Getting Started with OrientDB

A practical guide to learn, deploy, and customize OrientDB

Claudio Tesoriero

Getting Started with OrientDB

A practical guide to learn, deploy, and customize OrientDB

Claudio Tesoriero

Getting Started with OrientDB

Credits

Author
Claudio Tesoriero

Reviewers
Andrey Lomakin
Artem Orobets

Acquisition Editor
Kunal Parikh

Commissioning Editor
Harsha Bharwani

Technical Editors
Krishnaveni Haridas
Mrunmayee Patil

Project Coordinator
Deenar Satam

Proofreaders
Kelly Hutchinson
Joanna McMahon

Indexers
Tejal Soni
Priya Subramani

Graphics
Abhinash Sahu

Production Coordinator
Arvindkumar Gupta

Cover Work
Arvindkumar Gupta

About the Author

Claudio Tesoriero is an OrientDB Certified Developer and a senior software engineer with 20 years of experience in Information Technology. He started his career with the Italian Ministry of the Treasury before moving on to work for the Bull Group (`www.bull.com`), where he got involved in projects developed for Telecom Italia (`www.telecomitalia.it`) and in R&D projects developed in collaboration with the Rome Tor Vergata University. He then worked for FutureSpace Spa (`www.futurespace.it`) and he participated in the implementation of various projects for the government administration at the time. Currently, he is the cofounder of BaasBox, a solution of Backend as a Service based on the Play! Framework and OrientDB.

About the Reviewers

Andrey Lomakin is working as a software architect in Return On Intelligence projects.

He is an active committer of the OrientDB project, he is an author of composite and hash indexes. He has implemented several improvements in the SQL engine, mostly related to index usage.

His main areas of expertise include high performance computing and modern approaches to the implementation of business logic in enterprise applications: EDA, CQRS, Qi4j, and so on.

> I want to thank two people who have always supported me in my work on the OrientDB project.
>
> They are my mother, and my best friend Marina Melnik.
>
> Without their support, many already implemented features would still be in the planning stage.

Artem Orobets is a committer of the OrientDB community. He has provided contributions such as the introduction of composite indexes, improvements in index creation and processing speed, and improvements in query language.

He is currently working as a software engineer at Orient Technologies Ltd., where he designs and maintains OrientDB.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

In modern software applications, often there is the necessity to manage very large amounts of data. These are often unstructured and their schema may vary according to marketing rules or other external factors which you can't control. In this scenario, relational databases would not be the right choice.

This is where NoSQL databases come in, in particular the graph-based ones.

In graph databases, data is modeled like in a graph. Each piece of information is a node and links between them are edges.

They are a very young product and they were designed to satisfy modern necessities regarding the management of large amounts of data, often without a fixed schema or with a fluid schema that may vary very often even when the software is in production. Furthermore, this data could be related to each other in multiple ways. A great example of this is social networks.

They must manage billions of records and connect, traverse, and perform queries in a snap.

OrientDB is an open source document-graph NoSQL database born at the end of the last decade, but its algorithms and concepts have a long story that began in the late 90s when its authors wrote an object database.

Now, OrientDB is a very strong, mature, and robust platform. It has reached Version 1.5.0 and its community and committers are very active, and they are already working on a new version. OrientDB is used in production by many companies.

Some of these companies are listed at `https://github.com/nuvolabase/orientdb/wiki/Production-Deployments`.

What this book covers

This book is intended to be a beginner's guide to help you get started with OrientDB. It is a quick reference to the most common OrientDB tasks, from administrative ones to deployment, from designing a graph database to the different ways to perform queries and consume data.

Chapter 1, *Installing OrientDB*, provides an introduction to the OrientDB world as well as a quick guide to help you set up and run OrientDB.

Chapter 2, *Administering OrientDB*, explains the database architecture, data files structure, user management and security, and administration tools.

Chapter 3, *Programming OrientDB*, shows you how to perform operations against the database, covering the multiple modes supported by OrientDB to interact with it.

Chapter 4, *Performance Tuning*, gives some advice and tips for every possible use, since OrientDB can be used in several different scenarios.

Chapter 5, *Advanced Features*, covers several advanced features provided by OrientDB. And since it is a very active project, many of them are released and updated continuously. In this chapter, some of the features are discussed.

Appendix, contains quick references to the OrientDB embedded SQL-like parser and the native Java API interface as well as a list of the many configuration parameters available to control the server behavior. This appendix is available at `http://www.packtpub.com/sites/default/files/downloads/9956OS_Appendix.pdf`.

To complete the book, a convenient list of links and references to useful online sources about OrientDB is provided as well. This list is available at `http://www.packtpub.com/sites/default/files/downloads/9956OS_OrientDB_1.5.0.pdf`.

What you need for this book

To run OrientDB, it is necessary that you have a system capable to run at least Java SE 6.

To build from source code, it is necessary to have at least JDK 6, the Ant tool Version 1.6.5 or above, and Maven.

To get the source code, you can use Git, but this is not mandatory since the entire repository hosted by GitHub may be downloaded as a ZIP file.

Who this book is for

This book is great for database designers, developers, and systems engineers.

It is assumed that you are familiar with NoSQL concepts and have some experience with Java already.

For some topics like clustering, it's assumed that you have a basic knowledge of networking principles.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "Download the `orientdb-1.3.0.tar.gz` file from the OrientDB site and extract its content in a directory on your system."

A block of code is set as follows:

```
mail.send({
  profile: "default",
  to: "admin1@example.com,admin2@example.com",
  cc: "supervisor@example.com",
  subject: "Something happend!",
  message : "Alert! Something happend on OrientDB server!"
});
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
ODocument doc = new ODocument();
for(int i=0; i< 9999999; i++){
  doc.reset(); //here you will reset the ODocument instance
  doc.setClassname("Author");
  doc.field("id", i);
  doc.field("name", "John");
  doc.save();
}
```

Any command-line input or output is written as follows:

```
ORIENTDB_DIR="YOUR_ORIENTDB_INSTALLATION_PATH"
ORIENTDB_USER="USER_YOU_WANT_ORIENTDB_RUN_WITH"
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Clicking on the **Next** button moves you to the next screen."

> Warnings or important notes appear in a box like this.

> Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Installing OrientDB

There are several editions of **OrientDB Server**, each of them meets some criteria, and you have to choose which one you need. Keep in mind, however, that the data files are compatible through the server versions, so you can switch from one version to another anytime.

The three available versions are:

- OrientDB Standard Edition
- OrientDB GraphEd Edition
- OrientDB Enterprise Edition

There was another version which was focused on the key/value pair's management. It was called **OrientKV Server** but now it has deprecated and is no longer supported. In fact, the other versions of OrientDB have the capability to manage the key/value indexes as well, so if you need to manage large associative arrays or very big hash tables, you can use them to do that.

Standard Edition

The Standard Edition is the "main" edition of OrientDB; the other ones are extensions of the Standard Edition and add some specific capabilities to it.

The Standard Edition is shipped with a rich set of out of the box features; all of them are immediately available after the server installation.

Some of these features are:

- Graph database support
- Document database support
- Object database support

- Dictionary indexes support
- A SQL-like query language
- Transaction support
- Hooks (a sort of trigger, but at the system level rather than class level)
- Custom SQL function (to write your own functions in Java to expand the SQL parser)
- Stored procedures
- Plugin support to add features to the server's core
- REST interface
- Java API library
- Clustering
- Command-line administration console
- Web-based administration console (OrientDB Studio)

Graphed Edition

The Graphed Edition adds the **TinkerPop Blueprints** interface and the **Gremlin query** language to the Standard Edition.

Note that the Graphed Edition does not add the graph capabilities to OrientDB.

Relations through edges and other graph features are first-class capabilities in OrientDB, and are present in the Standard Edition too.

The Graphed Edition provides a layer on top of the OrientDB databases, so you can manage them through the Blueprints interface and provide the support to use the Gremlin language, if you want to use this de-facto query language for the graph databases instead of the OrientDB SQL-like language.

Key/Value Edition

The OrientDB Key Value server has not been supported since April 2011, and its source code was removed from the public repository in January 2013.

Its functionalities are available in the other versions.

In fact, each new OrientDB database has a special structure called **Dictionary**. The scope of the dictionary is to provide key/value management features to implement, for example, lookup tables, caches, and logs. You can also define as many dictionaries as you like, as per you needs.

Enterprise Edition

The Enterprise Edition will be released in 2013 and will add enterprise-class features to the Standard Edition, like an advanced monitoring cockpit, a collection of metrics related to the servers and clusters, Business Intelligence capabilities, and professional support.

Installing from the latest stable release

To install OrientDB you have to download its latest stable release, and you must have a Java virtual machine (Java SE 6 or above) installed on your target system.

Since the JVM is the only requirement to run an OrientDB server, this means it can be installed on any system supported by the Java platform, even the **Raspberry Pi**!

You can find the binary packages on the OrientDB official site: `www.orientdb.org`.

Currently, the 1.3.0 version is available and you can find two packages under the download section:

- `orientdb-1.3.0.tar.gz` which is a standard edition
- `orientdb-graphed-1.3.0.tar.gz` which is a graph edition

Installing the Standard Edition

Download the `orientdb-1.3.0.tar.gz` file from the OrientDB site, and extract its content to a directory on your system.

Now you should have a directory tree similar to the structure shown as follows:

/orientdb-1.3.0

/benchmarks

/bin

/config

/databases

/lib

/log

