+-----------+
|   FILE   ==>  CSV->FIELD->MERGE  ==> OrientDB |
+-----------+----------------------+-----------+
```

The pipeline, made of transformation and loading phases, can run in parallel by setting the configuration `{"parallel":true}`.

Installation

Starting from OrientDB v2.0 the ETL module will be distributed in bundle with the official release. If you want to use it, then follow these steps:

- 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

```
$ cd $ORIENTDB_HOME/bin
$ ./oetl.sh config-dbpedia.json
```

Available Components

- Blocks
- Sources
- Extractors
- Transformers
- Loaders

Examples:

- Import from CSV to a Graph
- Import from JSON
- Import DBPedia
- Import from a DBMS
- Import from Parse (Facebook)

ETL - Configuration

One of the most important OrientDB-ETL module features is the simplicity to configure complex ETL processed, just by working to a single JSON file.

The Configuration file is divided in the following sections:

- **config**, to manage all the settings and context's variables used by any component of the process
- **source**, to manage the source to process
- **begin**, as a list of Blocks to execute in order. This section is executed when the process begins
- **extractor**, contains the Extractor configuration
- **transformers**, contains the list of Transformers configuration to execute in pipeline
- **loader**, contains the Loader configuration
- **end**, as a list of Blocks to execute in order. This section is executed when the process is finished

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> } }
  ]
}
```

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" }]
      }
    }
}
```

Generic rules

- context variables can be used by prefixing them with $
- `$input` is the context variable assigned before each transformation
- to execute an expression using OrientDB SQL, use `={<expression>}`, example: `={eval('3 * 5')}`

Conditional execution

All executable blocks, like Transformers and Blocks, can be executed only if a condition is true by using the **if** conditional expression using the OrientDB SQL syntax. Example:

```
{ "let": {
    "name": "path",
    "value": "C:/Temp",
    "if": "${os.name} = 'Windows'"
  }
},
{ "let": {
    "name": "path",
    "value": "/tmp",
    "if": "${os.name}.indexOf('nux')"
  }
}
`
```

Log setting

Most of the blocks, like Transformers and Blocks, supports the `log` setting. Log can be one of the following values (case insensitive): `[NONE, ERROR, INFO, DEBUG]`. By default is `INFO`.

Set the log level to `DEBUG` to display more information on execution. Remember that logging slows down execution, so use it only for development and debug purpose. Example:

```
{ "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 (KHTM
    },
    "log": "DEBUG"
  }
}
```

Configuration variables

All the variables declared in "config" block are bound in the execution context and can be used by ETL processing.

There are also special variables used by ETL process:

Variable	Description	Type	Mandatory	Default value
log	Global "log" setting. Accepted values: `[NONE, ERROR, INFO, DEBUG]`. Useful to debug a ETL process or single component.	string	false	INFO
maxRetries	Maximum number of retries in case the loader raises a ONeedRetryException: concurrent modification of the same records	integer	false	10
parallel	Executes pipelines in parallel by using all the available cores	boolean	false	false

ETL - BLocks

Block components execute operations.

Available Blocks

let	code	console	

let

Assigns variable in the ETL process context.

- Component name: **let**

Syntax

Parameter	Description	Type	Mandatory	Default value
name	Variable name. Any $ prefix is ignored	string	true	-
value	Fixed value to assign	any	false	-
expression	Expression in OrientDB SQL language, to evaluate and assign	string	false	-

Example

Assign a value to the variable:

```
{ "let": { "name": "$filePath",  "value": "/temp/myfile"} }
```

Concats the $fileName variable to $fileDirectory to create the new variable $filePath:

```
{ "let": { "name": "$filePath",  "expression": "$fileDirectory.append( $fileName )"} }
```

code

Execute a snippet of code in any of the JVM supported languages. Default is Javascript.

- Component name: **code**

Syntax

Parameter	Description	Type	Mandatory	Default value
language	Programming language used	string	false	Javascript
code	Code to execute	string	true	-

Example

```
{ "code": { "language": "Javascript",
            "code": "print('Hello World!');"}
}
```

Console

Execute commands invoking the OrientDB Console.

- Component name: **console**

Syntax

Parameter	Description	Type	Mandatory	Default value
file	File path containing the commands to execute	string	false	-
commands	Array of commands, as string, to execute in sequence	string array	false	-

Example

Invoice the console with a file containing the commands to execute

```
{ "console": { "file": "/temp/commands.sql"}  }
```

```
{ "console": {
    "commands": [
      "CONNECT plocal:/temp/db/mydb admin admin",
```

```
        "INSERT INTO Account set name = 'Luca'"
    ] }
}
```

ETL - Sources

Source components represent the source where to extract the content. Source is optional, some Extractor like JDBCExtractor works without a source.

Available Sources

file	input	http	

File

Represents a source file where to start reading. Files can be text files or compressed with tar.gz.

- Component name: **file**

Syntax

Parameter	Description	Type	Mandatory	Default value
path	File path	string	true	-
lock	Lock the file while the extraction phase	boolean	false	false

Example

Extracts from the file "/temp/actor.tar.gz":

```
{ "file": { "path": "/temp/actor.tar.gz", "lock" : true } }
```

Input

Extracts data from console input. This is useful when the ETL works in PIPE with other tools

- Component name: **input**

Syntax

Parameter	Description	Type	Mandatory	Default value

Example

Extracts the file as input

```
cat /etc/csv|oetl.sh "{transformers:[{csv:{}}]}"
```

HTTP

Use a HTTP endpoint as content source.

- Component name: **http**

Syntax

Parameter	Description	Type	Mandatory	Default value
url	HTTP URL to invoke	String	true	-
method	HTTP Method between "GET", "POST", "PUT", "DELETE", "HEAD", "OPTIONS", "TRACE"	String	false	GET
headers	Request headers as inner document key/value	Document	false	

Example

Execute a HTTP request against the URL "http://ip.jsontest.com/" in GET setting the User-Agent in headers:

```
{ "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 (KHTM
    }
  }
}
```

ETL - Extractors

Extractor components are the first part of the ETL process responsible of extracting data.

Available Extractors

row	jdbc	json	

row

Extracts content row by row.

- Component name: **row**
- Output class: [**String**]

Syntax

Parameter	Description	Type	Mandatory	Default value

Example

```
{ "row": {} }
```

JDBC

Extracts data from any **DBMS** that support JDBC driver. In order to get the ETL component to connect to the source database, put the DBMS's JDBC driver in the **classpath** or **$ORIENTDB_HOME/lib** directory.

- Component name: **jdbc**
- Output class: [**ODocument**]

Syntax

Parameter	Description	Type	Mandatory	Default value
driver	JDBC Driver class	string	true	-
url	JDBC URL to connect	string	true	-
userName	DBMS User name	string	true	-

userPassword	DBMS User password	string	true	-
query	Query that extract the record to import	string	true	-
queryCount	Query that return the count of the fetched records. This is used to provide a correct progress indicator	string	false	-

Example

Extracts all the Client from the MySQL database "test" hosted on localhost:

```
{ "jdbc": {
    "driver": "com.mysql.jdbc.Driver",
    "url": "jdbc:mysql://localhost/test",
    "userName": "root",
    "userPassword": "",
    "query": "select * from Client"
  }
}
```

---

json

Extracts content by parsing json objects. If the content has more json items must be enclosed between [].

- Component name: **json**
- Output class: [**ODocument**]

Syntax

Parameter	Description	Type	Mandatory	Default value

Example

```
{ "json": {} }
```

ETL Transformers

Transformer components are executed in pipeline. They work against the received input returning an output.

Before the execution, the `$input` variable is always assigned, so you can get at run-time and use if needed.

Available Transformers

CSV	FIELD	MERGE	VERTEX
CODE	LINK	EDGE	SKIP
LOG	BLOCK	COMMAND	

CSV

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	Column separator	char	false	,
columnsOnFirstLine	Columns are described in the first line	boolean	false	true
columns	Columns array containing names, and optionally types by postfixing names with :. Specifying type guarantee better performance	string[]	false	-
nullValue	value to consider as NULL. Default is not declared	string	false	-
stringCharacter	String character delimiter	char	false	"
skipFrom	Line number where start to	integer	true	-

	skip			
skipTo	Line number where skip ends	integer	true	-

Example

Transforms a row in CSV (as ODocument), using comma as separator, considering "NULL" as null value and skipping the rows 2-4:

```
{ "csv": { "separator": ",", "nullValue": "NULL",
           "skipFrom": 1, "skipTo": 3 } }
```

FIELD

Execute a SQL transformation against a field.

Component description.

- Component name: **vertex**
- Supported inputs types: [**ODocument**]
- Output: **ODocument**

Syntax

Parameter	Description	Type	Mandatory	Default value
fieldName	Document's field name to assign	string	true	-
expression	Expression to evaluate. You can use OrientDB SQL syntax	string	true	-
value	Value to set. If the value is taken or computed at run-time, use `expression` instead	any	false	-
operation	Operation to execute against the field: set, remove. Default is set	string	false	set
save	Save the vertex/edge/document right after the setting of the field	boolean	false	false

Examples

Transforms the field 'class' into the ODocument's class by prefixing it with '_':

```
{ "field": { "fieldName": "@class", "expression": "class.prefix('_')"} }
```

Applies the class name based on the value of another field:

```
{ "field": { "fieldName": "@class", "expression": "if( ( fileCount >= 0 ), 'D', 'F')"} }
```

Assigns to the "name" field the last part of a path:

```
{ "field": { "fieldName": "name",
      "expression": "path.substring( eval( '$current.path.lastIndexOf(\"/\") + 1') )" }
```

Assign a fixed value:

```
{ "field": { "fieldName": "counter", "value": 0} }
```

Rename a field from 'salary' to 'remuneration':

```
{ "field": { "fieldName": "remuneration", "expression": "salary"} },
{ "field": { "fieldName": "salary", "operation": "remove"} }
```

MERGE

Merges input ODocument with another one, loaded by a lookup. Lookup can 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	Field name where the join value is saved	string	true	-
lookup	Can be the index name where to execute the lookup, or a SELECT query	string	true	-

unresolvedLinkAction	Action to execute in case the JOIN hasn't been resolved. Actions can be: 'NOTHING' (do nothing), WARNING (increment warnings), ERROR (increment errors), HALT (interrupt the process), SKIP (skip current row).	string	false	NOTHING

Example

Merges current record against the record returned by the lookup on index "V.URI" with the value contained in the field "URI" of the input's document:

```
{ "merge": { "joinFieldName":"URI", "lookup":"V.URI" } }
```

VERTEX

Transforms a ODocument in a OrientVertex.

Component description.

- Component name: **vertex**
- Supported inputs types: [**ODocument**, **OrientVertex**]
- Output: **OrientVertex**

Syntax

Parameter	Description	Type	Mandatory	Default value
class	Vertex class name to assign	string	false	V
	Vertices with			

skipDuplicates	duplicate keys are skipped. If `skipDuplicates:true` and a UNIQUE constraint is defined on vertices ETL will ignore it with no exceptions. Available v. 2.1	boolean	false	false

Example

Transform the ODocument in a Vertex setting as class the value of "$className" variable:

```
{ "vertex": { "class": "$className", "skipDuplicates": true } }
```

EDGE

Transform a JOIN value in one or more EDGEs between current vertex and all the vertices returned by the lookup. Lookup can be a lookup against an index or a SELECT query.

Component description.

- Component name: **EDGE**
- Supported inputs types: [**ODocument**, **OrientVertex**]
- Output: **OrientVertex**

Syntax

Parameter	Description	Type	Mandatory	Default value
joinFieldName	Field name where the join value is saved	string	true	-
direction	Edge direction	string	false	'out'
class	Edge's class name	string	false	'E'
	Can be the index name			

lookup	where to execute the lookup, or a SELECT query	string	true	-
unresolvedLinkAction	Action to execute in case the JOIN hasn't been resolved. Actions can be: 'NOTHING' (do nothing), CREATE (create a OrientVertex setting as primary key the join value), WARNING (increment warnings), ERROR (increment errors), HALT (interrupt the process), SKIP (skip current row).	string	false	NOTHING

Example

Creates an EDGE from the current vertex, with class "Parent", to all the vertices returned by the lookup on "D.inode" index with the value contained in the field "inode_parent" of the input's vertex:

```
{ "edge": { "class": "Parent", "joinFieldName": "inode_parent",
            "lookup":"D.inode", "unresolvedLinkAction":"CREATE"} }
```

SKIP

Skip an execution in pipeline if the condition in "expression" field is true.

Component description.

- Component name: **skip**

- Supported inputs types: [**ODocument**, **OrientVertex**]
- Output: same type as input

Syntax

Parameter	Description	Type	Mandatory	Default value
expression	SQL expression to evaluate. If true the current execution is skipped	string	true	-

Example

Skip the current record if name is null:

```
{ "skip": { "expression": "name is null"} }
```

CODE

Executes a snippet of code in any of the JVM supported languages. Default is Javascript. Last object in the code is returned as output. In the execution context are bound the following variables:

- `input` with the input object received
- `record` with the record extracted from input object when is possible. In case the input object is a Vertex/Edge, the underlying ODocument is assigned to the variable

Component description.

- Component name: **code**
- Supported inputs types: [**Object**]
- Output: **Object**

Syntax

Parameter	Description	Mandatory	Default value	
language	Programming language used	string	false	Javascript

code	Code to execute	string	true	-

Example

Displays current record and returns the parent.

```
{ "code": { "language": "Javascript",
            "code": "print('Current record: ' + record); record.field('parent');"}
}
```

LINK

Transform a JOIN value in LINK in current record with the result of the lookup. Lookup can be a lookup against an index or a SELECT query.

Component description.

- Component name: **link**
- Supported inputs types: [**ODocument**, **OrientVertex**]
- Output: **ODocument**

Syntax

Parameter	Description	Type	Mandatory	Default value
joinFieldName	Field name where the join value is saved	string	false	-
joinValue	Value to lookup	string	false	-
linkFieldName	Field name containing the link to set	string	true	-
linkFieldType	Type of link between: LINK, LINKSET and LINKLIST	string	true	-
	Can be the index name where to			

lookup	execute the lookup, or a SELECT query	string	true	-
unresolvedLinkAction	Action to execute in case the JOIN hasn't been resolved. Actions can be: 'NOTHING' (do nothing), CREATE (create a ODocument setting as primary key the join value), WARNING (increment warnings), ERROR (increment errors), HALT (interrupt the process), SKIP (skip current row).	string	false	NOTHING

Example

Transform a JOIN value in LINK in current record (set as "parent" of type LINK) with the result of the lookup on index "D.inode" with the value contained in the field "inode_parent" of the input's document:

```
{ "link": { "linkFieldName": "parent", "linkFieldType": "LINK",
            "joinFieldName": "inode_parent", "lookup":"D.inode", "unresolvedLinkAction":"CRE
```

LOG

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	Prefix to write before the content	string	false	-
postfix	Postfix to write after the content	string	false	-

Example

Simply log current value:

```
{ "log": {} }
```

Log current value with "-> " as prefix:

```
{ "log": { "prefix" : "-> "} }
```

Block

Executes a Block as transformation step.

Component description.

- Component name: **block**
- Supported inputs types: [**Any**]
- Output: **Any**

Syntax

Parameter	Description	Type	Mandatory	Default value
block	Block to execute	document	true	-

Example

Simply log current value:

```
{ "block": {
    "let": {
      "name": "id",
      "value": "={eval('$input.amount * 2')}"
    }
  }
}
```

Command

Executes a command.

Component description.

- Component name: **command**
- Supported inputs types: [**ODocument**]
- Output: **ODocument**

Syntax

Parameter	Description	Type	Mandatory	Default value
language	Command language. Available are: sql (default) and gremlin	string	false	sql
command	Command to execute	string	true	-

Example

```
{
  "command" : {
    "command" : "select from E where id = ${edgeid}",
    "output" : "edge"
  }
}
```

ETL - Loaders

Loader components are the last part of the ETL process responsible of the saving of records.

Available Loaders

Output	OrientDB		

Output

It's the default Loader. It prints the transformation result to the console output.

- Component name: **output**
- Accepted input classes: [**Object**]

OrientDB

Loads record and vertices into a OrientDB database.

- Component name: **orientdb**
- Accepted input classes: [**ODocument**, **OrientVertex**]

Syntax

Parameter	Description	Type	Mandatory	Default value
dbURL	Database URL	string	true	-
dbUser	User Name	string	false	admin
dbPassword	User Password	string	false	admin
dbAutoCreate	If the database not exists, create it automatically	boolean	false	true
tx	Use transactions or not	boolean	false	false
wal	Use WAL (Write Ahead Logging). Disable WAL to achieve better performances	boolean	false	true
	With transactions enabled, commit every X entries.			

batchCommit	Use this to avoid having one huge transaction in memory	integer	false	0
dbType	Database type, between 'graph' or 'document'	string	false	document
indexes	Contains the indexes used on ETL process. Before starting any declared index not present in database will be created automatically. Index configuration must have "type", "class" and "fields"	inner document	false	-

Example

Below an example of configuration to load in a OrientDB database called "dbpedia" in directory "/temp/databases" open using "plocal" protocol and used as "graph". The loading is transactional and commits the transaction every 1,000 inserts. To lookup vertices has been create the index against the property string "URI" in base vertex "V" class. The index is unique.

```
"orientdb": {
      "dbURL": "plocal:/temp/databases/dbpedia",
      "dbUser": "importer",
      "dbPassword": "IMP",
      "dbAutoCreate": true,
      "tx": false,
      "batchCommit": 1000,
      "wal" : false,
      "dbType": "graph",
      "indexes": [{"class":"V", "fields":["URI:string"], "type":"UNIQUE" }]
    }
```

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

Where the relationship is between Post and Comment as One-2-Many. One Post can have multiple Comments. We're representing them as they would appear in a RDBMS, but the source could be anything.

With a 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 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": { "row": {} },
  "transformers": [
    { "csv": {} },
    { "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 plocal because it's faster, but if you have an OrientDB server up & running, use "remote:" instead. Note the classes and indexes declared in 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": { "row": {} },
  "transformers": [
```

```
      { "csv": {} },
      { "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 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` represents a relationship with nodes of the same type. They are under form of array if 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, an unique has 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"}, {"n
      "indexes": [{"class":"Account", "fields":["id:integer"], "type":"UNIQUE_HASH_INDEX" }]
    }
  }
}
```

Note also the setting

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

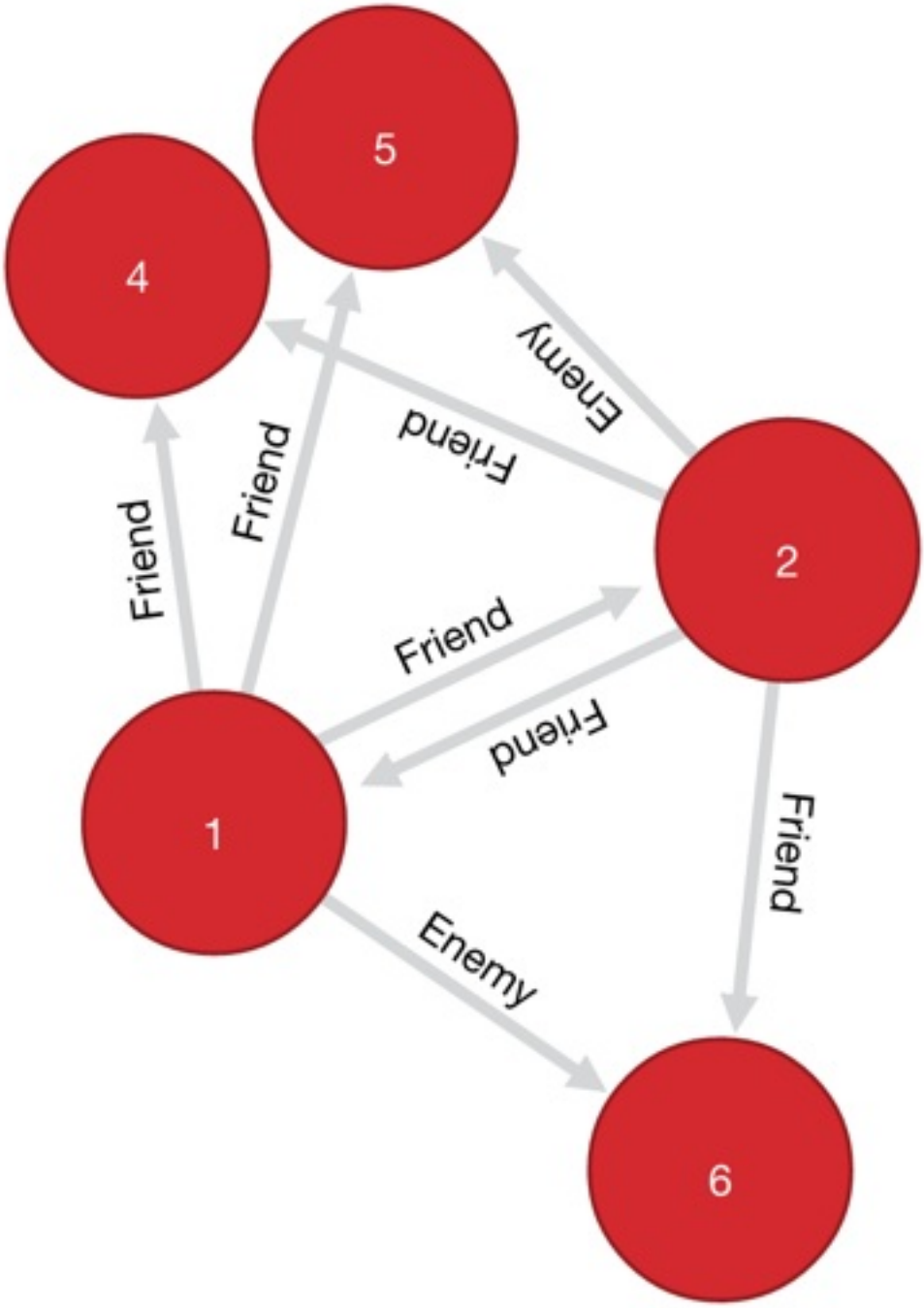
In OrientDB Loader to allow importing the property "id". Without this option the Blueprints standard would reject it because "id" is a reserved name.

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=42.205M
[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'
[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],ene
[1:merge] DEBUG Transformer output: Account#11:1{id:2,in_Friend:[#12:0],name:Suzie,friends:[
[1:vertex] DEBUG Transformer input: Account#11:1{id:2,in_Friend:[#12:0],name:Suzie,friends:[
[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],enemi
[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:Suz
[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) Tota
```

Once ready, let's open the database with Studio and this is the result:


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

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 we'll skip (look at `"skipFrom": 1, "skipTo": 3` in CSV transformer)
- The vertex class name is created automatically based on file name, so we can use the same file against any DBPedia file
- The Primary Key is the "URI" field where it has been created a UNIQUE index (look at "ORIENTDB" loader)
- The "merge" transformer is used to allow to reimport 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" : {
    "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" }]
    }
  }
}
```

Import from Parse

Parse is a very popular BaaS (Backend as a Service), acquired by Facebook. Parse uses MongoDB as database and allows to export the database in JSON format. The format is an array of JSON object. 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 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 if an embedded object containing:
  - `className` as target class name
  - `objectIf` 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 `user` class. Note the creation of class `User` in OrientDB that extends `V` (Base Vertex class). We created an index against property `User.objectId` to use the same ID as for Parse. If you execute this ETL importing multiple time, the record in OrientDB will be updated thanks to the `merge`.

```
{
  "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" }
      ]
    }
  }
}
```

Look also at - Import from JSON.

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

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.

Limitations

OrientDB v 2.0.x 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
- 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.

Distributed Architecture Lifecycle

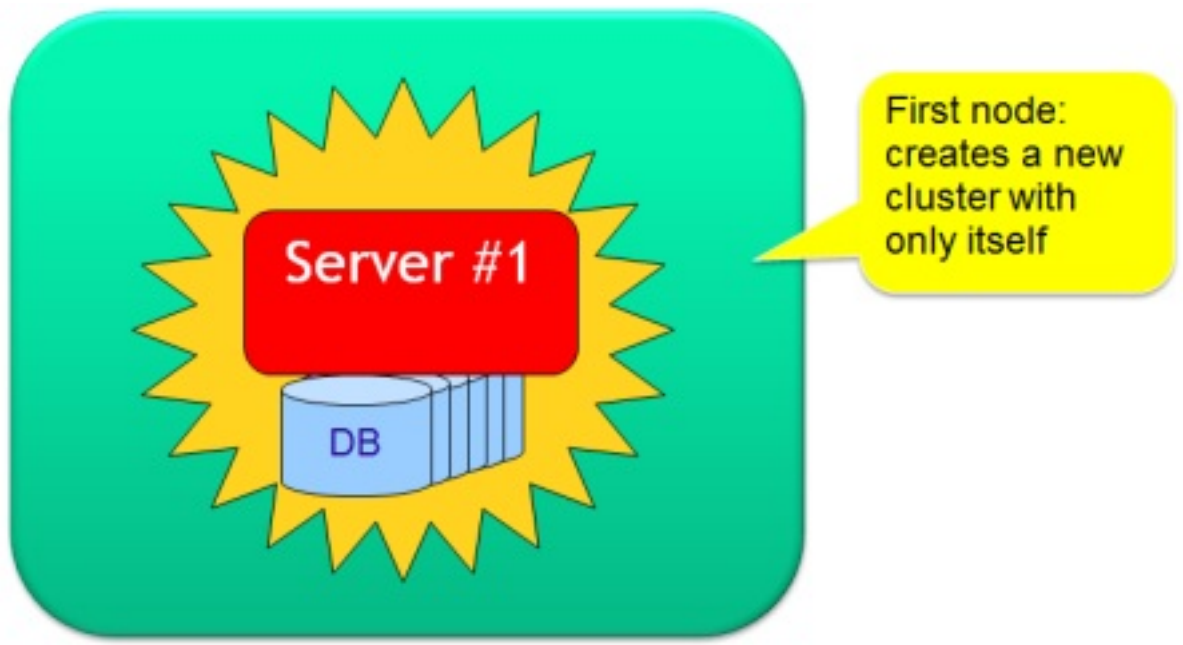
In OrientDB Distributed Architecture all the nodes are masters (Multi-Master), while in most DBMS the replication works in Master-Slave mode where there is only one Master node and N Slaves that are use only for reads or when the Master is down.

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:

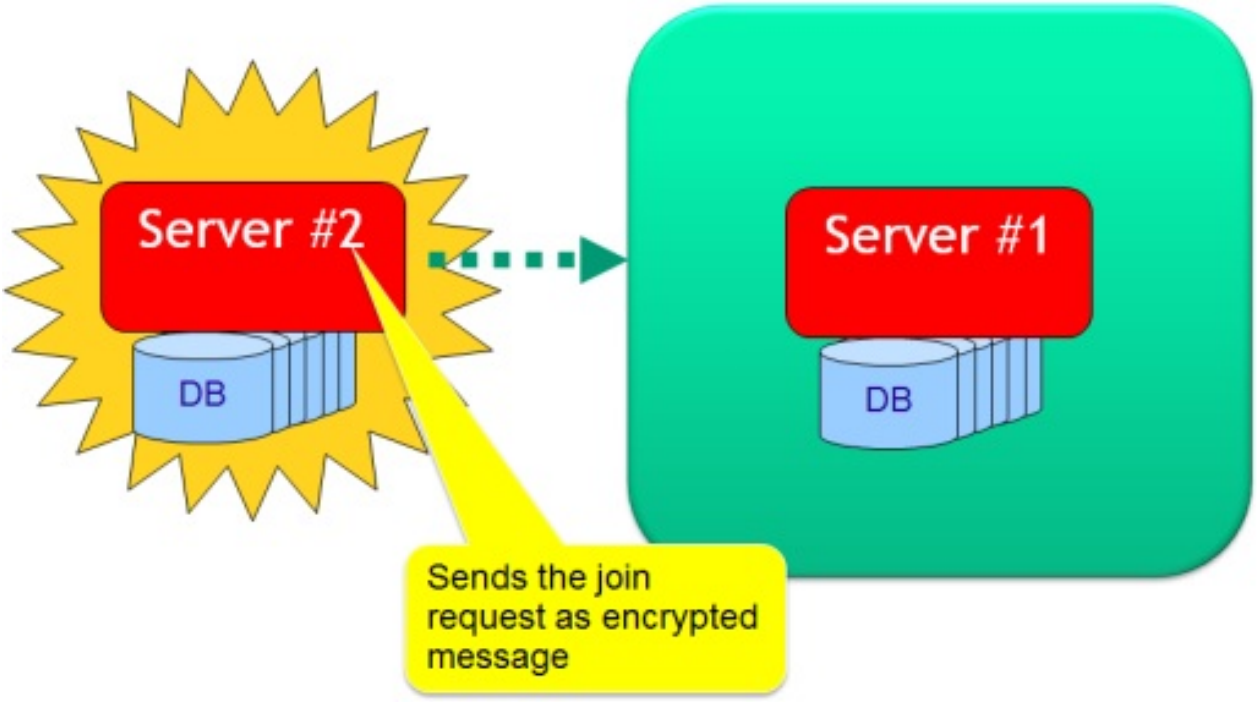
```
<hazelcast>
    ...
    <network>
        ...
        <!--
            Make sure to set enabled=true
            Make sure this configuration is exactly the same on
            all members
        -->
        <symmetric-encryption enabled="true">
            <!--
               encryption algorithm 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.



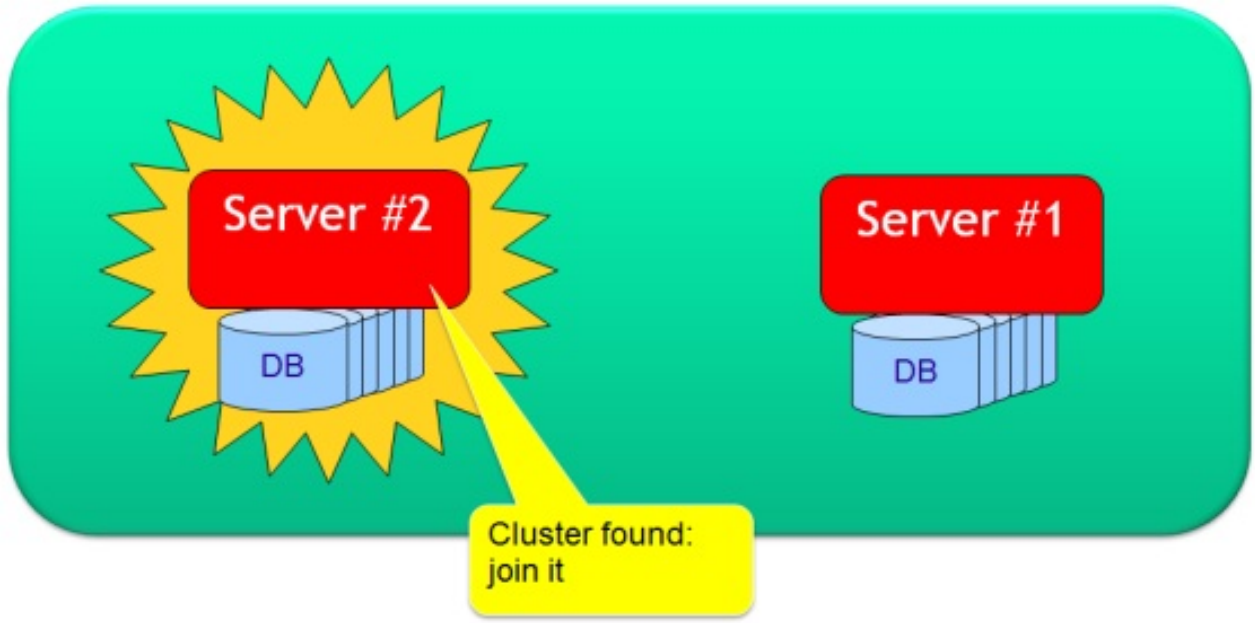
For more information look at: Hazelcast Encryption.

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.

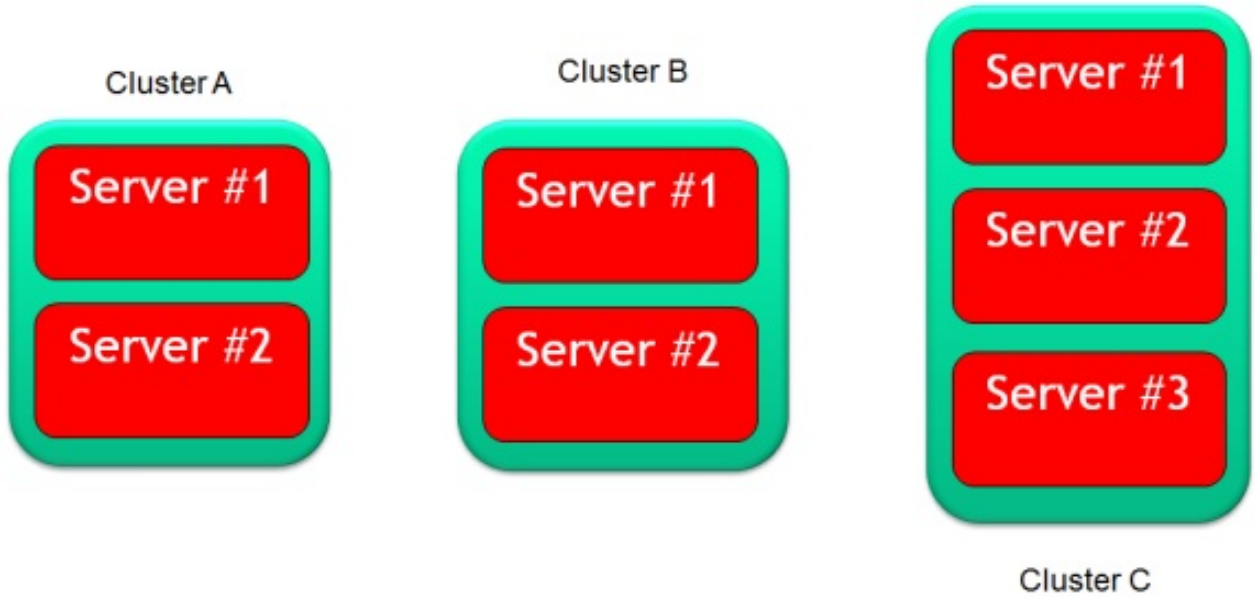
```
<hazelcast>
  <group>
    <name>orientdb</name>
    <password>orientdb</password>
  </group>
</hazelcast>
```

In this case Server #2 joins the existent cluster.



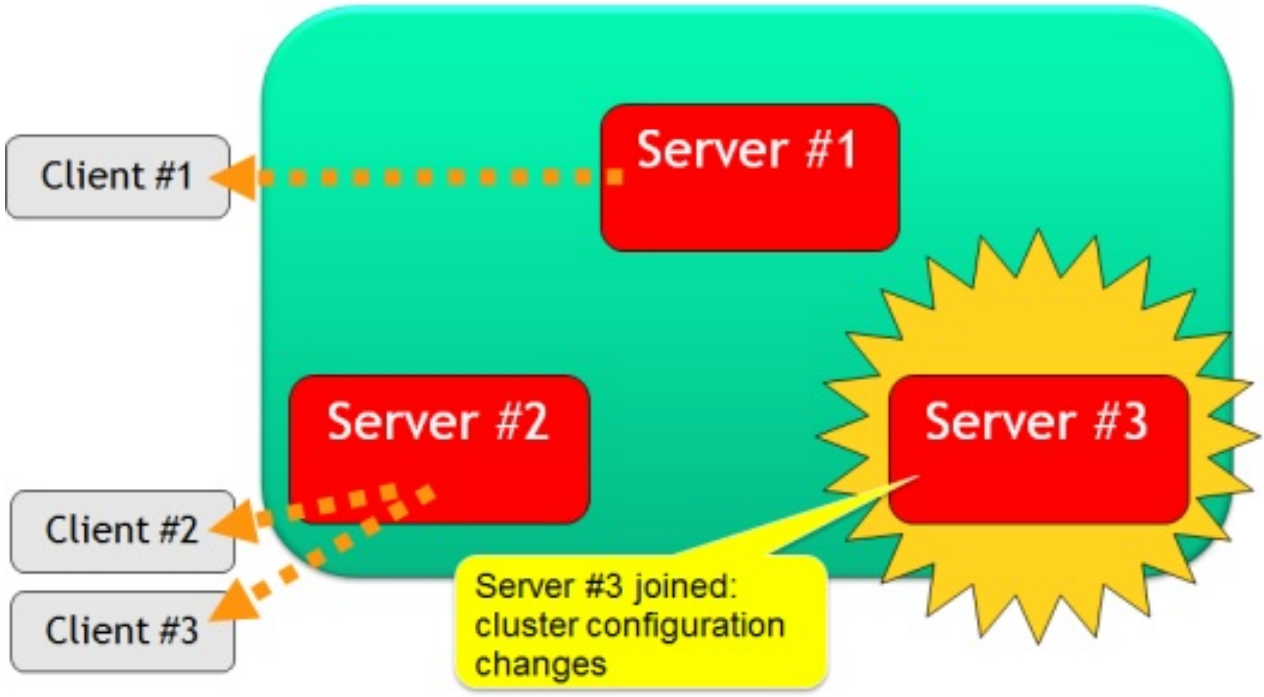
Multiple clusters

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

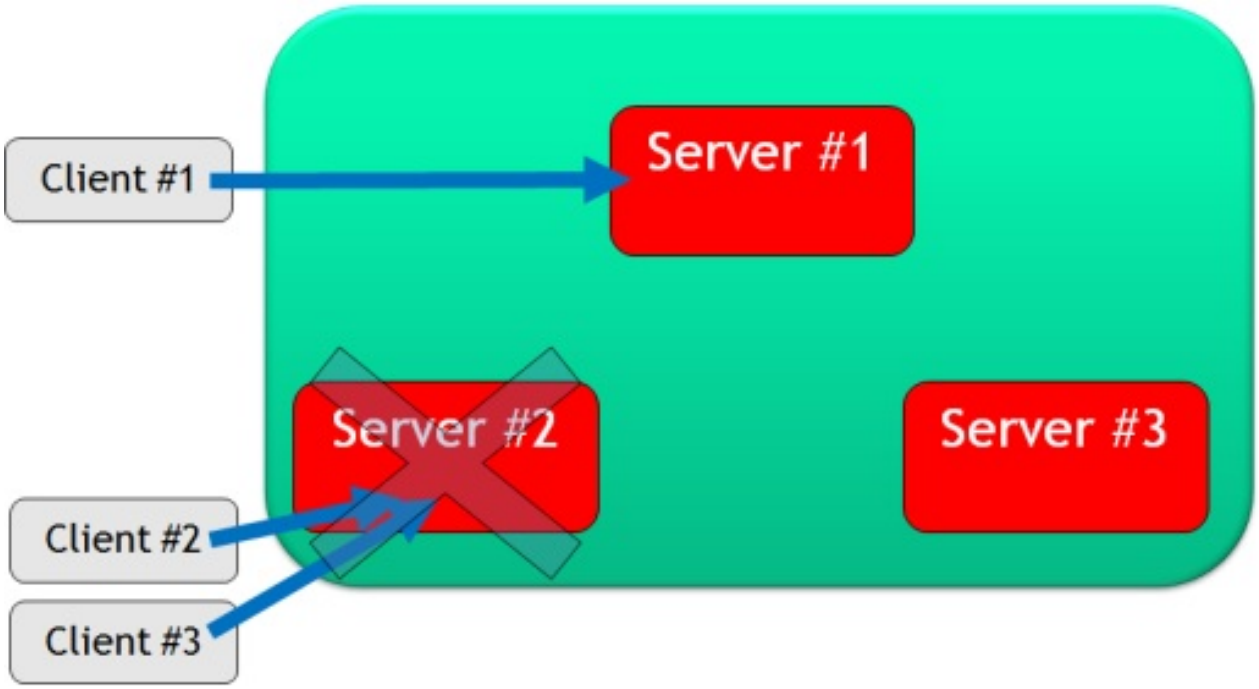
Every time a new Server Node joins or leaves the Cluster, the new Cluster configuration is broadcasted to all the connected clients. Everybody knows the cluster layout and who has a database!


Client #1
Server #1
Server #2
Server #3
Client #2
Client #3
Server #3 joined:
cluster configuration
changes

Fail over management

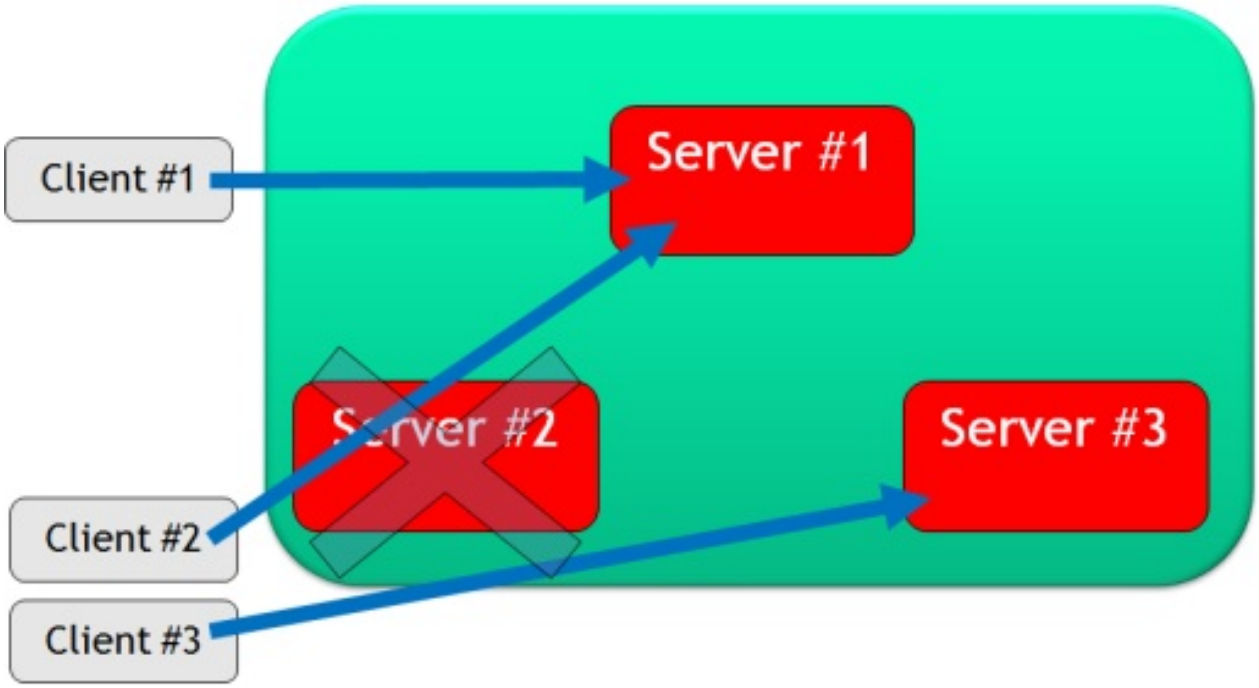
When a node is unreachable

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.



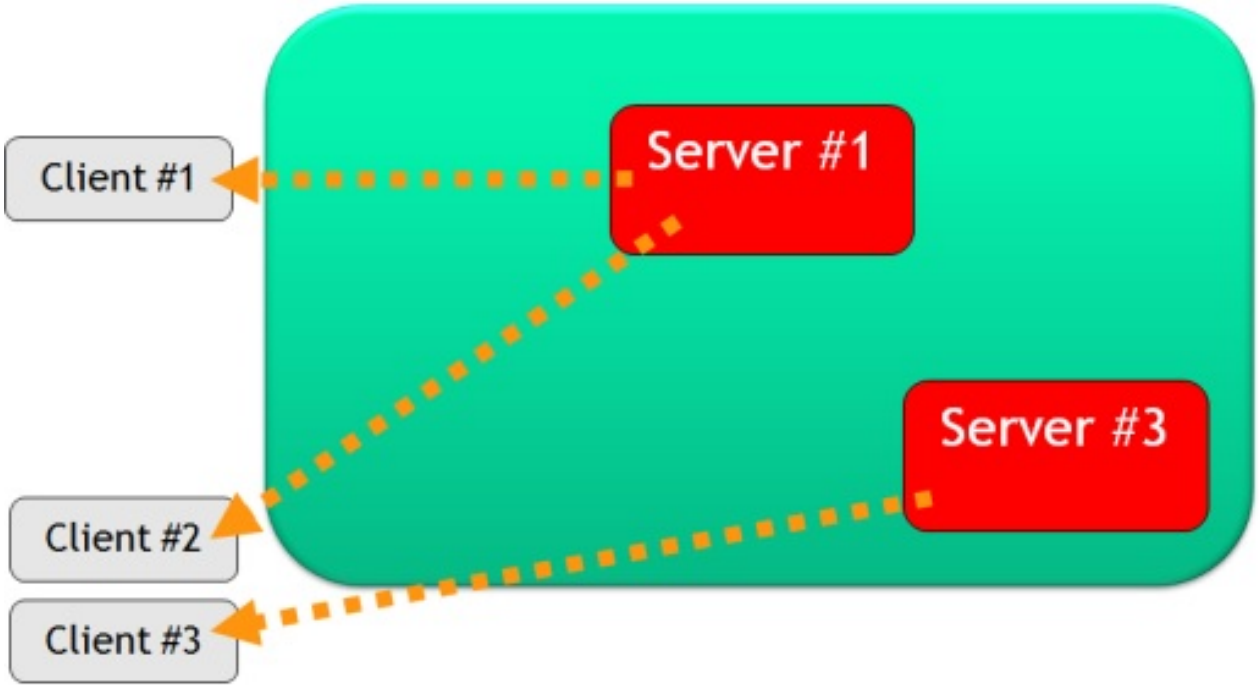
Automatic switch of servers

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

Look also at Replication and pages.

The distributed configuration is made of 3 files under the **config/** directory:

- orientdb-server-config.xml
- default-distributed-db-config.json
- Asynchronous replication mode
- hazelcast.xml
    - Cloud support

orientdb-server-config.xml

To enable and configure the clustering between nodes, add and enable the **OHazelcastPlugin**. This task 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, then every time the cluster shape changes the database specific file is changed.

Default **default-distributed-db-config.json** file content:

```
{
    "autoDeploy": true,
    "hotAlignment": false,
    "executionMode": "undefined",
    "readQuorum": 1,
    "writeQuorum": 2,
    "failureAvailableNodesLessQuorum": false,
    "readYourWrites": true,
    "clusters": {
        "internal": {
        },
        "index": {
        },
        "*": {
            "servers" : [ "<NEW_NODE>" ]
        }
    }
}
```

Where:

Parameter	Description	Default value
autoDeploy	Auto deploy the database in case the joining node hasn't it. It can be `true` or `false`	`true`
hotAlignment	In case a node left the cluster `hotAlignment` the synchronization queue is left or not for hot alignment when the node will join the cluster again. It can be `true` or `false`	`true`
executionMode	. It can be `undefined` to let to the client to decide per call execution if synchronous (default) or asynchronous. `synchronous` forces synchronous mode, and	`undefined`

	`asynchronous` forces asynchronous mode	
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`
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.x
           xmlns="http://www.hazelcast.com/schema/config" xmlns:xsi="http://www.w3.org/2001/
  <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 u
        <host-header>ec2.amazonaws.com</host-header>              <!-- optional, default is e
                                                                      shouldn't be set as it
        <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,
    "clusters": {
        "internal": {
        },
        "index": {
        },
        "*": {
            "servers" : [ "<NEW_NODE>" ]
        }
    }
}
```

Misc

Load balancing

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 3434 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);
```

History

1.7

Simplified configuration by moving. Removed some flags (replication:boolean, now it's deducted by the presence of "servers" field) and settings now are global (autoDeploy, hotAlignment, offlineMsgQueueSize, readQuorum, writeQuorum, failureAvailableNodesLessQuorum, readYourWrites), but you can overwrite them per-cluster.

For more information look at News in 1.7.

Distributed Architecture Plugin

Java class: `com.orientechnologies.orient.server.hazelcast.OHazelcastPlugin`

Introduction

This is part of Distributed Architecture. Configure a distributed clustered architecture. This task is configured as a Server handler. The task can be configured easily by changing these parameters:

- **enabled**: Enable the plugin: `true` to enable, `false` to disable it.
- **configuration.hazelcast**: The location of the Hazelcast configuration file (`hazelcast.xml`).
- **alias**: An alias for the current node within the cluster name. Default value is the IP address and port for OrientDB on this node.
- **configuration.db.default**: The location of a file that describes, using JSON syntax, the synchronization configuration of the various clusters in the database.

Default configuration in orientdb-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="${ORIENTDB_HOME}/config/default-d
      <parameter name="configuration.hazelcast" value="${ORIENTDB_HOME}/config/hazelcast.
   </parameters>
</handler>
```