```
/www
 history.txt
 license.txt
 readme.txt
```

The following is an explanation for the preceding directory tree structure:

- `benchmarks`: scripts to perform several benchmark tests
- `bin`: scripts to run and to stop the server and the command-line console
- `config`: XML configuration files
- `database`: default path for the database files
- `lib`: the JAR files
- `log`: default path for the server logfiles
- `www`: the OrientDB Studio web application
- `history.txt`: the change logfile
- `license.txt`: the Apache 2 license
- `readme.txt`: instruction to build OrientDB from the source code

To run the server, just go into the `bin` directory, and launch the `server.bat` (on Windows OS) or `server.sh` (on Unix/Linux systems).

If you are using a Unix-based system, you may have to set the execution permission to the script files:

```
chmod +x ./bin/*.sh
```

If you plan to run OrientDB outside the `bin` directory, you have to set the `bin` path in the `PATH` environment variable, furthermore you have to set the `ORIENTDB_HOME` environment variable to the path's directory in which you extracted the `tar.gz` package.

The `ORIENTDB_HOME` variable is used by the scripts to guess the OrientDB position.

Now you can launch the server.

In the `bin` directory there are several scripts, for both Windows and Linux:

- `aserver.*`: just ignore them, they are experiments for the next releases
- `console.*`: run the command-line console
- `dserver.*`: run the server in distributed mode

- `orientdb.sh`: script to run/stop OrientDB as a daemon on Unix-like systems
- `server.*`: run the server
- `shutdown.*`: shutdown the server in a clean way (that is, do not kill it)

> Type `server.bat` (on Windows) or `./server.sh` (on Unix/Linux).

You should have an output similar to this:

```
2013-04-19 09:20:21:600 INFO OrientDB Server v1.3.0 (build @BUILD@) is
starting up... [OServer]
2013-04-19 09:20:22:936 INFO -> Loaded memory database 'temp' [OServer]
2013-04-19 09:20:23:148 INFO Listening binary connections on 0.0.0.0:2424
[OServerNetworkListener]
2013-04-19 09:20:23:151 INFO Listening http connections on 0.0.0.0:2480
[OServerNetworkListener]
2013-04-19 09:20:23:181 INFO Installing GREMLIN language v.2.2.0-SNAPSHOT
[OGraphServerHandler]
2013-04-19 09:20:23:195 INFO OrientDB Server v1.3.0 is active. [OServer]
```

Now you can run the console to connect to the server and try some simple commands.

In another terminal window, go to the `bin` directory and launch the `console` script.

Type **?** and see all the available commands:

The first command you have to supply is the `connect` command, to connect to a server:

```
connect remote:127.0.0.1/demo admin admin
```

The server replies:

```
Connecting to database [remote:127.0.0.1/demo] with user 'admin' ...OK
orientdb>
```

You can try the simple command:

```
info
```

> OrientDB is shipped with a demo database, moreover, it starts an in-memory database called `temp`.

To exit from the console, type:

exit

Now to shutdown the server, go to the `bin` directory and launch the `shutdown` script.

Installing the Graphed Edition

The procedure is similar to that used for the Standard Edition.

Go to the download page `www.orientdb.org` and grab the `orientdb-graphed-1.3.0.tar.gz` file.

Once you have downloaded the file, unzip it in a convenient directory.

The tree structure is same as that of the Standard Edition. There are some differences in some of them:

- `bin`: there are the `gremlin.*` scripts to run the Gremlin scripts from command line, batches, and shells
- `databases`: there is the `Tinkerpop` demo database instead of the OrientDB one
- `lib`: in addition to the OrientDB libraries, there are also the libraries of the Tinkerpop Stack, and other necessary JARs

Again to run the server, just go into the `bin` directory, and launch the `server.bat` (on Windows OS) or `server.sh` (on Unix-like systems).

Remember to set the right execution permission to the script files:

chmod +x ./bin/*.sh

and to set the `ORIENTDB_HOME` environment variable.

> Check if another instance of OrientDB is running. If so, terminate it to avoid the port binding conflicts.

Once the server is started, take a look at the last two output rows. Here you can see if the Gremlin language support is loaded:

```
2013-04-19 09:20:23:181 INFO Installing GREMLIN language v.2.2.0-SNAPSHOT
[OGraphServerHandler]
2013-04-19 09:20:23:195 INFO OrientDB Server v1.3.0 is active. [OServer]
```

Now you can run the console tool from another terminal window. Also, in this case, you should see a notice indicating that the Gremlin language support is available.

In other terminal, go to the `bin` directory and launch the `console` command:

OrientDB console v.1.3.0 (build 16) www.orientechnologies.com

Type `help` to display all the commands supported.

Installing extensions for GREMLIN language v.2.2.0-SNAPSHOT

orientdb>

OrientDB Graphed Edition is shipped with the `Tinkerpop` demo database.

Try to connect to the db:

connect remote:127.0.0.1/tinkerpop admin admin

And then try to execute the `info` command:

info

To exit from the console type:

exit

Compiling from the latest source snapshot

The source code is available through the **GitHub** platform at the URL `http://github.com/nuvolabase/orientdb`.

Since the build process will put the generated file in a directory called `release`, which will be a sibling of the `source` directory, I suggest you create a directory just for OrientDB. Inside that directory create a new directory to place the downloaded source code.

You can download the latest source code by selecting the master branch and downloading the `master.zip` file using the **ZIP** button.

Otherwise, you can clone the GIT repository on your machine using `git`. Type:

git clone git://github.com/nuvolabase/orientdb.git

To build OrientDB from source code you must have a JDK SE 6 or above, and the Ant Tool version 1.6.5 or above.

You can download Ant from `http://ant.apache.org/`.

Please note that the `ant` executable directory must be included in the `PATH` environment variable.

Once you have downloaded the source code and installed the necessary tool, you can proceed to build OrientDB.

Go into the source code directory and type:

`ant clean install`

You can also launch the `build.bat` (or `build.sh` in Unix-like system), however these scripts call `ant`.

To build the Graphed Edition too, type:

`ant clean installg`

At the end of the build process you should have a screen shown as follows:

```
installg:

init:
     [echo] ------------------------------------
     [echo] -> orientdb-graphdb
     [echo] ------------------------------------

compile:
    [javac] Compiling 32 source files to C:\OrientDB\sorgenti\github\orientdb-m
    [javac] Note: Some input files use unchecked or unsafe operations.
    [javac] Note: Recompile with -Xlint:unchecked for details.
     [copy] Copying 2 files to C:\OrientDB\sorgenti\github\orientdb-master\orie

jar:
      [jar] Building jar: C:\OrientDB\sorgenti\github\orientdb-master\orientdb\
     [copy] Copying 2 files to C:\OrientDB\sorgenti\github\orientdb-master\orie

install:
     [copy] Copying 3 files to C:\OrientDB\sorgenti\github\orientdb-master\rele
     [copy] Copying 184 files to C:\OrientDB\sorgenti\github\orientdb-master\re
     [copy] Copying 6 files to C:\OrientDB\sorgenti\github\orientdb-master\rele
     [copy] Copying 2 files to C:\OrientDB\sorgenti\github\orientdb-master\rele

BUILD SUCCESSFUL
Total time: 1 minute 7 seconds
```

The generated files are in the `release\orientdb-x.y.z-SNAPSHOT` and `release\orientdb-graphed-x.y.z-SNAPSHOT` directories, where `x.y.z` is the current version under development.

Running the test suite

To run the test suite against a freshly-built OrientDB snapshot, you can type the following command:

```
ant test
```

After the test suite has finished (you should have 0 errors), in the database directory of the Standard Edition, you should have some databases created to run the tests, including the demo database.

There are another set of tests that need of Maven to be executed.

So, if you want to execute these tests you must download Maven from `http://maven.apache.org/download.cgi`

and install it following the instruction provided in the `readme.txt` file within the installation package.

Once you have Maven on your system, you can type:

```
mvn clean test
```

Installing as a daemon/service

OrientDB can run as a background process. The setup process depends on the server platform.

Linux systems

OrientDB is shipped with a script that can be used to run OrientDB like a daemon.

It supports the start/stop/status parameters and can be configured to execute OrientDB with a specified user's credentials.

The script is `orientdb.sh` and is located in the `/bin` directory of the installation path.

You must open it and change the first lines:

```
ORIENTDB_DIR="YOUR_ORIENTDB_INSTALLATION_PATH"
```

```
ORIENTDB_USER="USER_YOU_WANT_ORIENTDB_RUN_WITH"
```

Set the installation path and the user as stated, save the script, and deploy like other scripts for the other daemon.

Different Linux distribution uses different ways to manage the start/stop process at the system bootstrap/shutdown.

Windows systems

Since OrientDB is a Java application, it does not offer any native way to run as a Windows service.

However, there are some tools that can be wrapped with any executable so that it can be installed and managed as a Windows service.

The correct procedure to do this is illustrated at the wiki page `https://github.com/nuvolabase/orientdb/wiki/Wrapping-As-A-Windows-Service`. The steps are as follows:

1. You must download the **Apache Commons Daemon** tools for your system. The latest version is available at: `http://www.apache.org/dist/commons/daemon/binaries/windows/`.

2. Download and unzip it.

3. You will find a `prunsrv.exe` file in the root directory. This is the file for x86/32 bit systems. In the directory `amd64` there is a version of the `prunsrv.exe` file for x86/64 architectures, while the directory `ia64` contains a version of the `prunsrv.exe` file for the Itanium machines.

4. Let's say you have installed OrientDB in the `%ORIENTDB_HOME%` directory, you have to create a new directory called `%ORIENTDB_HOME%/service`.

5. Copy to this new directory the two `.exe` files shipped with the Apache Commons Daemon tools: `prunmgr.exe` and `prunsrv.exe`, according to the architecture of your machine.

6. Rename `prunsrv.exe` to `OrientDB.exe`, and `prunmgr.exe` to `OrientDBw.exe`.

7. Copy to the same directory the file `installService.bat`.

8. In order to execute this script you must locate the `jvm.dll` file installed on your system. Generally this file is in the `%JAVA_HOME%\jre\bin\server` directory.

9. Assuming that Java is installed in the `C:\Program Files\Java\jdk1.6.0_37` path, and that OrientDB is installed in the `C:\OrientDB\releases\orientdb-1.4.0-SNAPSHOT` directory, you must type the following command:

   ```
   installService.bat "C:\Program Files\Java\jdk1.6.0_37\jre\bin\
   server\jvm.dll" C:\OrientDB\releases\orientdb-1.4.0-SNAPSHOT
   ```

10. Now you can open the **Windows Services Management Console** and see OrientDB listed as a service.

Summary

In this chapter, we had our first encounter with OrientDB. We have seen the available versions, have learned to build it from the latest available source code, and have run a server instance.

We have also seen how to deploy it as a Windows service.

In the next chapter, we will go into more detail about the OrientDB architecture. We will explore the first basic concepts, such as how the data are organized and stored on filesystem, the security framework, and how to perform administrative tasks using the embedded tools.

2
Administering OrientDB

Before starting to administer a server, you have to know how it is structured and how each component fits into the big picture.

The following diagram shows the OrientDB general architecture:
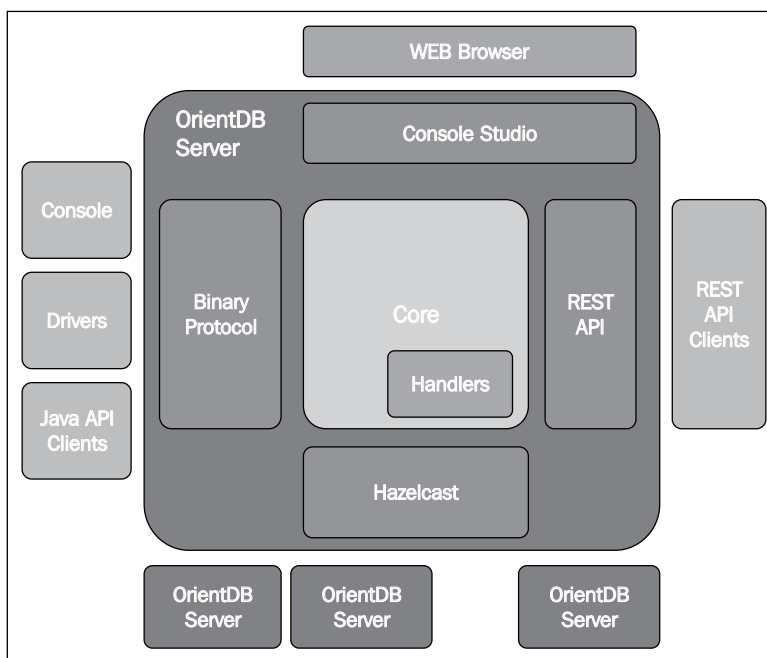
Each building block performs a specific task, which are as follows:

- **Core**: This is the core of the system. It is responsible to store, manage, and access the data. It has an embedded SQL-like parser and it can be extended through the handlers. Each server can manage many different databases.

- **Handlers**: These are the custom modules (also known as handlers or plugins) developed by Orient Technologies or other third-party developers. The plugins extend the main functionality of the core module. Some out of the box some plugins are provided, for example, the automatic backup plugin and the send email plugin.

- **REST API**: This block provides a REST interface to perform queries and other tasks against OrientDB. In this way you can send and receive data via a standard HTTP client and use JSON to represent data.

- **Binary Protocol**: This is the main interface and the best way to communicate with the server. It uses proprietary protocols to achieve the best results in terms of bandwidth consumption and speed. Of course, the clients must be able to manage these protocols.

- **Hazelcast**: OrientDB uses the Hazelcast technology as a transport layer to organize communication between nodes in a cluster. In fact OrientDB can also be deployed in a distributed, multi-master cluster architecture.

- **OrientDB Console Studio**: This is a web application embedded into the server, which allows administering OrientDB through a web browser. OrientDB Studio executes REST calls to perform its tasks.

- **Console**: This is an external tool provided with the OrientDB package. It allows us to connect to an OrientDB server, or even to connect to an OrientDB data file on a local filesystem and perform administrative tasks, queries, and other actions. It can also be used in batch scripts.

- **Java API clients**: They provide a set of JAR files to embed into the client applications. Their main intention is to abstract the underlying database and the network protocols to communicate with the server, exposing a coherent OOP interface. They use the binary protocol to communicate with the server.

- **Binary Drivers**: They are generally provided by the third-party developers. They include JDBC drivers and other non-Java language drivers.

Configuration

You can configure OrientDB by editing the files located in the `ORIENTDB_HOME/config` directory. In this directory you will find the following files:

- `default-distributed-db-config.json`
- `hazelcast.xml`
- `orientdb-dserver-config.xml`
- `orientdb-server-config.xml`
- `orientdb-server-log.properties`

The main configuration files are `orientdb-server-config.xml` and `orientdb-dserver-config.xml`, which are used to configure a standalone server and a cluster node, respectively. Please note that when OrientDB starts for the first time, it rewrites the config files, deleting from them all the comments! So I suggest you keep a backup of these files for future reference. Let's see their content.

orientdb-server-config.xml

The `orientdb-server-config.xml` file has the following sections:

- **handlers**: This section defines the handlers (plugins) and theirs parameters.
- **network**: This section defines which protocols are active, which ports are used, and other useful parameters for the HTTP protocol.
- **storages**: This section defines the default in-memory database and other databases located outside of the default path. For example, you may have a database located in a specific folder. You could define a new storage as follows:

```
<storage name="myNewDb" path="local:C:/OrientDB/alternative/
myNewDb"
          userName="admin" userPassword="admin"
          loaded-at-startup="true"
/>
```

  - `userName` and `userPassword` are optional, if not provided the default pair admin/admin is used. If the database is not found at the specified location, a new one is created.
  - `path` is the location of the database. Please note that just before the directory location there is the storage type keyword. This could be `local` or `memory`. The first one indicates that the database is persisted on the filesystem, the second one indicates that the database must be kept in memory, and consequently any data will be lost when the server is stopped.

- ° `name` of the database is case sensitive.
- ° `loaded-at-startup` indicates whether the database has to be mounted and opened at the startup or at the first connection request.

- **users**: This section defines the default users of the server.
- **properties**: This section can be used to set some OrientDB properties.

The users section is empty. But when OrientDB starts for the first time, it creates two default users: root and guest. The root user will have full access to all databases, status information, statistics, and other useful information. It is the most powerful user and because of this OrientDB generates a very long random password and puts it in the configuration file.

To access with the root credentials you must know the password placed in the XML configuration file. OrientDB also generates the guest user which can only perform a query to obtain the name of the available databases.

orientdb-dserver-config.xml

It is similar to the `orientdb-server-config.xml` file, but it has an additional handler to configure the Hazelcast plugin parameters.

- `default-distributed-db-config.json` and `hazelcast.xml`: These are additional parameters to configure a distributed server
- `orientdb-server-log.properties`: This is an additional configuration for logging

The OrientDB console

The first approach to OrientDB after starting it, is of course through its console. To start the console, if you followed the installation steps in *Chapter 1*, *Installing OrientDB*, all you have to do is just type as follows:

console

This is because you have to set both the `ORIENTDB_HOME` and the `PATH` shell variables, if you are using a *nix system, to set the execution flag of the `*.sh` files in the `ORIENTDB_HOME/bin` path.

Now you can connect to an instance of a running OrientDB server.

If you started the server, you should connect to it by typing (inside the console tool):

connect remote:localhost/demo admin admin

The `connect` command follows the given pattern:

```
connect remote:<host>[:<port>] root <root-password>
```

> **Downloading the example code**
>
> You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

The port is optional, if not specified the default value of `2480` is used. Using this command, you can connect to a remote server with root credentials. You don't need to specify any database. You can find the root password is in the XML configuration file.

Of course, you can't execute any query against a specific database, since you have not given a database name to connect to. The root username is required in many server scope administrative tasks.

> Note that the default configuration files initially don't have any entry related to the root user. OrientDB will create it as soon it starts.

You can connect to a specific database providing the following command:

```
connect remote:<host>[:<port>]/<database> <user> <password>
```

This second command is used to connect to a remote database. You must provide a valid username and password. Once connected to a database you can execute queries against it.

Finally, there is a third way to connect to a database, which is as follows:

```
connect local:<path> <user> <password>
```

This third command is used to interact with a database without using a remote server (note the use of the `local` keyword instead of the `remote` one). In this case the console tool itself acts like a local server. You cannot use the root credentials because actually you're not using any server, you are just "mounting" the data files into the console. The `console` command is also used to execute commands against a server within a batch file. In this case you can include the following line into your script:

```
console "connect remote:localhost/demo admin admin;select * from profile"
```

Please note that the commands are separated by a semicolon. Of course, you can put all the commands to execute in a separated text file:

```
console commands.txt
```

OrientDB data files

Each database is stored by default in a subdirectory in the `<ORIENTDB_HOME>/ databases` path. Since Version 1.3, it is possible to customize the default database path to set the value in the `orientdb-server-config.xml` section properties:

```
        <properties>
                <entry name="server.database.path" value="/var/
orientdb/databases" />
        </properties>
```

Before inspecting a database file, let's introduce the OrientDB cluster definition. In OrientDB, clusters are the main data structures used to organize data. In the relational world there is no concept similar to the OrientDB cluster. You may think a cluster as a way to group records. Clusters can have no schema and no columns, their records are called documents, and each document can be different from the others that belong to the same cluster. There are two kinds of clusters: physical and in-memory. The first ones are persistent, the second one are volatile and are destroyed when the server is shut down. Clusters are not only important from a conceptual point of view, but also from the point of view of the physical organization of files. In fact for each physical cluster, OrientDB creates one or more files. Generally each database consists of at least the following files:

- `database.ocf`: It is an internal dictionary file. It contains the physical location and names of other data files. You can move a database in a different path using the internal import/export tools provided by OrientDB, but not because of this.

- `default.0.ocl`: OCL files contain the pointers to the documents contained in the ODA files. Each cluster may have one or more OCL files. This is the first file of pointers for the default cluster.

- `default.0.oda`: ODA files store the content in the records. Each oda file can store data belonging to different clusters. Despite the name, this file does not have content only for the default cluster, and also for the other ones.

- `default.och`: OCH files (one per cluster) contain the "holes" generated in the OCL files by the `delete` and `update` commands. OrientDB will use these holes to reallocate the space for new records.

- `default.odh`: ODH (one per ODA file) files contain holes generated in ODA files by the `delete` commands.

- `index.0.ocl`: The `index.*` files are used to manage the indices created at schema level.

- `index.och`

- `internal.0.ocl`

- `internal.och`
- `manindex.0.ocl`: The `manindex.*` files are used to manage the so called manual indices, that is, indices not related to the schema, for example, dictionaries.
- `manindex.och`
- `ofunction.0.ocl`: The `ofunction.*` files are used to store and manage server-side custom functions and code.
- `ofunction.och`
- `orids.0.ocl`: The ORIDS files are used to manage the Record IDs (RIDs). An RID is the unique record identifier.
- `orids.och`
- `orole.0.ocl`: The `orole.*` files contain information about users' roles and permissions.
- `orole.och`
- `ouser.0.ocl`: The `ouser.*` files contain information about database's users.
- `ouser.och`
- `txlog.otx`: This file, one per database, is used to keep track of the active transactions.

Classes

Classes are concepts at a higher layer of abstraction of clusters. The definition of "class" in the OrientDB world comes directly from the OOP paradigm. However, you can imagine a class similar to a table in the relational world: a structure to store records, they can have typed properties, for example, the tables' columns, but the similarities end here. In fact, classes could be of the following types:

- schema-full (that is, having strong typed mandatory properties)
- schema-less (that is, no defined properties)
- mixed-mode (that is, some properties could be defined and strong typed, others without a type, or even not defined)

In the OrientDB world, the word documents, records, and objects are all synonyms and all of them indicate a generic entity that can be stored. When a class is defined, OrientDB creates a corresponding cluster and a pair of OCL-OCH files. As said, the OCH file contains the pointers to the records stored in the data segment files (ODA files). The OCH file contains the "holes" present in the OCL file due to the `delete` and `update` commands. However, records of a class could be stored in different clusters, if you want to do so. This is the case where you have a large dataset of records that could be convenient to partition. For example, the billing information broken down by year of issue.

Just to try a class-related command, let's enter into the console and connect to the demo database:

connect remote:localhost/demo admin admin

Now let's create a new class:

```
create class newclass
```

Now let's inspect what happened to the database:

```
info
```

OrientDB replies with a lot of database information. The first ones are related to the defined cluster. As you can see, there is a new cluster called, like our new class, `newclass`. In the report you can see the cluster name (equal to the class name), its unique ID, the cluster type (physical/in-memory), the number of records stored in, and the file size (not available through remote connection). If you continue to inspect the report provided by the `info` command, you will find the list of defined classes. Our new class is listed and the cluster ID is shown in the second column. You can obtain synthetic information about our new class as follows:

```
desc newclass
```

OrientDB associates a unique ID to each cluster. This is a very important information because all the records have a unique ID called RID, which is of the following form:

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

RIDs are physical pointers to the records. This means that the operation of traversal and links following among documents are very fast, because there are no lookup overheads when the database engine has to access them. Note that, this maybe the greatest feature of OrientDB as compared to relational databases. Knowing the physical position of the records allows OrientDB to reach them without performing any index lookup. RDBMS, if an index is defined against a table's column, has to do an index lookup to know where the record is stored. This operation generally is executed $O(log\ n)$ times, whereas OrientDB performs in $O(1)$ time.

Abstract classes

Like in the OOP paradigm, there is also the concept of abstract class in OrientDB. The abstract classes are not associated to any cluster (their cluster ID is -1), and cannot store any records, but are useful to define properties which every derived classes will have. An abstract class is created as follows:

```
create class MyAbstractClass abstract
```

To see information related to the new abstract class, enter the following code:

```
desc MyAbstractClass
```

OrientDB will show `MyAbstractClass` details. Note that the cluster ID is -1.

Let's define some properties:

create property MyAbstractClass.name string

Property created successfully with id=1

create property MyAbstractClass.birthDate datetime

Property created successfully with id=2

Now retype:

```
desc MyAbstractClass
```

As you can see now, OrientDB also shows the information related to the properties.

Now, if we define a new class which extends our abstract class, it will inherit all the properties:

```
create class MyConcreteClass extends MyAbstractClass
Class created successfully. Total classes in database now: 95
desc MyConcreteClass
```

Class................: MyConcreteClass

Super class..........: MyAbstractClass

Default cluster......: myconcreteclass (id=98)

Supported cluster ids: [98]

Properties:

---------+--------+----------------+---------+--------+--------+-----+--
---+
NAME | TYPE |LINKED TYPE/CLASS|MANDATORY|READONLY|NOT NULL| MIN |
MAX |
---------+--------+----------------+---------+--------+--------+-----+--
---+

```
birthDate|DATETIME|null               |false     |false  |false    |      |
|

name      |STRING  |null               |false     |false  |false    |      |
|

--------+-------+--------- -------+---------+------+--------+-----+--
---+
```

Keep in mind that the class name is case insensitive, whereas the properties are
case sensitive.

Security

Security is a very important topic for every software platform. OrientDB has a very
robust and well-known security model based on users, roles, and rules. Each database
has users, each user can have one or more roles, and finally each role has rules.

Furthermore, an OrientDB server could have one or more root users.

Rules

Rules define security criteria against specific resources. A rule consists of a resource
and an operation to be applied to the resource itself. The allowed operations are
as follows:

- **NONE**: 0 bitmask #0000
- **CREATE**: 1 bitmask #0001

- **READ**: 2 bitmask #0010
- **UPDATE**: 4 bitmask #0100
- **DELETE**: 8 bitmask #1000
- **ALL**: 15 bitmask #1111

Since the preceding operations are represented internally by a mask of bits, their combination can also be done. For example, if you would give to a role a permission to read, update, and delete a resource, you should use the code 2 + 4 + 8 = 14 (`bitmask #1110`). The allowed resources are as follows:

- `database`
- `database.class.*`
- `database.class.<class-name>`
- `database.cluster.*`
- `database.cluster.<cluster-name>`
- `database.command`
- `database.config`
- `database.hook.record`
- `database.bypassRestricted`
- `server.admin`

Roles

OrientDB uses the roles to know if an operation is allowed to a specific user. Each role has one or more rules associated to it. The roles are stored into the `ORole` class. Let's start to explore how OrientDB manages the security. Start the console and connect to the demo database as the admin user as you did before:

```
connect remote:localhost/demo admin admin
```

And then type:

```
desc orole
```

There are four properties defined as follows:

- `name`: This property defines the name of the role.
- `mode`: This property defines how the rules should be interpreted. It can be 0 or 1.

  0 means "deny all but"… the rules. By default, all the operations are denied against all the resources, except for rules explicitly declared.

  1 means "allow all but"… the rules. By default, all the operations are allowed against any resource, except for rules explicitly declared.

- `rules`: This property is a map of key/value pairs. The keys are the resources and the values are the operations allowed/denied (depends on the mode value) against the resources.
- `inheritedRole`: This property defines the roles that can extend other roles.

  Now type:

  ```
  select from orole
  ```

  You should see the default roles. The default ORole's records should be as follows:

  - **admin**: It is the role of database administrators. A user that belongs to this role can do everything on any structure of the database.
  - **writer**: The users belonging to this role can read, create, update, and delete records. They can't access internal information.
  - **reader**: The users belonging to this role can read any record of any class.

To create a new role you have to provide a command as follows:

```
insert into orole (name,mode,rules) values ("mynewrole",1,{"database.
class.Person":0})
```

This means: create a new role with the name `mynewrole`, it will have access all resources but the `Person` class.

Users

Users are stored in the `OUser` class, except for those defined in the XML config file (by default root and guest). Each user must have a password and at least one role. The roles associated with the user, define the rights it has against the database. By default, every new database has the following three users:

- **admin**: It's is the database administrator, password is admin. It can do everything on any structure of the database.
- **writer**: It can read, create, update, and delete records and its password is writer. It can't access internal information.
- **reader**: Its password is reader and can only read any record.

Again, connect via console to the demo database as admin, and type:

```
select from ouser
```

You will see the default users.

Now type:

```
desc ouser
```

Class................: OUser

Super class..........: OIdentity

Default cluster......: ouser (id=5)

Supported cluster ids: [5]

Properties:

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

Create a new user as follows:

```
insert into ouser (name,password,status,roles) values ("mynewuser","my
password","ACTIVE",(select from orole where name="mynewrole"))
```

Now, disconnect from the console:

```
disconnect
```

And reconnect using the new `mynewuser` user. Since the `mynewuser` user belongs to the `mynewrole` role, which has a rule that does not permit access to the `Person` class, you will be able to perform any operation but not access the `Person` class.

Users can be temporarily disabled by acting on the `status` property.

It can assume two values: `ACTIVE` or `SUSPENDED`

To disable a user, just update the `status` field:

```
update ouser set status="SUSPENDED" where name = "reader"
```

In this case, the `reader` user cannot connect to the server until its `status` reverts to `ACTIVE`. Passwords are stored in the SHA-256 hash format. To change a user's password, you don't need to calculate the hash code by yourself. This is done transparently by the server:

```
update ouser set password="newpass" where name="reader"
```

Server users

The server users are special users defined in the XML config file. They are not counted in the databases' `OUser` classes because they act at server level and are not related to a single dtabase. The server users are defined in the users section of the `orientdb-dserver-config.xml` and `orientdb-server-config.xml` file. When OrientDB starts for the first time, it creates two entries: one for the `root` user and one for the `guest` user. Root can do anything and access any resource; guest can only connect and obtain the list of current databases. You can of course declare how many users as you want, but I suggest to leave the default ones so that OrientDB can use them. For example, OrientDB Console Studio uses the guest user to list the available databases on the login screen.

Available resources are as follows:

- `server.info`: This resource retrieves information and statistics about the server
- `server.listDatabases`: This resource retrieves the list of available databases
- `server.dbList`: This resource retrieves the previous one, but used when the binary protocol is used (that is, the command-line console)
- `database.create`: This resource creates a new database
- `database.drop`: This resource can drop an existing database
- `database.passthrough`: This resource can access all the databases, even if the user is not present in the OUser classes
- `*`: This resource can access all the previous resources

Record-level security

Record level security is one of the most powerful features of OrientDB. Using this functionality, developers can apply a fine access control and security permissions to any single record of a class. By default all users that have the permission to access the records of a class can act on them, reading, updating and deleting (despite of who created them). By activating this functionality on a class, each record will have additional fields that will indicate to the db engine whether a user could access it and with what privileges. To have the record security level, a class must extend the `ORestricted` abstract class. Enter into the console and connect to the demo database as the admin user and type as follows:

```
desc ORestricted

Class................: ORestricted

Default cluster......: null (id=-1)

Supported cluster ids: [-1]

Base classes.........: CMSDocument

Properties:

---------------+---------+------------------+----------+----------+----
------+-----+-----+
 NAME          | TYPE    | LINKED TYPE/CLASS | MANDATORY | READONLY | NOT
NULL | MIN | MAX |
---------------+---------+------------------+----------+----------+----
------+-----+-----+
 _allowDelete  | LINKSET | OIdentity         | false     | false    |
false     |     |     |
```

```
 _allowUpdate  | LINKSET | OIdentity          | false     | false     |
false      |       |       |
 _allowRead    | LINKSET | OIdentity          | false     | false     |
false      |       |       |
 _allow        | LINKSET | OIdentity          | false     | false     |
false      |       |       |
---------------+---------+-------------------+-----------+-----------+----
------+-----+-----+
```

You can see that there are four properties defined, each one is a LINKSET (that is a set of links to other documents) of RID belonging to the OIdentity class:

```
desc OIdentity
Class.................: OIdentity
Default cluster......: null (id=-1)
Supported cluster ids: [-1]
Base classes.........: ORole, OUser
```

The OIdentity class is an internal abstract class extended by the ORole and OUser classes.

This means that the LINKSET fields of ORestricted may contain link to users or roles or both.

This is a very important concept, since it means that you could allow a specific user to act on a record or even to all users belonging to a role.

In the demo database there is already a class that extends ORestricted: CMSDocument.

In fact if you type desc cmsdocument you will see that it extends ORestricted and owns all the fields of ORestricted'.

What happens under the hood is that OrientDB injects a hook when the following operations are executed against the class that extends ORestricted:

- Create: This command is executed when a new record is created, the current db user is put in the _allow field. It will have full access to the record.
- Read: This command is executed when a document is read, the engine checks if the current user or one of its roles is listed in the _allow or _allowRead fields. If not, the record is skipped.

- `Update`: This command is executed when a document is updated, a check against the `_allow` and `_allowUpdate` is performed.
- `Delete`: This command is executed when a document is deleted, a check against the `_allow` and `_allowDelete` is performed.

It is possible to change the default behavior of OrientDB.

By using the custom properties `onCreate.fields` and `onCreate.identityType` you can instruct OrientDB about how it must operate when a new record is created:

- `onCreate.fields`: This property allows you to specify the name of the fields to set when a record is created. For example, by default it is `_allow`, but you can specify any other allowed field, that is, `_allowRead`, `_allowUpdate`, `_allowDelete`, or even their combination.
- `onCreate.identityType`: This property allows you to indicate if the user's RID will be put in the `_allow` field or its role. Since a user could have multiple roles, only the first one will be inserted.

For example:

```
alter class cmsdocument custom onCreate.identityType=role;
alter class cmsdocument custom onCreate.fields=_allowRead,_
allowUpdate;
```

The OrientDB Studio

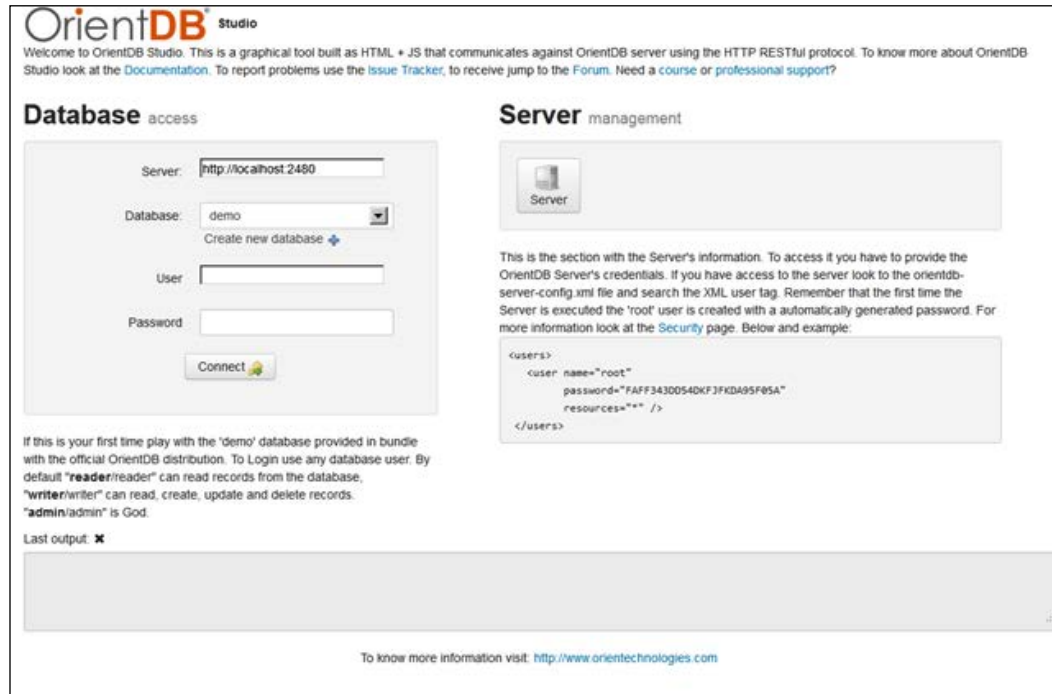OrientDB comes with a powerful web-based GUI console.

It is embedded in the server. So, to access to it, all you have to do is to start your browser and open the following link:

```
http://<OrientDB-host>:2480
```

If you are running the server on localhost you can access through the following link:

```
http://localhost:2480
```

The port 2480 is defined in the XML config file and can be changed.

As you can see, you can choose an existing database or even create a new one. You could also log in as root and access a special section for the server management. Select the demo database and connect as admin.

After you log in you can see a toolbar in the top of the screen. Through the toolbar placed on the top of the screen, you can change section and access the Studio features. The first one is the **Database** section, which is the default one.

The Database section

The **Database** section looks as follows:
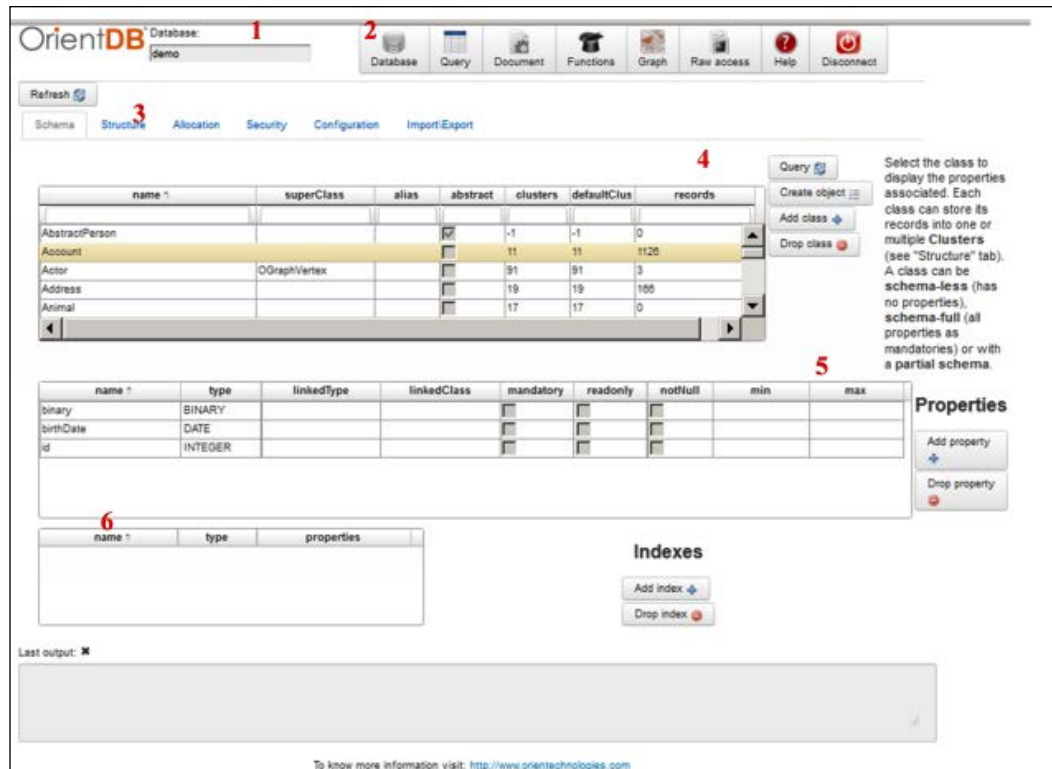
As you can see in the preceding screenshot, the screen is split into the following subsections:

- The name of the current database
- The toolbar
- The tabs to access to the subsections of the sections selected by clicking on a toolbar button
- The list of the defined classes of the current database
- The defined properties of the selected class in the section 4
- The defined indexes of the selected class in the section 4

The tabs allow to access and work on various aspects of database management.

In the first one you can see and modify the schema, define a new class, drop or edit an existing one.

The second one, **Structure**, shows the physical structure of the database, listing all the files, their position into the filesystem. You can add or drop in clusters.

Under **Security** are listed the users and roles.

In the **Configuration** tab are listed all the defined properties, for example, locale, time zone, and so on.

The Query section

In the **Query** section you can execute any SQL or Gremlin query (if you are running the Graphed version). In the bottom part of the screen you will see the reply from the server. Note that you cannot perform schema-related command, for example, `create class`.

The Document section

In the Document section, you can see and modify the content of a given document. You have to know its RID, or you can click directly on an RID in the result pane of the **Query** section. Unfortunately if a field contains an embedded object, its content cannot be edited.

The Functions section

In the **Functions** section, you can see, create, and edit the server functions. They are a sort of stored procedures written in JavaScript. This is explained later in this book.

The Graph section

The **Graph** section is perhaps the most curious tool of the Console Studio. You have to know an RID, or from the **Document** section click on the **Graph** button. In this section, you may see all the connections (links and graph edges) that a specified document has with other documents. To try this on the screen, come back to the first query and type the following query:

```
select from actor
```

Now click on an RID, and then on the **Graph** button. You will see the node you selected and its connections. You can change and modify various graph parameters on the right of the graph.
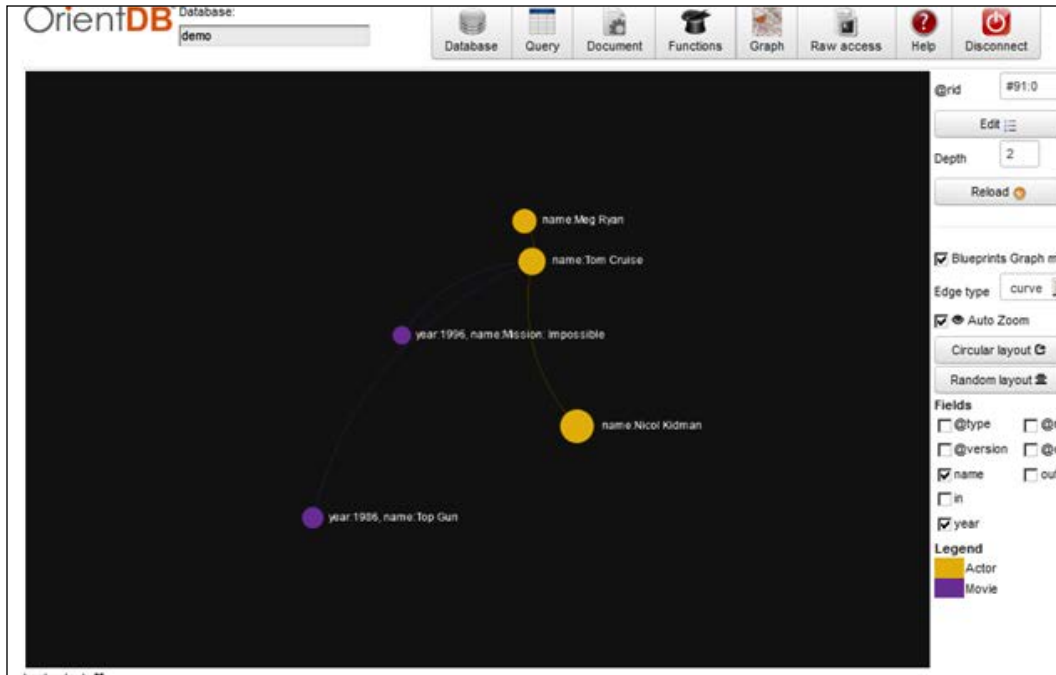

The Raw access section

This section allows us to query the server using the HTTP REST API. The server response is shown in the bottom panel. The HTTP REST protocol API is discussed later in this book.

The root user

A root user has access to a special screen of the OrientDB Studio.

Since the authentication with the server is done via basic auth, many browsers cache the credentials until the session with the server is closed. This often can be achieved by closing the browser and reopening it.

Once you have restarted the browser, open the Studio and click on the **Server** icon on the right of the home page:
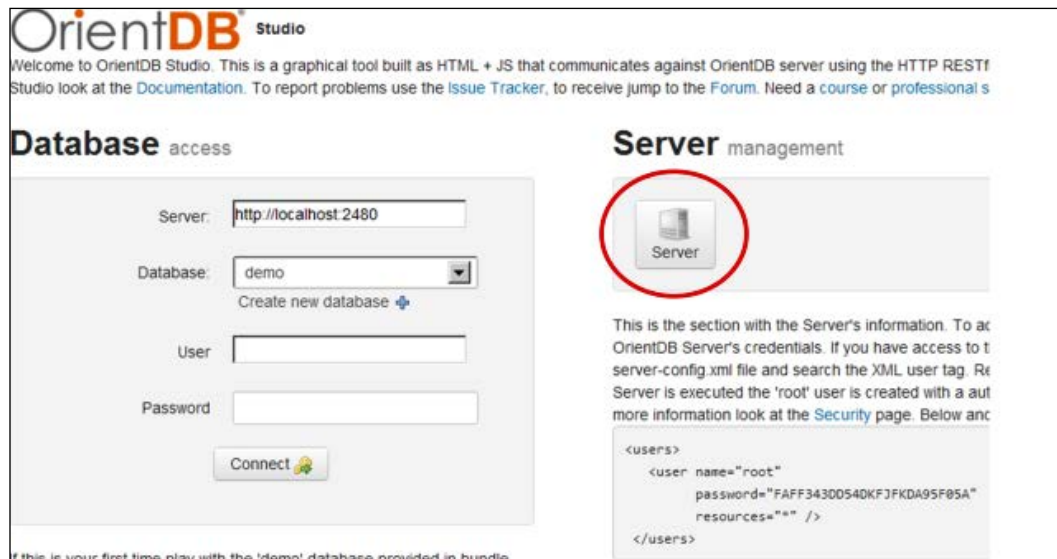
After this, click on the **Refresh** button. A popup should appear asking for the user credentials. You have to enter the root user credentials as defined in the XML config file.

The screen is split into the following tabs:

- **Active connections**: This tab shows the active connections and related properties
- **Configuration**: This tab shows the server-side properties and their values as defined in the properties section in the XML config file
- **Database pool**: This tab shows the status of the connection pool
- **Used Storages**: This tab shows the used storages

These tabs are used to show the server status and health.

Back up / restore

OrientDB does not have any real backup and restore procedures, but rather they have the import/export ones. The export/import procedure is useful when you plan to upgrade your version of OrientDB. In fact, since the compatibility between different engine versions is not guaranteed, to move a database from an engine to a new one, it is recommended to do an export from the old system and an import into the new one. To import/export a database you can use both the command-line console and the OrientDB Studio.

Using the console

Using the console, you will be able to export an entire database or only its schema. To do export a database fully, you have to connect as admin and execute the following command:

```
export database <output-file>
```

For example:

```
export database demo.json
```

The database is exported and gzipped. The compressed gzipped file contains a text file which has all the database info, schema, and data in JSON format. This file can be imported back into a new fresh database. This means that you have to create a new database and then you will be able to import the backup. To export only the database schema without the data, execute the following command:

```
export schema <output-file>
```

To import a previously exported database, you need to create a new database. For example, from the console type the following command:

```
create database remote:localhost/newdemo root <root-password> local
```

After the creation, you should automatically connect to the new database. If you type info, you should see a line as follows:

Current database: newdemo (url=remote:localhost/newdemo)

Total size: 4,09Kb

Now you can perform the import as follows:

```
import database <input-file>
```

For example:

```
import database demo.json.gz
```

> There is a well-known issue when you try to import a database from an old version of OrientDB into a newer version. You may have an error as follows:
>
> ```
> Imported cluster 'XXX' has id=X different from the
> original: Y
> ```
>
> To avoid this problem you must drop the ORids class before performing the import.

Using the OrientDB Studio

Of course, you can perform the export/import within the OrientDB Studio as well. Enter into the Studio and connect to the desired database with the admin credentials. Now you have to go into the database section and select the **Import/Export** tab. You will see a screen with two tabs.

In the first tab there are two buttons, one for the export procedure and one to import a previously generated export file. By clicking on the **Export Database** button a GZIP file is generated and downloaded. To import a backup you have to click on the **Import Database** button.

By default, a very small file can be uploaded (100 KB). You can change this parameter by modifying the XML config file. In the section properties create an entry as follows:

```
<entry name="network.http.maxLength" value="<max-length-in-bytes>" />
```

You have to restart the server after this change. Note that the import procedure accepts only JSON files. Therefore, you must unzip the exported file before uploading it.

Automatic backup

Since the export procedure could be run within the command-line console, you could place a script that performs the backup operation and schedule it via cron in the*nix system or via scheduled tasks in Windows. However, there is another way to achieve this result.

OrientDB is shipped with a special handler that provides the automatic backup functionality. In the XML config file there should be a section as follows:

```
<handler class="com.orientechnologies.orient.server.handler.
OAutomaticBackup">
   <parameters>
   <parameter name="enabled" value="false" />
  <!-- parameter name="firstTime" value="03:00:00" / -->
  <parameter name="delay" value="4h" />
  <parameter name="target.directory" value="backup" />
  <parameter name="target.fileName" value="${DBNAME}-
${DATE:yyyyMMddHHmmss}.json" /><!-- ${DBNAME} AND ${DATE:} VARIABLES ARE
SUPPORTED -->
  <parameter name="db.include" value="" /><!-- DEFAULT: NO ONE, THAT
MEANS ALL DATABASES. USE COMMA TO SEPARATE MULTIPLE DATABASE NAMES -->
  <parameter name="db.exclude" value="" /><!-- USE COMMA TO SEPARATE
MULTIPLE DATABASE NAMES -->
   </parameters>
</handler>
```

To enable the handler, change the value of the `enabled` property to `true`.

The complete list of the available parameters is as follows:

- `enabled`: This parameter activates or deactivates the handler (values are `true` or `false`).

- `firstTime`: This optional parameter is used to specify the time of the first backup. If not specified, the first backup will be done at the server startup time adding the `delay` parameter.

- `delay`: This mandatory parameter defines the delay time.

  You can use different suffixes to specify different measures:

  - `ms` for milliseconds: The handler will perform a backup every `ms` millisecond.

  - `s` for seconds: The handler will perform a backup every `s` second.

  - `m` for minutes: The handler will perform a backup every `m` minute.

  - `h` for hours: The handler will perform a backup every `h` hour.

  - `d` for days: The handler will perform a backup every `d` day. `1d` means every day.

- `target.directory`: This parameter defines the target backup directory, the default value is `"backup"`. If it does not exist, it will be created.

- `target.fileName`: This parameter defines the target backup filename. It can be configured using the following placeholders:

    ◦ `${DBNAME}`: This is used to configure the current database name

    ◦ `${DATE}`: This is used to configure the current date followed by the Java DateTime format

- `db.include`: This parameter defines the database comma-separated list to be included in the backup. (`Empty` means all databases)

- `db.exclude`: This parameter defines the database comma-separated list to be excluded from the backup. (`Empty` means none)

Once you have configured the handler, you must stop and restart the server. OrientDB will print the handler configuration to show that the plugin is active, up and running:

```
INFO Automatic backup plugin installed and active: delay=14400000ms,
firstTime=null, targetDirectory=backup/ [OAutomaticBackup]
INFO OrientDB Server v1.3.0 is active. [OServer]
```

Summary

In this chapter, we have seen how to administer an OrientDB server, how to organize its data structure, and how to use the tools distributed in the server.

In the next chapter, we will learn how to build a database, perform queries, and use the Java API to build programs that uses OrientDB-like database.

3
Programming OrientDB

The first thing you have to do when you have to design an OrientDB database is to choose the right kind of database you will use.

OrientDB could be used similar to a document database, an object database, or a graph database.

Basically you have to choose between types of "document" and "graph", this is because some features may not be present in all kinds of databases.

When you choose the database type, keep in mind the following diagram:
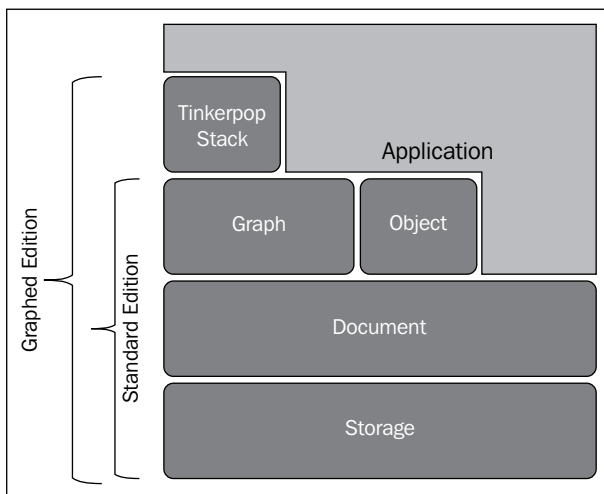
The first choice you have to make is whether you want to use a GraphDB or not.

If yes, then you have to decide if you will use the Gremlin language and the Tinkerpop stack or not. In this case you must use the Graphed Edition.

Another important consideration you have to evaluate is if you will be using OrientDB like an object database or not. As you can see in the preceding diagram, the graph and the object layer are on the same level. This means that each of them have specific features that cannot be used jointly. However keep in mind the following things:

- If you use the Standard Edition, you can always switch to the Graphed Edition, because they are fully compatible with each other.
- If you use a graph or an object database, you can always access the underlying document database layer. If you choose a document database, you will always use it as an object database within Java to persist and retrieve the data as POJO.

The following diagram helps you choose that:
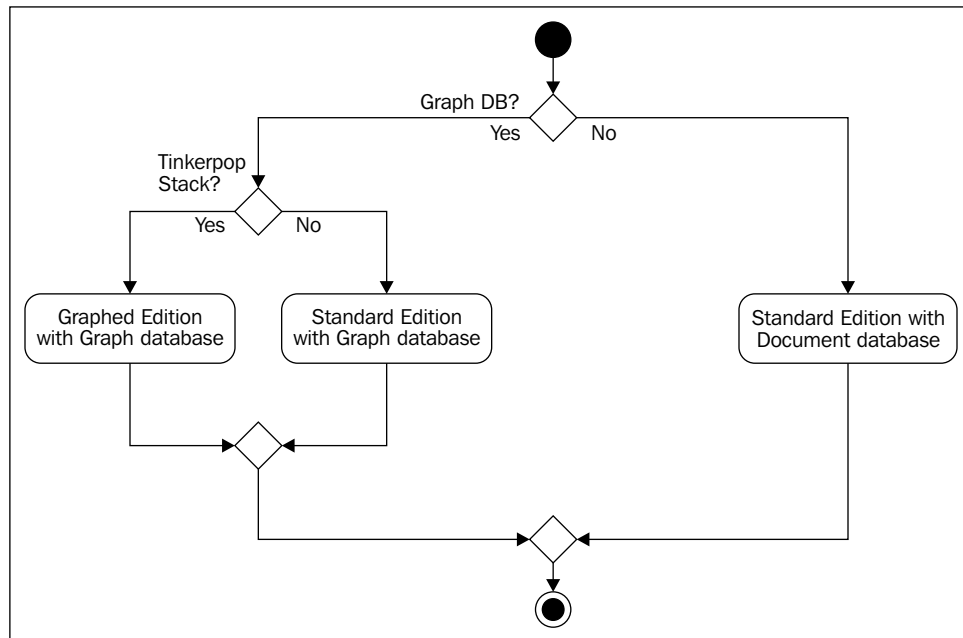
Data types

When a class is created into OrientDB, by default it is schema-less. This means that each record belonging to a class could have different fields (or properties in the OrientDB terminology) of any supported data type, and could even mean that two or more records could have same fields but with different value types. This behavior is by design. Of course, OrientDB also supports schema-full classes, that is, we can explicitly declare all or some fields and define the data type for them, specify whether they allow null values and whether they are mandatory or optional. Both schema modes have pros and cons.

The schema-less mode has the pro that it could be useful in those cases where we don't know which value will be inserted at the design time.

This is the case, for example, for a field that could store an arbitrary input from the user. This means that different records of the same class can store different value types in the same field. The mode with schema is just the opposite: OrientDB will check further record insertions validating the value type against the class schema, raising an error if the value is not of the expected type. This is very useful if we want data integrity. Possible property data types are given in the following table (remember that record, object, document are synonyms):

Type	Description	Value range
Simple types		
binary	An array of bytes	Range is not available
boolean	Possible values are: true/false	Values are false (stored as 0) and true (stored as 1)
byte	A single byte	Range is from -128 to 127
date	Any date as year, month and day. The default format is yyyy-mm-dd	Range is not available
double	A high precision decimal number	Range is from 4.9e-324 to 1.7976931348623157e+308
embedded	A document object embedded in a field	Range is not available
float	A decimal number	Range is from 1.4e-45f to 3.4028235e+38f
integer	An integer number	Range is from -2,147,483,648 to 2,147,483,647
long	A big integer number	Range is from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
short	A small integer number	Range is from -32,768 to 32,767

Type	Description	Value range
string	A string	Value is a string that can contain up to 100.000 chars
link	A link to another document; it stores only the RID in the form of `cluster:position`	Range is from -1 to 32767:9223372036854775807 (this means that you can have up to 32767 clusters, each of them can contain more than 90.000 billions of records)
Container types		
embeddedlist	A list of embedded ordered objects.	Range is up to 41,000,000 items
embeddedset	A set of embedded unordered objects.	Range is up to 41,000,000 items
embeddedmap	A map of key/value pairs. The values could be objects. The keys are strings.	Range is up to 41,000,000 items
linklist	An ordered list of RIDs pointing to other records	Range is up to 41,000,000 items
linkset	An unordered set of RIDs pointing to other records. RIDs cannot be duplicated.	Range is up to 41,000,000 items
linkmap	A map of key/value pairs. The values must be valid RIDs. The keys are strings.	Range is up to 41,000,000 items

Extended SQL

The OrientDB extended SQL is a proprietary extension of the SQL language. It seems similar to SQL, but it is actually not ANSI SQL compliant. However, if you know SQL you will be comfortable with it. The simplest way in which we can interact with an OrientDB server is through a command-line console, using the SQL. So let's start with it.

Be sure to have your OrientDB server up and running, and connect to it through the console, using the root credentials. Remember, the root password is stored in the XML config file:

```
C:\ >console
OrientDB console v.1.3.0 (build 16) www.orientechnologies.com
```