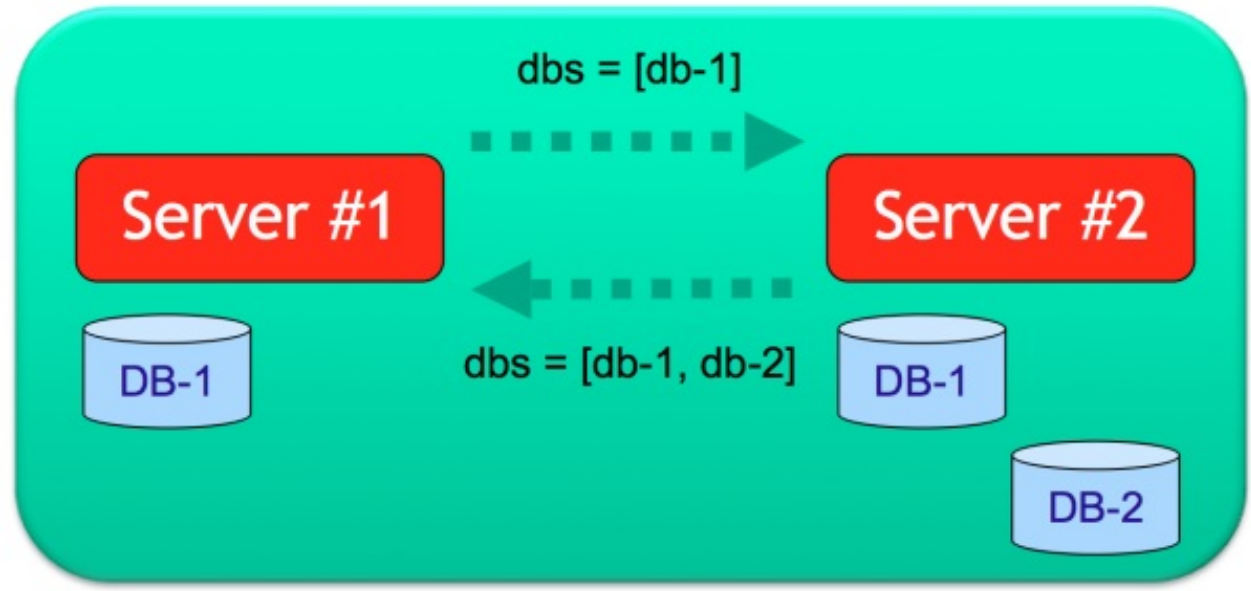
Replication

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.

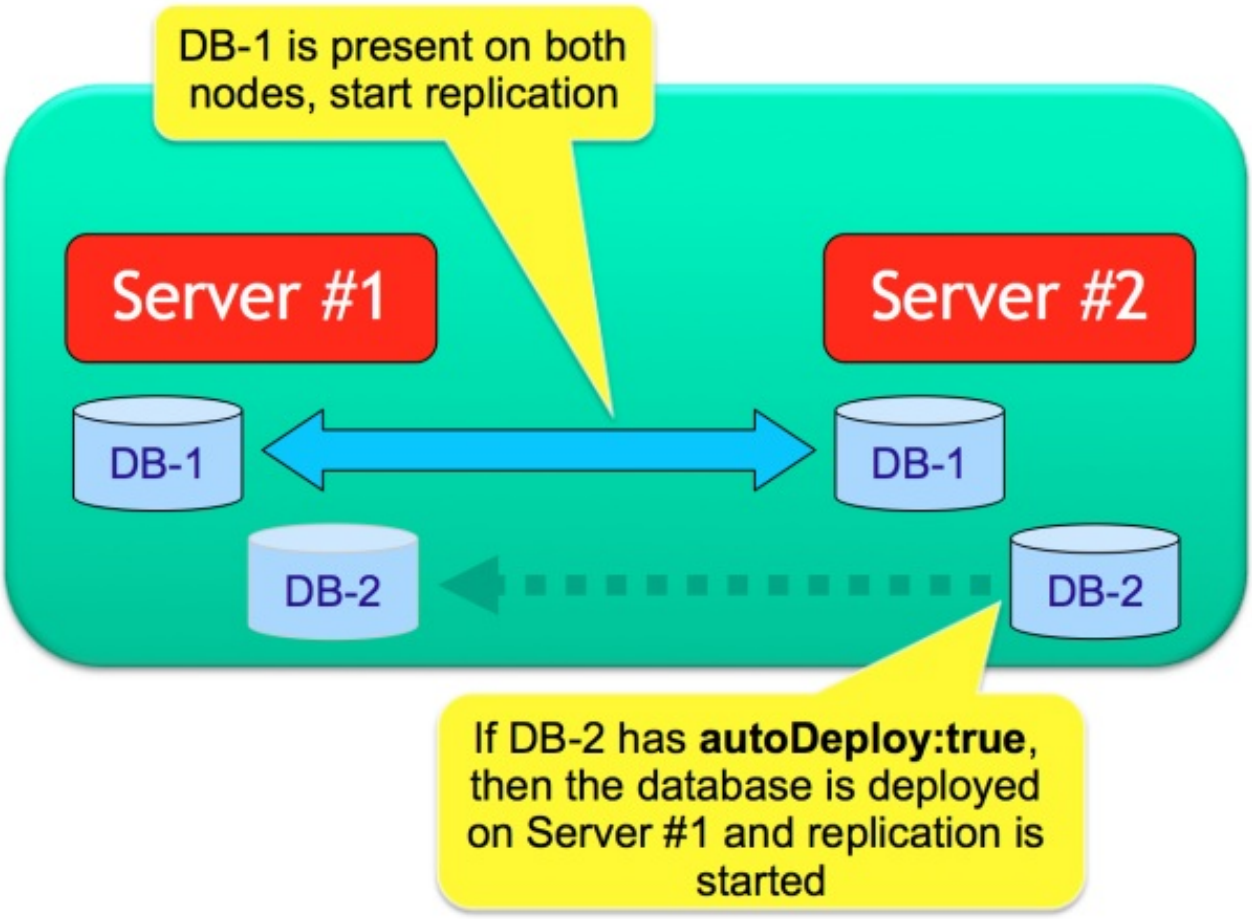
Sharing of database

In Distributed Architecture the replicated database must have he same name. When a OrientDB Server is starting, 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.



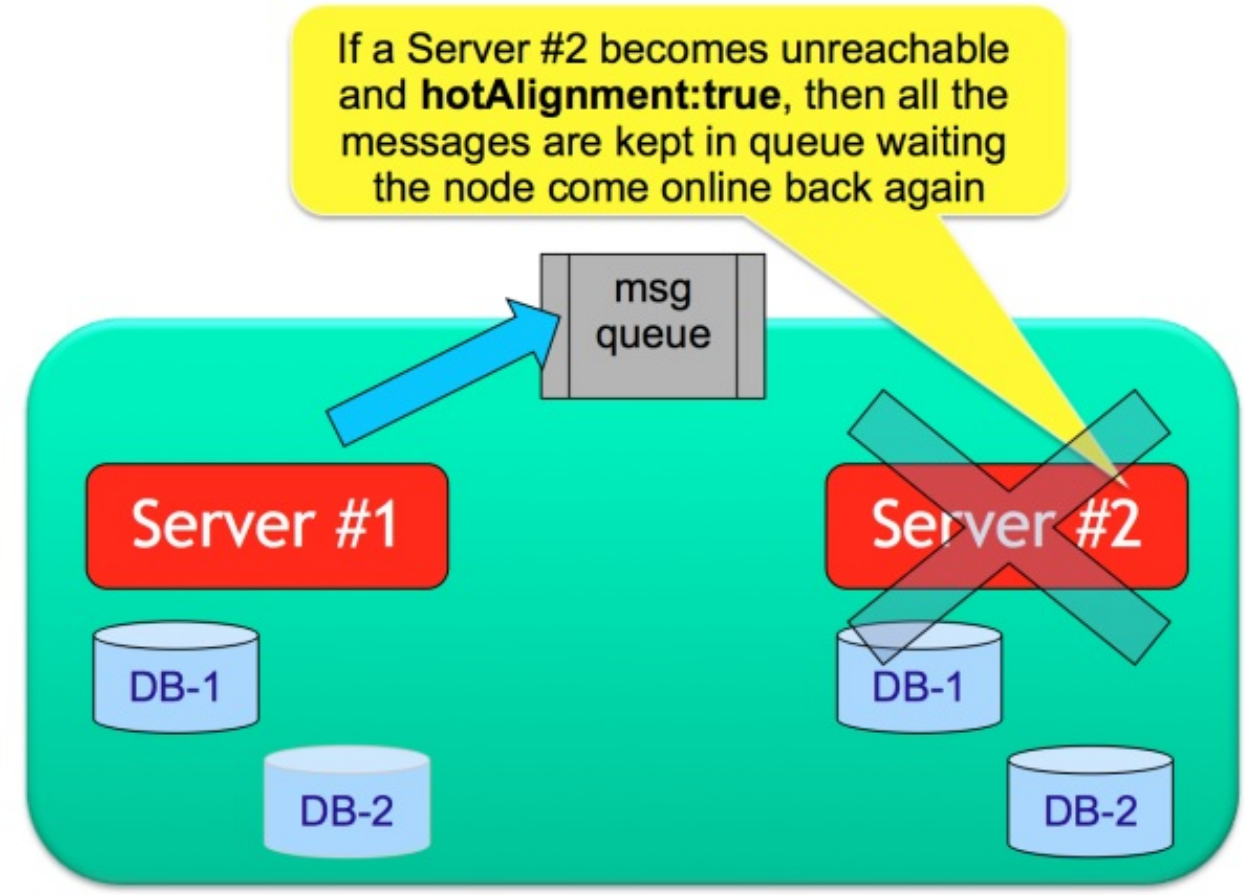
NOTE: In Distributed Architecture assure to avoid conflict with database names, otherwise 2 different databases could start replication with the chance to get corrupted.

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.

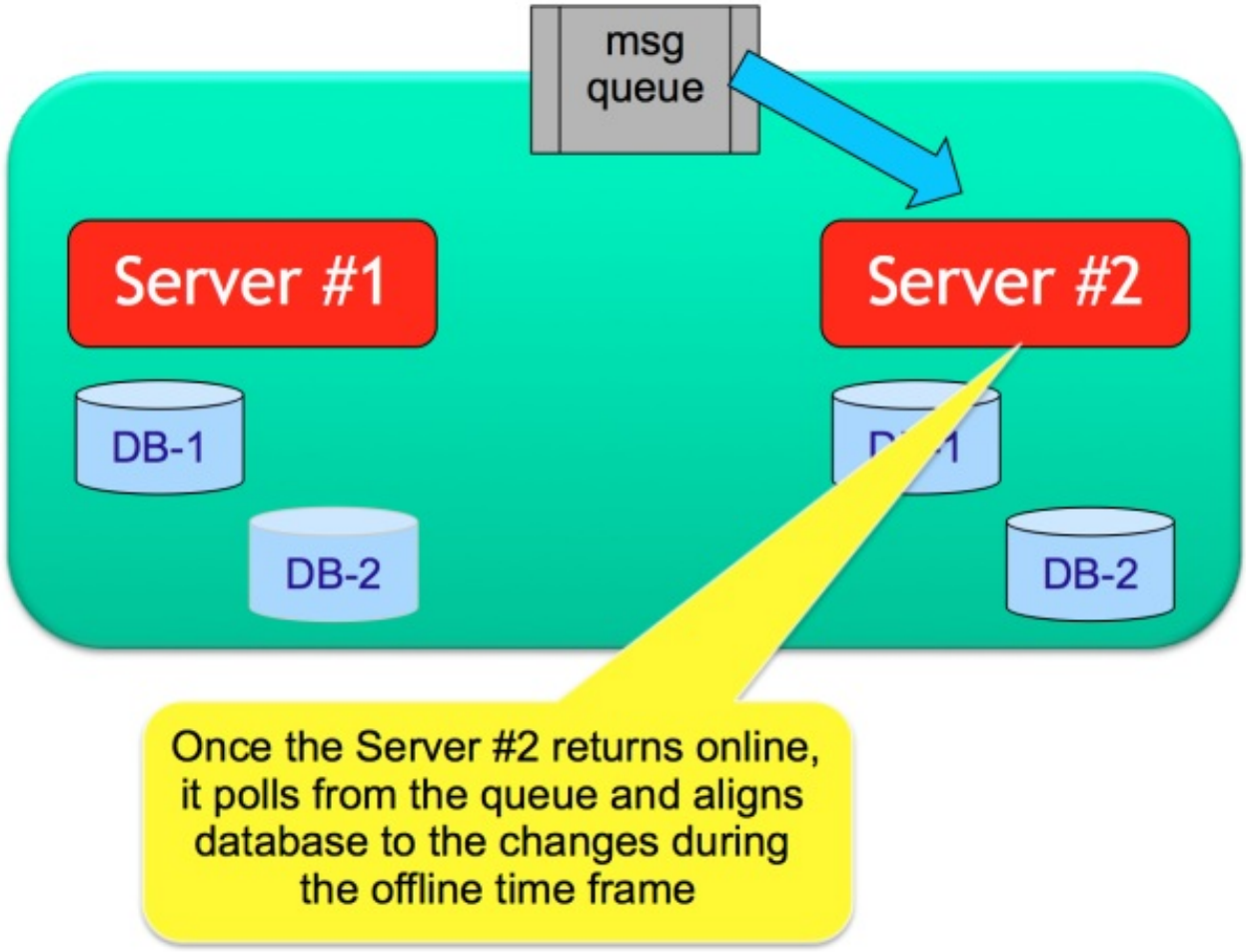

DB-1 is present on both nodes, start replication
Server #1
Server #2
DB-1
DB-1
DB-2
DB-2
If DB-2 has **autoDeploy:true**, then the database is deployed on Server #1 and replication is started

Server unreachable

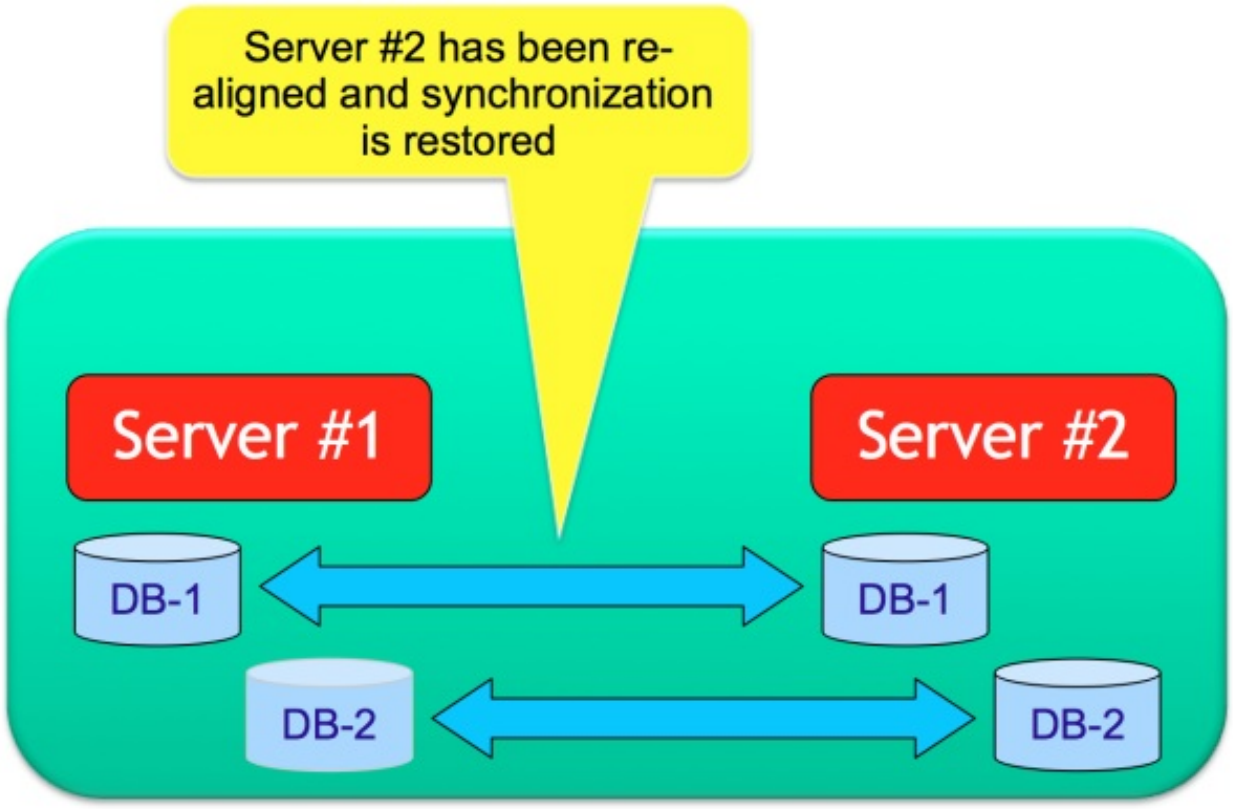
In case a server becomes unreachable, the node is removed by database configuration unless the setting `"hotAlignment" : true`. In this case all the new synchronization messages are kept in a distributed queue.



As soon as the Server becomes online again, it starts the synchronization phase (status=SYNCHRONIZING) by polling all the synchronization messages in the queue.



Once the alignment is finished, the node becomes online (status=ONLINE) and the replication continues like at the beginning.



Further readings

Continue with:

- Distributed Architecture
- Distributed Sharding
- Distributed database configuration

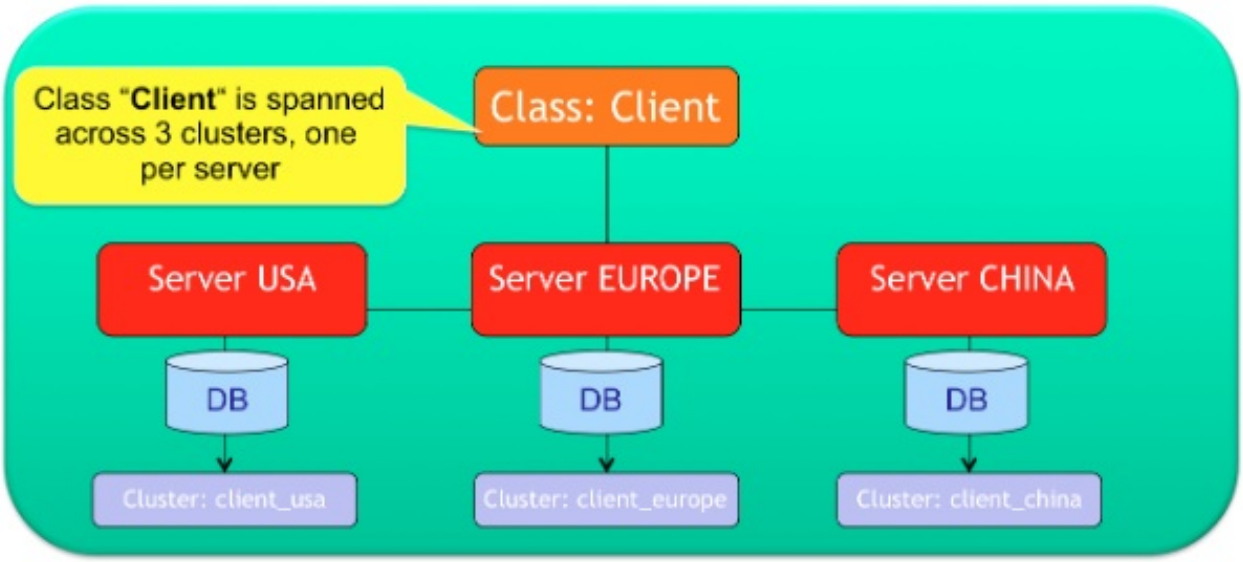
Sharding

OrientDB supports sharding of data at class level, by using multiple clusters[clusters] per 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.



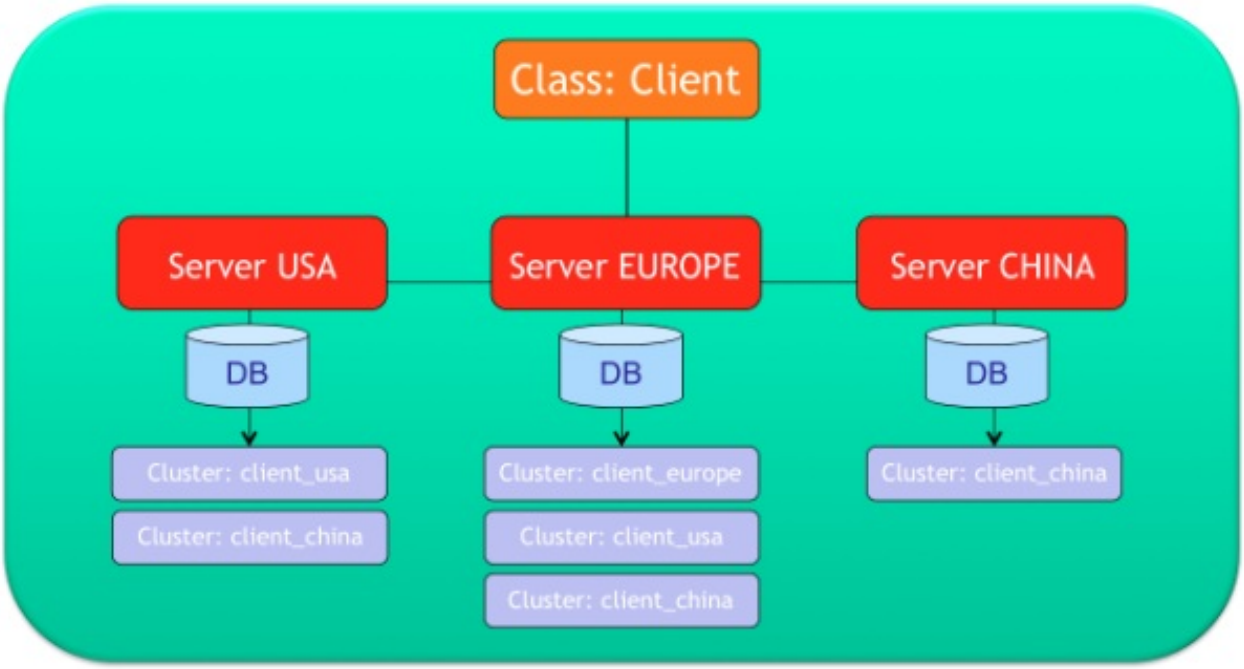
Shards, based on clusters, work against indexed and non-indexed class/clusters.

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 "china" nodes
- `client_europe`, will be managed by all the nodes (it would be equivalent as writing `“<NEW_NODE>”`, see cluster "*", the default one)
- `client_china`, will be managed only by "china" node



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#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 ne record is created. This avoids conflicts and allow 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 CHINA nodes, so the "create record" operation is sent to both "usa" (locally) and "china" 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 receive 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:

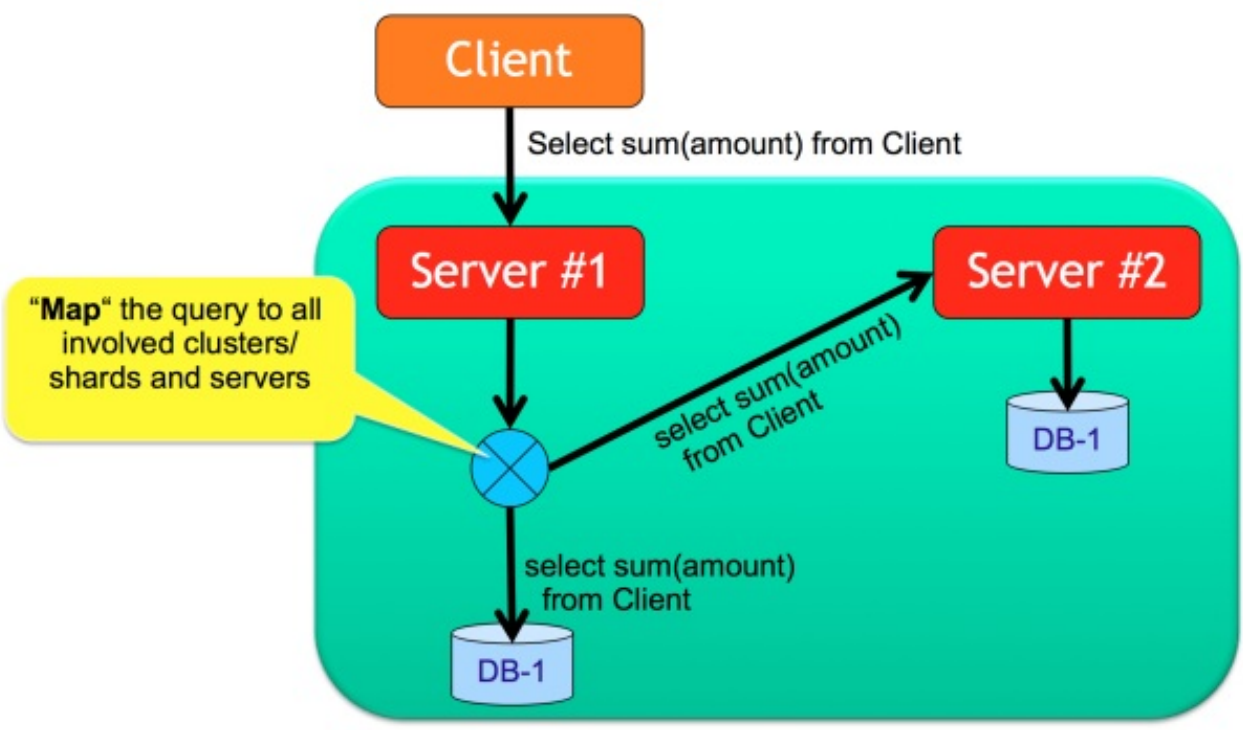
```
select from cluster:client_china
```

In this case the local node (USA) is used, because `client_china` is hosted on local node.

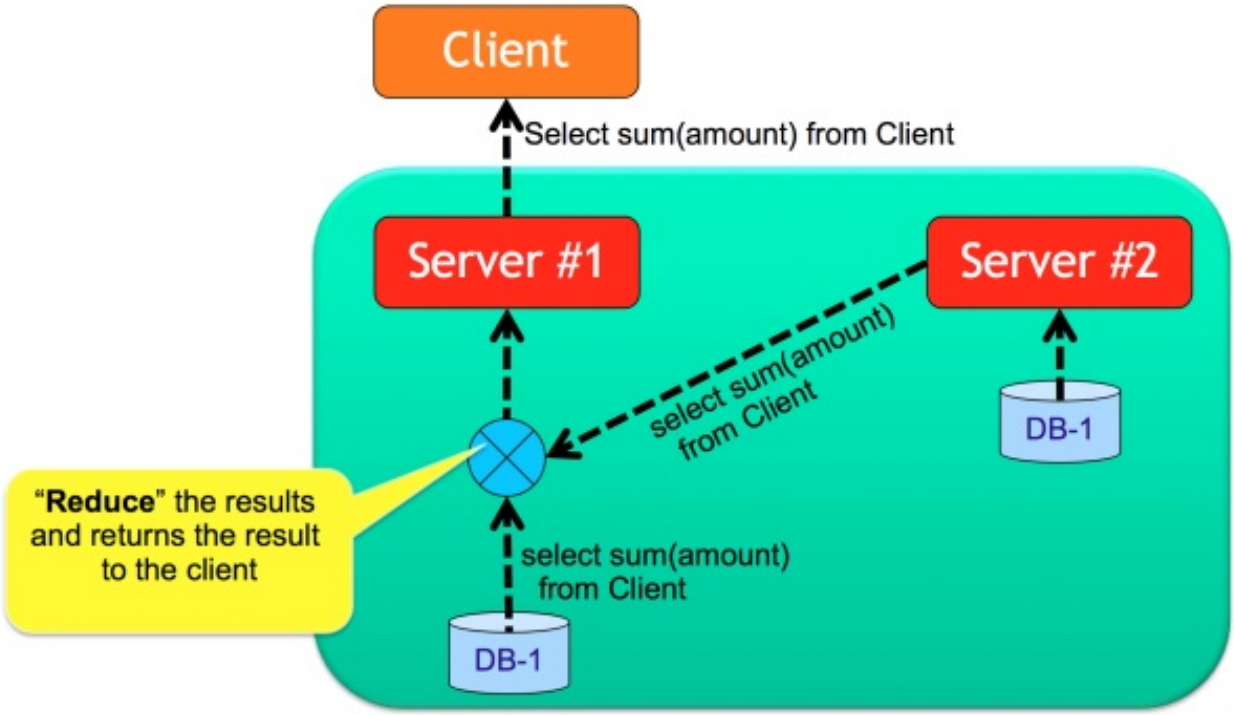
Map-Reduce

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
```



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 Client CLUSTER:client_usa SET 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" );
```

Limitation

1. 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).
2. Not complete merging of results for all the projections. Some functions like AVG() doesn't work on map/reduce

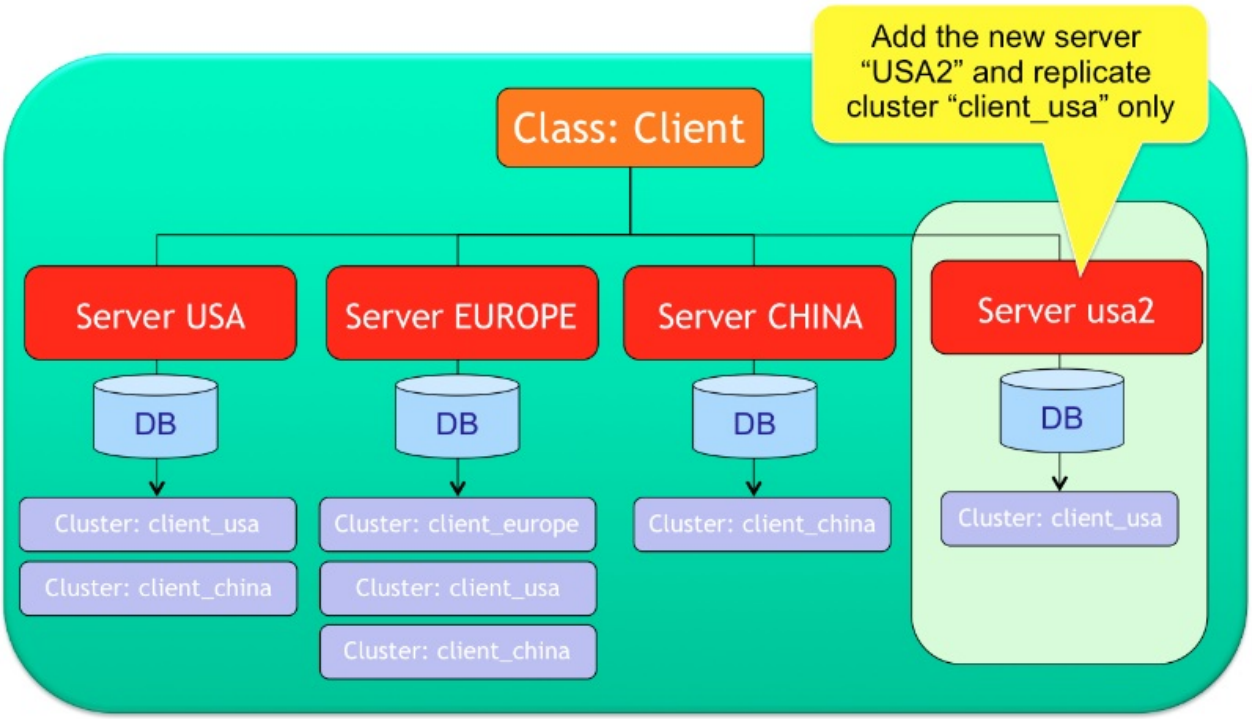
Indexes

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".



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.OHaze
</properties>
```

Backup & Restore

OrientDB supports backup and restore operations like any RDBMS.

Backup executes a complete backup against the currently opened database. The backup file is compressed using the ZIP algorithm. To restore the database use the Restore Database command. Backup is much faster than Export Database. Look also to Export Database and Import Database commands. Backup can be done automatically by enabling the Automatic-Backup Server plugin.

When to use backup and when export?

Backup does a consistent copy of database, all further write operations are locked waiting to finish it. The database is in read-only mode during backup operation. If you need an read/write database during backup setup a distributed cluster of nodes.

Export, instead, doesn't lock the database and allow concurrent writes during the export process. This means the exported database could have changes executed during the export.

Backup database

Starting from v1.7.8, OrientDB comes with the script "backup.sh" under the "bin" directory. This script executes the backup by using the console. Syntax:

```
./backup.sh <dburl> <user> <password> <destination> [<type>]
```

Where:

- **dburl**: database URL
- **user**: database user allowed to run the backup
- **password**: database password for the specified user
- **destination**: destination file path (use .zip as extension) where the backup is created
- **type**: optional backup type, supported types are:
  - **default**, locks the database during the backup
  - **lvm**, uses LVM copy-on-write snapshot to execute in background

Example of backup against a database open with "plocal":

```
./backup.sh plocal:../database/testdb admin admin /dest/folder/backup.zip
```

Non-Blocking Backup

backup.sh script supports non-blocking backup if the OS supports LVM). Example:

```
./backup.sh plocal:../database/testdb admin admin /dest/folder/backup.zip lvm
```

Same example like before, but against a remote database hosted on localhost:

```
./backup.sh remote:localhost/testdb root rootpwd /dest/folder/backup.zip lvm
```

For more information about LVM) and Copy On Write (COS) look at:

- File system snapshots with LVM
- LVM snapshot backup

Using the console

You can also use the console to execute a backup. Below the same backup like before, but using the console.

```
orientdb> connect plocal:../database/testdb admin admin
orientdb> backup database /dest/folder/backup.zip
Backup executed in 0,52 seconds
```

Restore database

Use the console to restore a database. Example:

```
orientdb> restore database /backups/mydb.zip
Restore executed in 6,33 seconds
```

See also

- Backup Database
- Restore Database
- Export Database
- Import Database
- Console-Commands

Export & Import

OrientDB supports export and import operations like any RDBMS.

The Export command exports the current opened database to a file. The exported file is in JSON format using the Export-Format. By default the file is compressed using the GZIP algorithm. The Export/Import commands allow to migrate the database between different releases of OrientDB without loosing data. If you receive an error about the database version, export the database using the same version of OrientDB that has generated the database.

Export doesn't lock your database, but browses it. This means that concurrent operation can be executed during the export, but the exported database couldn't be the exact replica when you issued the command because concurrent updates could occurs. If you need a snapshot of database at a point in a time, please use Backup.

Once exported, use the Import to restore it. The database will be imported and will be ready to be used. Look also to Backup Database and Restore Database commands.

When to use backup and when export?

Backup does a consistent copy of database, all further write operations are locked waiting to finish it. The database is in read-only mode during backup operation. If you need an read/write database during backup setup a distributed cluster of nodes.

Export, instead, doesn't lock the database and allow concurrent writes during the export process. This means the exported database could have changes executed during the export.

Export Example

```
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.
```

Import Example

```
> 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
```

See also

- Export-Format
- Restore Database
- Export Database
- Import Database
- Console-Commands

Export format

This page contains the format used by Export Database and Import Database tools. The file is 100% JSON compliant.

See also

- Export Database
- Import Database

Sections

Info Section

First section resuming the database information and all the version used to check compatibility on import.

```
{
  "info":{
  "name": <database-name>,
  "default-cluster-id": <default-cluster-id>,
  "exporter-version": <exporter-format>,
  "engine-version": <engine-version>,
  "storage-config-version": <storage-version>,
  "schema-version": <schema-version>,
  "mvrbtree-version": <mvrbtree-version>
}
```

Parameter	Description	JSON Type
database-name	Name of database	String
default-cluster-id	Cluster Id used by default. Range: 0-32,767	Integer
exporter-format	Version of Database exporter	Integer
engine-version	Version of OrientDB	String
storage-version	Version of Storage layer	Integer
schema-version	Version of the schema exporter	Integer
mvrbtree-version	Version of the MVRB-Tree	Integer

Example

```
{
  "info":{
  "name": "demo",
  "default-cluster-id": 2,
  "exporter-version": 2,
  "engine-version": "1.7-SNAPSHOT",
  "storage-config-version": 2,
  "schema-version": 4,
  "mvrbtree-version": 0
}
```

Clusters Section

Contains the database structure in clusters.

```
"clusters": [
  {"name": <cluster-name>, "id": <cluster-id>, "type": <cluster-type>}
]
```

Parameter	Description	JSON Type
cluster-name	Name of cluster	String
cluster-id	Cluster id. Range: 0-32,767	Integer
cluster-type	Cluster type between "PHYSICAL", "LOGICAL" and "MEMORY"	String

Example

```
"clusters": [
  {"name": "internal", "id": 0, "type": "PHYSICAL"},
  {"name": "index", "id": 1, "type": "PHYSICAL"},
  {"name": "default", "id": 2, "type": "PHYSICAL"}
]
```

Schema Section

Contains the schema as classes and properties.

```
"schema":{
  "version": <schema-version>,
  "classes": [
    { "name": <class-name>,
      "default-cluster-id": <default-cluster-id>,
      "cluster-ids": [<cluster-ids>],
      "properties": [
        { "name": <property-name>,
          "type": <property-type>,
          "mandatory": <property-is-mandatory>,
          "not-null": <property-not-null> }
      ]
    }
  ]
}
```

Parameter	Description	JSON Type
schema-version	Version of the record where the schema is stored. Range: 0-2,147,483,647	Integer
class-name	Class name	String
default-cluster-id	Default cluster id for the class. Represents the cluster where records will be stored	Integer
cluster-ids	Array of cluster ids where the class records can be stored. The first is always the `<default-cluster-id>`	Array of Integer
property-name	Name of the property	String
property-type	Property type between the supported ones	String
property-is-mandatory	Is this property mandatory? true or false	Boolean
property-not-null	The property can't accept null? true or false	Boolean

Example

```
"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}
      ]
    }
  ]
}
```

Records Section

Contains the exported records with metadata (prefixed by @) and fields.

```
"records": [
  {
    "@type": <record-type>,
    "@rid": <record-id>,
```

```
    "@version": 0,
    "@class": <record-class>,

    <field-name>: <field-value>,

    ["@fieldTypes": "<field-name>=<field-type>"]
  }
]
```

Parameter	Description	JSON Type
record-type	Record type: d = document, b = binary, f = flat	String
record-id	RecordID in the format `#<cluster-id>:<cluster-position>`	String
record-version	Record version from 0 to 2,147,483,647	Integer
record-class	Record class name	String
field-name	Field name	String
field-value	Field value	Any
field-type	Optional, it's the field type: 'l'=Long, 'f'=Float, 'd'=Double, 's'=Short, 't'=Datetime, 'd'=Date, 'c'=Decimal, 'b'=Byte	Any

Example

```
"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"
  }
]
```

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": "flat", "id": 7, "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": "Acc
          "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": "Gr
          "properties": [
          ]
        },
        {"name": "GraphMotocycle", "default-cluster-id": 30, "cluster-ids": [30], "super-class
          "properties": [
          ]
        },
        {"name": "GraphVehicle", "default-cluster-id": 28, "cluster-ids": [28], "super-class":
          "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-clas
            {"name": "out", "type": "LINK", "mandatory": false, "not-null": false, "linked-cla
          ]
        },
        {"name": "OGraphVertex", "default-cluster-id": 26, "cluster-ids": [26], "short-name":
          "properties": [
            {"name": "in", "type": "LINKSET", "mandatory": false, "not-null": false, "linked-c
            {"name": "out", "type": "LINKSET", "mandatory": false, "not-null": false, "linked-
          ]
        },
        {"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, "l
          ]
        },
        {"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, "linke
          ]
```

```
    },
    {"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, "li
      ]
    },
    {"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
        {"name": "name", "type": "STRING", "mandatory": false, "not-null": false, "min": "
        {"name": "nick", "type": "STRING", "mandatory": false, "not-null": false, "min": "
        {"name": "photo", "type": "TRANSIENT", "mandatory": false, "not-null": false},
        {"name": "registeredOn", "type": "DATETIME", "mandatory": false, "not-null": false
        {"name": "surname", "type": "STRING", "mandatory": false, "not-null": false, "min"
      ]
    },
    {"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, "cl
      "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, "linke
      ]
    },
    {"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
          {"name": "date", "type": "DATE", "mandatory": false, "not-null": false, "min": "20
          {"name": "id", "type": "INTEGER", "mandatory": false, "not-null": false},
          {"name": "replyTo", "type": "LINK", "mandatory": false, "not-null": false, "linked
          {"name": "text", "type": "STRING", "mandatory": true, "not-null": false, "min": "1
        ]
      },
      {"name": "classclassIndexManagerTestClassTwo", "default-cluster-id": 25, "cluster-ids"
      },
      {"name": "newV", "default-cluster-id": 31, "cluster-ids": [31], "super-class": "OGraph
        "properties": [
          {"name": "f_int", "type": "INTEGER", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "vertexA", "default-cluster-id": 32, "cluster-ids": [32], "super-class": "OGr
        "properties": [
          {"name": "name", "type": "STRING", "mandatory": false, "not-null": false}
        ]
      },
      {"name": "vertexB", "default-cluster-id": 34, "cluster-ids": [34], "super-class": "OGr
        "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: Starting from OrientDB 2.0 you can use the OrientDB-ETL module to import data from RDBMS. You can use ETL also with 1.7.x by installing it as 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 RDBMS to Document Model

This guide is to import an exported relational database into OrientDB using the Document Model. If you're using the Graph Model, look at Import into Graph Model.

OrientDB supports a subset of SQL, so importing a database created as "Relational" is straightforward. You can import a database using the API, the OrientDB Studio visual tool or the Console. In this guide the console is used.

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

Since the Relational Model hasn't Object Oriented concepts you can create a class per table in OrientDB. Furthermore in the RDBMS references One-2-Many are inverted from the target table to the source one. In OrientDB the Object Oriented model is respected and you've a collection of links from POST to COMMENT instances. In a RDBMS you have:

```
Table POST    <- (foreign key) Table COMMENT
```

In OrientDB the Document model uses Links to manage relationships:

```
Class POST ->* (collection of links) Class COMMENT
```

Export your Relational Database

Most of Relational DBMSs provide a way to export a database in SQL format. What you need is a text file containing the SQL INSERT commands to recreate the database from scratch. Take a look to the documentation of your RDBMS provider. Below the link to the export utilities for the most common RDBMSs:

- MySQL: http://www.abbeyworkshop.com/howto/lamp/MySQL_Export_Backup/index.html
- Oracle: http://www.orafaq.com/wiki/Import_Export_FAQ
- MS SqlServer: http://msdn.microsoft.com/en-us/library/ms140052.aspx

At this point you should have a `.sql` file containing the Relational database exported in SQL format like this:

```
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' );
```

Modify the SQL script

What we're going to do is to change the generated SQL file to be imported into a OrientDB database. Don't execute the following commands but include them into the SQL file to be executed in batch mode by the OrientDB Console.

What database to use?

Before to import the database you need an open connection to a OrientDB database. You can create a brand new database or use an existent one. You can use a volatile in-memory only database or a persistent disk-based one.

For persistent databases you can choose to create it in a remote server or locally using the "plocal" mode avoiding the server at all. This is suggested to have better performance on massive insertion.

Create a new database

Use the embedded mode

```
create database plocal:/tmp/db/blog admin admin plocal document
```

This will create a new database under the directory "/tmp/db/blog".

Use the remote mode

Or start a OrientDB server and create a database using the "remote" protocol in the connection URL. Example:

```
create database remote:localhost/blog root dkdf383dhdsj plocal document
```

NOTE: When you create a remote database you need the server's credentials to do it. Use the user "root" and the password stored in `config/orientdb-server-config.xml` *file.*

Use an existent database

Use the embedded mode

If you already have a database where to import, just open it:

```
connect plocal:/tmp/db/blog admin admin
```

Use the remote mode

```
connect remote:localhost/blog admin admin
```

Declare the 'massive insert' intent

In order to obtain the maximum of performance you can tell to OrientDB what you're going to do. These are called "Intents". The "Massive Insert" intent will auto tune the OrientDB engine for fast insertion.

Add the following line:

```
DECLARE INTENT massiveinsert
```

Create the classes, one for tables

Since the Relational Model hasn't Object Oriented concepts you can create a class per table. Change the `CREATE TABLE ...` statements with `CREATE CLASS`:

```
CREATE CLASS POST
CREATE CLASS COMMENT
```

Pseudo Object Oriented database

This is the case when your Relational database was created using a OR-Mapping tool like Hibernate or Data Nucleus (JDO).

In this case you have to re-build the original Object Oriented structure directly in OrientDB using the Object Oriented capabilities of OrientDB.

Remove not supported statements

Leave only the `INSERT INTO` statements. OrientDB supports not only INSERT statement but for this purpose is out of scope.

Create links

At this point you need to create links as relationships in OrientDB. The Create Link command creates links between two or more records of type Document. In facts in the Relational world relationships are resolved as foreign keys.

Using OrientDB, instead, you have direct relationship as in your object model. So the navigation is from *Post* to *Comment* and not viceversa as for Relational model. For this reason you need to create a link as **INVERSE**.

Execute:

```
CREATE LINK comments TYPE linkset FROM comment.postId To post.id INVERSE
```

Remove old constraints

This is an optional step. Now you've direct links the field 'postId' has no more sense, so remove it:

```
UPDATE comment REMOVE postId
```

Expected output

After these steps the expected output should be similar to that below:

```
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
```

Import the records

Now you have modified the SQL script execute it by invoking the console tool in batch mode (text file just created as first argument). Example of importing the file called "database.sql":

```
$ console.sh database.sql
```

Enjoy

That's all. Now you've a OrientDB database where relationships are direct without JOINS.

Now enjoy with your new document-graph database and the following queries:

Select all the post with comments:

```
orientdb> select * from post where comments.size() > 0
```

Select all the posts where comments contain the word 'flame' in the text property (before as column):

```
orientdb> select * from post where comments contains ( text like '%flame%' )
```

Select all the posts commented today. In this case we're assuming a property "date" is present in Comment class:

```
orientdb> select * from post where comments contains ( date > '2011-04-14 00:00:00' )
```

To know more about other SQL commands look at SQL commands.

This is a command of the Orient console. To know all the commands go to 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

This guide explains how to export a graph from Neo4j* and import it into OrientDB in 3 easy steps. If you want to know more about the differences between OrientDB and Neo4j, take a look at OrientDB vs Neo4j.

1) Download Neo4j Shell Tools plugin

In order to export the database in GraphML format, you need the additional neo4j-shell-tools plugin:

- Download it from the Neo4j web site
- Extract the ZIP content inside Neo4j `lib` folder

2) Export the Neo4j database to a file

Now launch the Neo4j-Shell tool located in the `bin` directory and execute the following command:

```
export-graphml -t -o /tmp/out.graphml
```

Where `/tmp/out.graphml` is the location to save the file in GraphML format.

Example:

```
$ 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)$ export-graphml -t -o /tmp/out.graphml
Wrote to GraphML-file /tmp/out.graphml 0. 100%: nodes = 302 rels = 834 properties = 4221 tim
```

3) Import the database into OrientDB

The third and last step can be done in 2 ways, based on the OrientDB version you are using.

3a) With OrientDB 2.0.x or further

If you have OrientDB 2.0, this is the suggested method because it's easier and is able to import Neo4j labels as OrientDB classes automatically.

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

orientdb {db=test}> import database /tmp/out.graphml

Importing GRAPHML database database from /tmp/out.graphml...
Transaction 8 has been committed in 12ms
```

3b) With OrientDB 1.7 or previous

This method uses the standard Gremlin importer, but doesn't take in consideration any label declared in Neo4j, so everything is imported as V (base Vertex class) and E (base Edge class). After importing you could need to refactor your graph element to fit in a more structured schema.

To import the GraphML file into OrientDB complete the following steps:

- execute the Gremlin console located in `$ORIENTDB_HOME/bin`
- create a new graph with the command `g = new OrientGraph("plocal:/tmp/db/test");` specifying the path of your new OrientDB Graph Database, `/tmp/db/test` in this case
- execute the command `g.loadGraphML('/tmp/out.graphml');` where `/tmp/out.graphml` is the path of the exported file

Example:

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

```
         \,,,/
         (o o)
-----oOOo-(_)-oOOo-----
gremlin> g = new OrientGraph("plocal:/tmp/db/test");
==>orientgraph[plocal:/db/test]
gremlin> g.loadGraphML('/tmp/out.graphml');
==>null
gremlin> quit
```

Congratulations! The database has been imported into OrientDB.

---

*Neo4j is a registered trademark of Neo Technology Inc.

Logging

OrientDB uses the Java Logging framework bundled with the Java Virtual Machine.

Supported levels are those contained in the JRE class java.util.logging.Level:

- SEVERE (highest value)
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST (lowest value)

By default 2 loggers are installed:

- **console**, as the output of the shell/command prompt that starts the application/server. Can be changed by setting the variable `log.console.level`
- **file**, as the output to the log files. Can be changed by setting the `log.file.level`

Configuration file

The logging strategies and policies can be configured using a file following the Java syntax: Java Logging configuration.

Example taken from **orientdb-server-log.properties**:

```
# 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
```

To tell to the JVM where the properties file is placed you need to set the *"java.util.logging.config.file"* system property to it. Example:

```
$ java -Djava.util.logging.config.file=mylog.properties ...
```

Set the logging level

To change the logging level without modify the logging configuration just set the *"log.console.level"* and *"log.file.level"* system variables to the requested levels.

In the server configuration

Open the file **orientdb-server-config.xml** and add or update these lines at the end of the file inside the `<properties>` section:

```
<entry value="fine" name="log.console.level" />
<entry value="fine" name="log.file.level" />
```

At startup

Set the system property *"log.console.level"* and *"log.file.level"* to the levels you want using the -D parameter of java.

Example:

```
$ java -Dlog.console.level=FINE ...
```

At run-time

The system variable can be setted at startup using the `System.setProperty()` API. Example:

```
public void main(String[] args){
  System.setProperty("log.console.level", "FINE");
  ...
}
```

Install Log formatter

OrientDB Server uses own LogFormatter. To use the same by your application call:

```
OLogManager.installCustomFormatter();
```

LogFormatter is installed automatically by Server. To disable it define the setting `orientdb.installCustomFormatter` to `false`. Example:

```
java ... -Dorientdb.installCustomFormatter=false=false ...
```

Enterprise Edition

This is the main guide on using **OrientDB Enterprise Edition**. For more information look at OrientDB Enterprise Edition.

Enterprise Edition is in Beta stage, ask for a Trial by writing to: info@orientechnologies.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@orientechnologies.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'... [OServerPluginMa

***********************************************
*       ORIENTDB  -  ENTERPRISE EDITION       *
*                                             *
* Copyrights (c) 2013 Orient Technologies LTD *
***********************************************
* Version...: 1.6.2                           *
* License...: 2P2tA1EO8oOoS/WkR2/023kdks922JDw *
* Expires in: -25 days                        *
***********************************************
```

NOTE: OrientDB Enterprise Plugin and OrientDB Server must be of the same main version. Workbench 1.7.x works against all Agents 2.7.x. If you don't have the right version please write to the Orient Technologies: info@orientechnologies.com.

OrientDB Enterprise Workbench

Download

Download the right OrientDB Workbench distribution, using the same Agent version:

- Workbench Web Application v. **1.7.4**:
  - Windows users: ZIP
  - MacOSX, Linux, Any Unix like OSs: TAR.GZ
  - Help

- Workbench Web Application v. **1.6.2**:
  - Windows users: ZIP
  - MacOSX, Linux, Any Unix like OSs: TAR.GZ
  - Help

- Workbench Web Application v. **1.6.1**:
  - Windows users: ZIP
  - MacOSX, Linux, Any Unix like OSs: TAR.GZ

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:

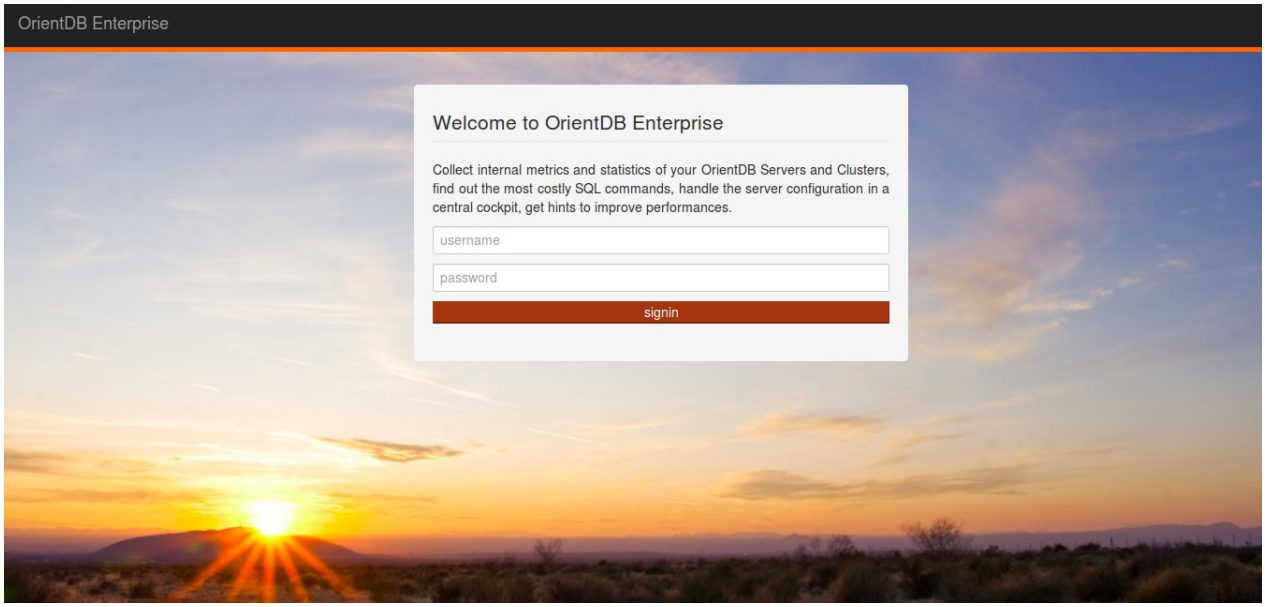
- **start-workbench.sh** for MacOSX, Linux and Unix users
- **start-workbench.bat**, for Windows users

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'
```

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).



For the complete guide goto Workbench Guide.

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
```

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 MemoryLocker.lockMemoryPerformance-Tuning#remote_connections.suggestions]].

Could not find artifact com.orientechnologies:orientdb-core:jar:1.0-SNAPSHOT in orientechnologies-repository(http://www.orientechnologies.com/listing/m2)

Your maven configuration points to the old Orient Technologies repository: http://www.orientechnologies.com/listing/m2. Follow this configuration: [### Record id invalid -1:-2

This message is relative to a temporary Concepts#recordidrecordid.]] 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 major 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.

Why it's so slow with index and massive insertion?

Try to enable automatic flush of index nodes. Via API:

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

or via configuration:

```
java ... -Dmvrbtree.optimizeThreshold=-1 ...
```

Error: com.orientechnologies.orient.core.db.record.ORecordLazySet cannot be cast to com.orientechnologies.orient.core.db.record.OIdentifiable

This happens if you've migrated a database created with an old version of OrientDB where indexes were managed in different way. Just drop and recreate the indexes.

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

This is because the database is locked by another process is using it. To fix:

- check there's no a process is using OrientDB. Most of the times a OrientDB Server is running. Just shutdown it and retry
- set the storage.keepOpen setting to false

012-04-20 11:29:56:132 SEVE Received unread response from /10.0.0.2:2434 for session=0, probably corrupted data from the network connection. Cleared dirty data in the buffer (330 bytes): [----7-c-o-m-.-o-r-i-e-n-t-e-c-h-n-o-l-o-g-i-e-s-.-c-o-m-m-o-n-.-c-o-n-c-u-r-.-l-o-c-k-.-O-L-o-c-k...] [OChannelBinaryClient

This is a message of an old version of OrientDB: upgrade it.

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. To fix this problem you can:

- if you're running in a multi-thread application and your JVM is the only client is writing to the database then disabling the Level1 cache could be enough.
- If you're using the GraphDB API look at: concurrency

If you want to leave the MVCC and write code concurrency proof:

```
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 container couldn't be happy 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.

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

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.

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 it's 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

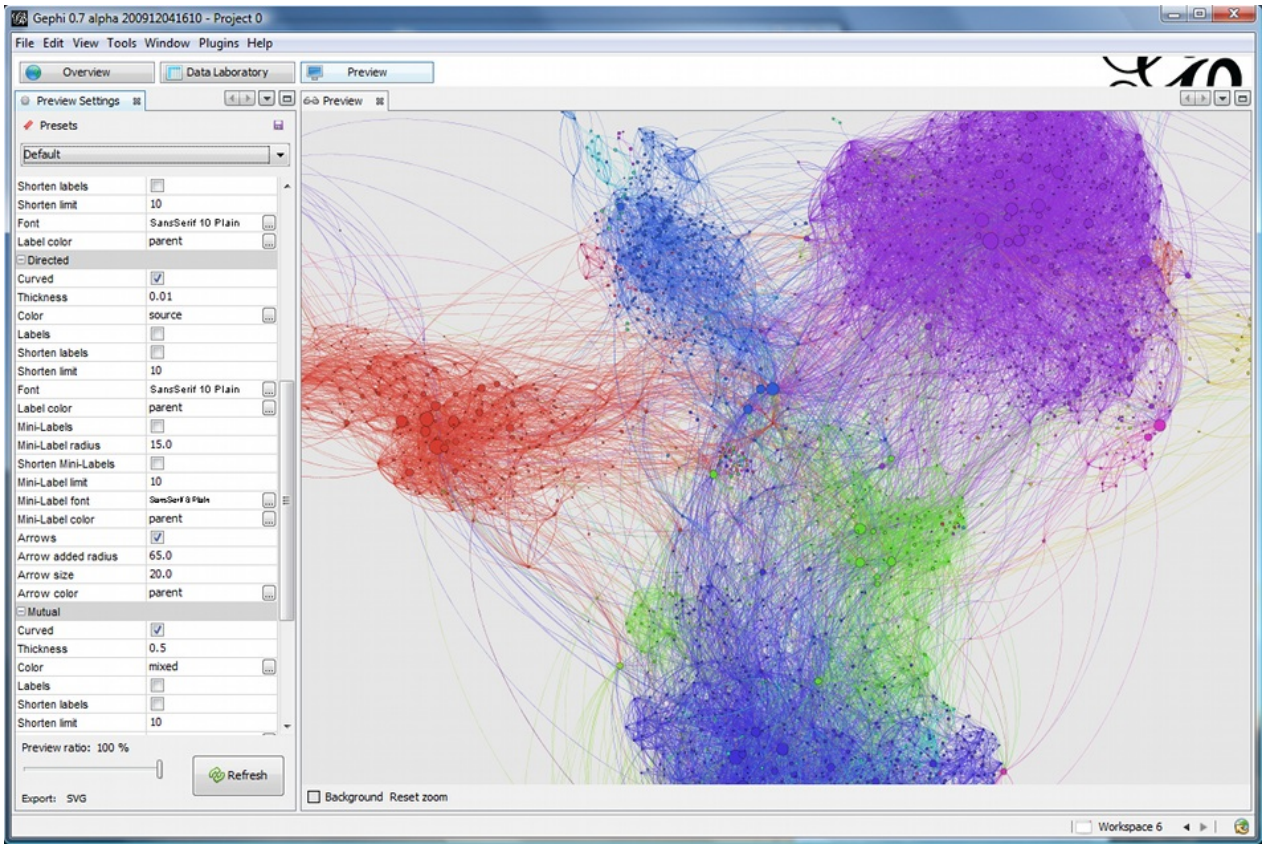
In this section we present a step-by-step guide to Rexster-ify an OrientDB graph.
We assume that:

- 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]
[INFO] RexsterApplicationGraph - Graph [tinkergraph] - configured with allowable namespa
[INFO] GraphConfigurationContainer - Graph tinkergraph - tinkergraph[vertices:6 edges:6
[INFO] RexsterApplicationGraph - Graph [tinkergraph-readonly] - configured with allowabl
[INFO] GraphConfigurationContainer - Graph tinkergraph-readonly - (readonly)tinkergraph
[INFO] RexsterApplicationGraph - Graph [gratefulgraph] - configured with allowable names
[INFO] GraphConfigurationContainer - Graph gratefulgraph - tinkergraph[vertices:809 edg
[INFO] GraphConfigurationContainer - Graph sailgraph - sailgraph[memorystore] loaded
[INFO] GraphConfigurationContainer - Graph my-orient-graph - orientgraph[remote:orientho
[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..packag
[INFO] WebServer - No servlet initialization parameters passed for configuration: admin
[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
```

- 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



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 major.

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

Upgrade

OrientDB uses the Semantic Versioning System (http://semver.org) where given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes,
- MINOR version when you add functionality in a backwards-compatible manner
- PATCH version when you make backwards-compatible bug fixes.

So between PATCH versions the compatibility is assured (example 1.7.0 -> 1.7.8). Between MINOR and MAJOR versions you could export and re-import the database. See below in the column "Database":

Compatibility Matrix

FROM	TO	Guide	Blueprints	Database	Binary Protocol	H Pro
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

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 different way. Below 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've 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

Since OrientDB 1.7 RidBag is default collection that manage adjacency relations in graphs. While the older database managed by MVRB-Tree are fully compatible, you can update your database to 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()

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.

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	
API	ODatabaseRecord.getLevel1Cache()	ODatabase
API	ODatabaseRecord.getLevel2Cache()	Not availab

Configuration	OGlobalConfiguration.CACHE_LEVEL1_ENABLED	OGlobalCo
Configuration	OGlobalConfiguration.CACHE_LEVEL2_ENABLED	Not availab

No more LOCAL engine

We completely drop the long deprecated LOCAL Storage. If your database was created using "LOCAL:" then you have to export it with the version you was 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.

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 bases on disk, all the 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 crc32 check sum of page content. This information is used to check storage integrity after DB crash. To start 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 data file and cluster position, with extension .cpm

File System

To speed up the 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, where each entry of this list is a fixed size element which is the pointer to the physical position of record in data file.

Because data file is paginated, this pointer consist of 2 items: page index (long value) and position of record inside page (int value), so each record pointer consumes 12 bytes.

Creation of new records in cluster

When a new record is inserted, a new pointer is added to the list so index of this pointer becomes cluster position. The list is 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 record pointer is transformed in record tombstone. You can think about 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 could be mitigated with a periodic "offline compaction" by performing database export/import. In such case records cluster positions will be changed (tombstones will be ignored during export) and the lost space will be revoked. So during the import process, the cluster positions can change.

Migration of RID

OrientDB import tool uses a manual hash index (by default the name is '___exportImportRIDMap') to map the old record ids and new record ids.

Write Ahead (operation) Log (WAL)

Write Ahead Log, WAL from now, 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 on these components. WAL is append only data structure, you can think about it like a list of records which contains 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 the trade off between performance and durability.

Put the WAL to a separate disk

It's strongly recommended to store WAL records on separate 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 data are needed to rollback transactions are stored. WAL records can not be used to restore index content after crash, in such case automatic indexes are rebuild. In FULL mode indexes can be restored after DB crash without rebuild. You can change index durability mode by setting the property **index.txMode**.

You can find more details about WAL **here**.

File types

PLocal storage writes the database on file system using different files. Below all the extensions:

- **.cpm**, contains the mapping between real physical positions and cluster positions. If you delete 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 (is used internally)
- **.irs**, is the RID set file for not unique index

How it works (Internal)

Basically paginated storage is nothing more than 2-level disk cache which works together with write ahead log.

Every file is spitted on pages, and each file operation is atomic at page level. 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 **Read Cache** (implementation is based on 2Q cache algorithm) and **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 files does not exist it will be automatically created. Id of file is stored in special meta data file and always will belong to 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 write lock or read lock if you read data and your single file page is shared across several data structures. Write lock must be acquired whether single page are used between several data structures or not. Write lock is needed to prevent flush of inconsistent pages to the disk inside of background “data flush” thread of 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, in other words indicate cache that you do 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 at first Read Cache looks it in one of LRU lists. There are two of them for data which are accessed several times and then not accessed for very long period of item (it consumes 25% of memory) and data which are accessed frequently for long period of time (it consumes 75% of memory).

If page is absent in LRU queues, then Read Cache asks to the Write Cache to load data from the disk.

If we are lucky and pages which are queued to flush are still in Write Queue of Write Cache it will be retrieved from there or otherwise Write Cache will load data from file on the 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 page and this page is marked as dirty this page is put into the Write Cache which adds it to the Write Queue. Write Queue can be considered as ring buffer where all the pages are sorted by its position on the disk. This trick allows to minimize disk head movements during pages flush. What is more interesting that pages are always flushed in background in “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 page durable and not durable.

So 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 component which will present part of your data structure and extend it from com.orientechnologies.orient.core.storage.impl.local.paginated.ODurablePage class. This class has similar methods for manipulation of data in off heap page but also it tracks all changes are done to the page and we can always return diff between old/new states of page using 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

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
 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:

```
-storage.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
  else
    go through all atomic operation records from last to first, set old page's content, set
```

```
    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.

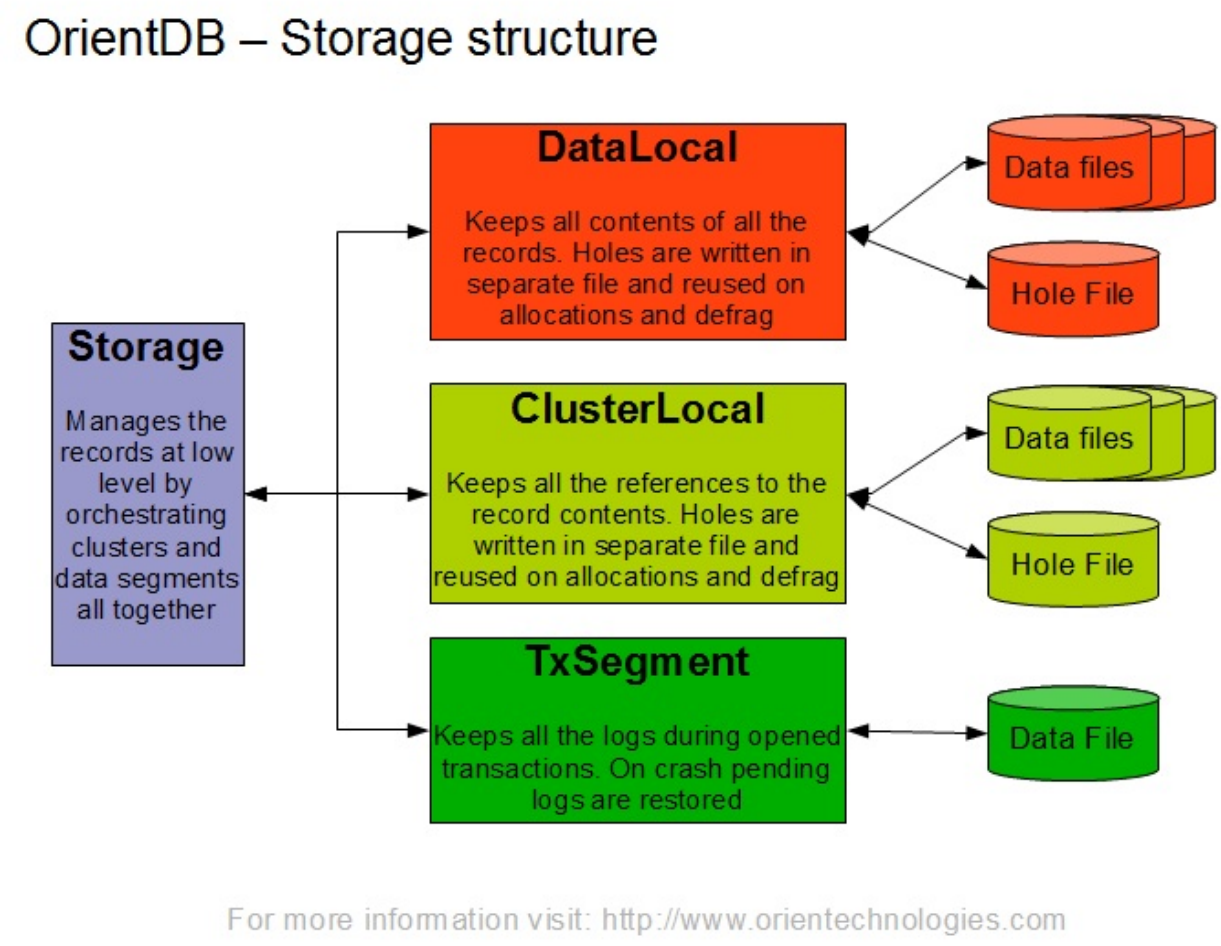
Second case a bit more tricky. Because WAL is append only log, there is two possible sub-cases, lets suppose we have 3 pages after 2-nd (broken) flush. First and first half of second page were flushed during first flush and second half of second page and third page were flushed during second flush. Because 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.



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:

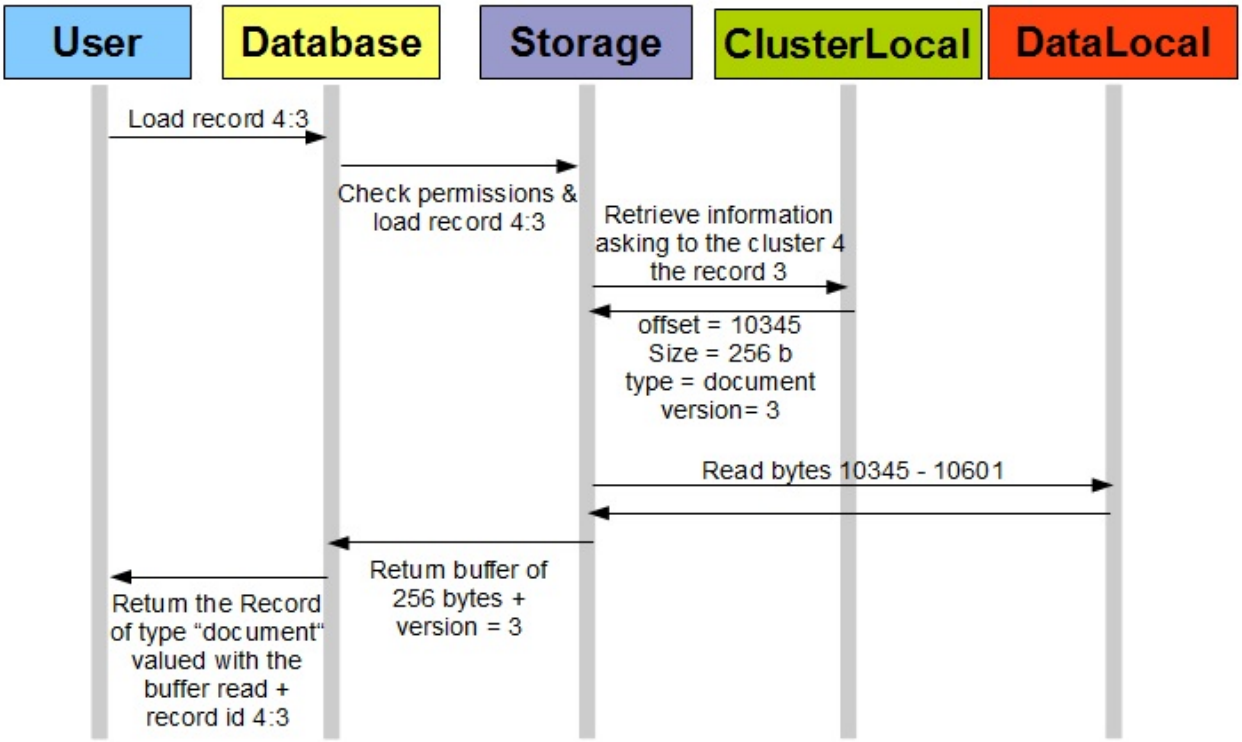
- default.0.oda
- default.odh

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



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

Limitations running distributed

OrientDB v 2.0.x 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
- 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.

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: 80*.
- `RID_BAG_SBTREEBONSAI_TO_EMBEDDED_THRESHOLD` (`ridBag.sbtreeBonsaiToEmbeddedToThreshold`) - The threshold of LINKBAG conversion to embedded implementation. *Disabled by default*.

Interaction with remote clients

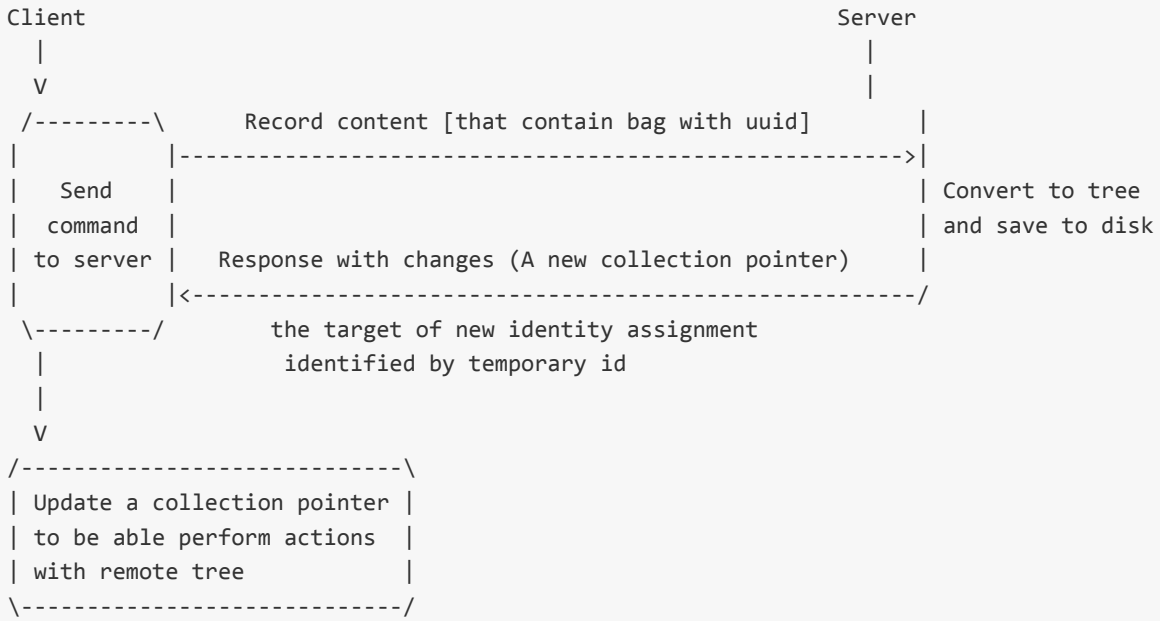
> NOTE: This topic is rather for contributors or driver developers. OrientDB users don't have to care about bag internals.

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:



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_L
| <FLOATING_POINT_LITERAL: <DECIMAL_FLOATING_POINT_LITERAL> | <HEXADECIMAL_FLOATING_POINT_LI
| <#DECIMAL_FLOATING_POINT_LITERAL: ("." (["0"-"9"]("0"-"9"].md)+))** (<DECIMAL_EXPONENT>)?
| <#HEXADECIMAL_FLOATING_POINT_LITERAL: "0" [(["0"-"9","a"-"f","A"-"F"]("x","X"].md))+ (".")
| <CHARACTER_LITERAL: "\'" (~[| "\\" (["n","t","b","r","f","\\","\'","\""]("\'","\\","\n","\
| <STRING_LITERAL: "\"" (~["\"","\\","\n","\r"]("0"-"7"].md))) | "\\" ([| ["0"-"7"]("n","t",
}

/* 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" | "
| <CONTAINS: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("N" |
| <CONTAINSALL: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("N"
| <CONTAINSKEY: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("N"
| <CONTAINSVALUE: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("
| <CONTAINSTEXT: ("C" | "c") ("O" | "o") ("N" | "n") ("T" | "t") ("A" | "a") ("I" | "i") ("N
| <MATCHES: ("M" | "m") ("A" | "a") ("T" | "t") ("C" | "c") ("H" | "h") ("E" | "e") ("S" | "
```

```
| <TRAVERSE: ("T" | "t") ("R" | "r") ("A" | "a") ("V" | "v") ("E" | "e") ("R" | "r") ("S" |
}

<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 | UpdateStatemer
    SelectStatement    :=    <SELECT> ( Projection )? <FROM> FromClause ( <WHERE> WhereClaus
    DeleteStatement    :=    <DELETE> <FROM> <IDENTIFIER> ( <WHERE> WhereClause )?
    UpdateStatement    :=    <UPDATE> ( <IDENTIFIER> | Cluster | IndexIdentifier ) ( ( <SET>
    UpdateItem    :=    <IDENTIFIER> <EQ> ( <NULL> | <STRING_LITERAL> | Rid | <INTEGER_LITERA
    UpdateAddItem    :=    <IDENTIFIER> <EQ> ( <STRING_LITERAL> | Rid | <INTEGER_LITERAL> |
    InsertStatement    :=    <INSERT> <INTO> ( <IDENTIFIER> | Cluster ) <LPAREN> <IDENTIFIER
    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> |
    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 )** )
    FieldOperator    :=    ( <DOT> <IDENTIFIER> <LPAREN> ( FilterItem ( "," FilterItem )** )
        |    ( "[<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> | DottedIden
    MatchesCondition    :=    FilterItem <MATCHES> <STRING_LITERAL>
    TraverseCondition    :=    <TRAVERSE> ( <LPAREN> <INTEGER_LITERAL> ( "," <INTEGER_LITERA
    TraverseFields    :=    <STRING_LITERAL>
    OrderBy    :=    <ORDER> <BY> <IDENTIFIER> ( "," <IDENTIFIER> )** ( <DESC> | <ASC> )?
    Limit    :=    <LIMIT> <INTEGER_LITERAL>
    Range    :=    <RANGE> Rid ( "," Rid )?

DOCUMENT END
```

Custom Index Engine

(Since OrientDB v 1.7)

Entry Points

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, St
      String valueContainerAlgorithm, ODocument metadata) throws OConfigurationException {
    return createLuceneIndex(oDatabaseRecord, indexType, valueContainerAlgorithm, metadata);
  }

  private OIndexInternal<?> createLuceneIndex(ODatabaseRecord oDatabaseRecord, String indexT
      ODocument metadata) {
    if (OClass.INDEX_TYPE.FULLTEXT.toString().equals(indexType)) {
      return new OLuceneFullTextIndex(indexType, LUCENE_ALGORITHM, new OLuceneIndexEngine<Se
          new OLuceneFullTextIndexManager(), indexType), valueContainerAlgorithm, metadata);
    } else if (OClass.INDEX_TYPE.SPATIAL.toString().equals(indexType)) {
      return new OLuceneSpatialIndex(indexType, LUCENE_ALGORITHM, new OLuceneIndexEngine<Set
          new OLuceneSpatialIndexManager(new OShapeFactoryImpl()), indexType), valueContaine
    }
    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

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

If you'd like to contribute to OrientDB with a patch follow the following steps:

- apply your changes,
- test that Test Suite hasn't been broken. Execute both commands:
  - ant clean test
  - mvn clean test

- if all the tests pass then do a **Pull Request** (PR) against **"develop"** branch on GitHub and write a comment about the change. Don't sent PR to "master" because we use that only for releasing

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.

Issue priorities

As soon as github issues does not provide way to manage issue prioritization, priorities of issues are defined as tags.

Priority	Definition
Critical	The OrientDB server in Production doesn't start up or the database is corrupted in anyway. Issues encountered in a test or development environment and enhancement requests should not be listed as Critical.
High	The Issue has impact on OrientDB functionality in Production, but against a non critical component. OrientDB still works.
Medium	The Issue hasn't impact on the OrientDB operation and can be bypassed with a round trip.
Low	Little or no impact on OrientDB. Cosmetic problem or minor enhancement.

The Team

If you want to contribute to the project, follow the Contributor rules.

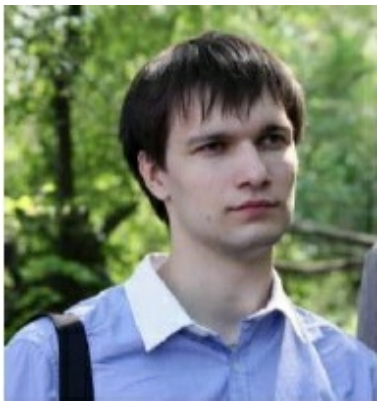
Committers

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 Orient Technologies 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.
Company Orient Technologies Ltd
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 Orient Technologies Ltd
   Links Twitter LinkedIn
   Since 2012

Luca Molino

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.

Gitter.io

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

@orientechno

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@orientechnologies.com

If you want more information about Commercial Support, Consultancy or Training, email us.

Table of Contents