```
Type 'help' to display all the commands supported.
orientdb> connect remote:localhost root <your root password>
Connecting to remote Server instance [remote:localhost] with user
'root'...OK
orientdb>
```

Creating a database

Now let's create a simple DB, for example a minimalistic blog, with a class for the categories, another one for the posts, and a final one for the comments. So the first thing to do is to create a new fresh database.

Let's first try a document implementation of the database, and then a graph approach. To create a database, you must use the `create database` command. To get the complete list of the console's commands, type `help` inside it. The syntax of the `create database` command is as follows:

```
create database <database-url> <user> <password> <storage-type>
[<db-type>]
```

Where:

- `database-url`: Is the URL of the database to be created in the `<mode>:<path>` format. In our case it is: `remote:localhost/<database name>`.

- `user`: Is the root username.

- `password`: Is the root password.

- `storage-type`: Indicates if the database must be persisted or kept in memory (in this case, all data will be lost when the server stops). This parameter can be: `local` (for the persistent one) or `memory`. The default is `local`.

- `db-type`: Is the type of the database. Possible values are `document` or `graph`. By default, it is `document`.

So in the console, let's create our document database as follows:

```
orientdb> create database remote:localhost/minimalblog root
<your root password>
 local document
Creating database [remote:localhost/minimalblog] using the storage type
[local]...
Disconnecting from remote server [remote:localhost/]...OK
```

```
Connecting to database [remote:localhost/minimalblog] with user
'admin'...OK
Database created successfully.
Current database is: remote:localhost/minimalblog
orientdb>
```

OrientDB created a new document database called `minimalblog` and the console automatically connects to it as admin. In fact, every new database has three default roles and three default users as follows:

Role	Username	Password	Description
admin	admin	admin	The database (not the server!) administrator
writer	writer	writer	It can read, write, update, and delete any record of any class
reader	reader	reader	It can read any record of any class

The first thing to do is to change the default passwords. To do so, we must update the `password` field of the `OUser` class.

For example:

```
orientdb>update OUser set password="newadminpass" where name="admin"
Updated 1 record(s) in 0,003000 sec(s).
orientdb>update OUser set password="newreaderpass" where name="reader"
Updated 1 record(s) in 0,003000 sec(s).
orientdb>update OUser set password="newwriterpass" where name="writer"
Updated 1 record(s) in 0,003000 sec(s).
```

Note that OrientDB automatically applies the SHA-256 algorithm to the passwords, so they are not stored in plain text. You can verify this simply by executing the following statement:

```
select from OUser
```

For this reason, I suggest you write the password of the new users we just created.

> Class names are case insensitive whereas the names of the fields are case sensitive!

Creating classes

Since we are creating a minimalistic blog example, now we would have a `Categories` class, a `Posts` class, and an `Authors` class. To create these classes, we will use the `create class` command:

```
orientdb> create class Categories
Class created successfully. Total classes in database now: 7
orientdb> create class Posts
Class created successfully. Total classes in database now: 8
orientdb> create class Authors
Class created successfully. Total classes in database now: 9
```

OrientDB has created three classes and three related clusters with the same name as that of the classes. You can explore the database structure using the commands: `info clusters` and `desc <class name>`.

Inserting records

By default, all classes are schema-less, that is, we can "throw in" anything we want without any previous declaration. For example we could create a post like this:

```
orientdb> insert into posts (title,text) values ("My very first post",
"This is the very first post I wrote!")
Inserted record 'Posts#9:0{title:My very first post,text:This is the very
first post I wrote!} v0' in 0,265000 sec(s).
```

As you can see, the OrientDB console replies showing us the result of the `insert` command. It writes the name of the cluster, the **Record ID** (**RID**) of the record (`#9:0`), and a JSON-like string representing that the record is inserted. Note that this output is not a valid JSON string, it is just a reminder of what the database has performed. You can even perform several inserts with a single statement. In this case, you can declare the tuples one by one:

```
orientdb> insert into posts (title,text) values ("title1", "text1")
("title2","text2") ("title3","text3")
Inserted record '[Posts#9:16{title:title1,text:text1} v0, Posts#9:17{t
itle:title2,text:text2} v0, Posts#9:18{title:title3,text:text3} v0]' in
0,031000 sec(s).
```

Deleting records

To delete a single record or group of records, you can use the `Delete` statement that acts exactly the same as in Standard SQL. If you want to delete all records of a class or of a cluster, you can use the `Truncate` command.

> When you delete a record, its RID becomes available to be reassigned to a new record, so you must pay great attention to manage deletions.

Reading and updating records

To read the inserted records there is, of course, the `select` command:

```
select [<projections>] from <target> [where <conditions>] [group by
<field>] [order by <fields> [asc|desc]] [skip <numRecords>] [limit
<MaxRecords>]
```

Where:

- `projections`: They are the fields used to retrieve each record. Note that if you want to retrieve all the fields, you can omit the `*` character that is mandatory in SQL standard. The projections could contain any supported SQL functions and a special operator, `LET`.

- `target`: Is the database object on which we execute the query. It can be a class, a cluster, a RID, or a set of RIDs. This is very important. If you want to perform a query on a single RID or a set of them, you can perform the query directly on it/them. This is much faster than a query on a class. For example: `select from #23:1` is much faster than `select from Posts where @rid=#23:1`.

- `where`: This specifies the conditions to be applied to select the records.

- `group by`: This is similar as in standard SQL, which allows us to specify a field to group by. Note that at the moment, only one field is supported.

- `order by`: This is similar as in standard SQL, which allows us to specify a list of fields to use as a criteria for sorting. It is possible to indicate the sorting criteria for each of them (`ASC` to specify ascending order or `DESC` for descending order).

- `skip` and `limit`: These can be used in paginations. The first one allows us to skip a number of records, meanwhile the second one allows us to specify the number of records to return.

For example:

```
orientdb> select from posts


---+---------+-------------------+-------------------
  #| RID      |title             |text
---+---------+-------------------+-------------------
  0|     #9:0|My very first post |This is my very first post I wrote!
  1|     #9:1|title1             |text1
  2|     #9:2|title2             |text2
  3|     #9:3|title3             |text3
---+---------+-------------------+-------------------
4 item(s) found. Query executed in 0.076 sec(s).
```

Let me explain the output:

In the first column there is a simple counter. It is added by the console tool, it is not a records field. In the second column, there are records' RIDs. This is the record ID and it univocally identifies each record. Other fields are the ones that we have inserted. Now let's execute the following command:

```
orientdb> select title from Posts
---+---------+--------------------
  #| RID      |title
---+---------+--------------------
  0|    #-2:1|My very first post
  1|    #-2:2|title1
  2|    #-2:3|title2
  3|    #-2:4|title3
---+---------+--------------------
4 item(s) found. Query executed in 0.037 sec(s).
```

What happened to the RIDs?

This is a very important concept in the OrientDB programming and may cause you many issues if you do not handle it properly. The RIDs are not simply logical identifiers, but they are physical pointers to a file indicating where the records' data begins. When you execute a `select` command providing a projection, the returned records aren't the ones stored on files; but they are virtual records built at the runtime and have no physical pointers. In these cases, OrientDB returns negative RIDs. To obtain the RIDs of the parsed records, you can use the special `@rid` keyword:

```
orientdb> select @rid as RealRid,title from Posts

---+---------+-------------------+-------------------
  #| RID     |RealRid            |title
---+---------+-------------------+-------------------
  0|    #-2:1|#9:0               |My very first post
  1|    #-2:2|#9:1               |title1
  2|    #-2:3|#9:2               |title2
  3|    #-2:4|#9:3               |title3
---+---------+-------------------+-------------------
4 item(s) found. Query executed in 0.032 sec(s).
```

As you can see, we can use aliases in query statements. Other special field names are as follows:

- `@class`: This specifies the class of the record.
- `@version`: This specifies the version number of the record. Every time a record is updated, its version number changes. This is useful for checking concurrent conflicts.
- `@size`: This specifies the record size in bytes.
- `@type`: This specifies the record type. It can be `document` or `binary`. The `binary` records are used to store blob objects.
- `@this`: This specifies the record itself.

Try it yourself by executing:

```
select @class,@rid,@version,@size,@type from Posts
```

To update one or more properties, we can use the `update` statement:

```
update <class>|<RID> set|increment [<field-name> = <field-value>] [,
<field-name> = <field-value>]* [where  <conditions>] [limit <max-
records>]
```

Where:

- `class/cluster`: Is the class or the cluster name
- `field-name`: Is the field(s) name to be updated
- `field-value`: Is the new value to be assigned
- `conditions`: Is the filter condition to select the records to update
- `limit`: Limits the update to only the records that satisfy the `where` condition

Note that the `update` command can have the `set` or `increment` keywords in it. The `set` command operates the same as in the standard SQL, the keyword `increment` allows you to increment the field value by the specified number. This is useful for counters. You can use negative numbers to decrement the value. Note that since we are using schema-less classes, we can update the fields with a new and different value type.

For example:

```
orientdb> update #9:0 set title=2
Updated 1 record(s) in 0,003000 sec(s).
```

In this example, I updated the RID `#9:0` which should be the first record belonging to the `Posts` class and I set the `title` field with an integer value. OrientDB has no objection to this. Like in the select statements, updates performed directly on RID or a set of them are more efficient than selecting them by a `where` condition.

Fields

By default OrientDB tries to infer the fields type by their content. So if we try to execute the following statements:

```
create class Examples;

insert into examples (num) values (2) (3) (4);

select sum(num) from Examples;

---+---------+--------------------
  #| RID     |sum
---+---------+--------------------
  0|    #-2:0|9
---+---------+--------------------
1 item(s) found. Query executed in 0.022 sec(s).
```

We can see that OrientDB understands that the inserted fields are numbers and when we execute a sum() on the num fields, it replies correctly. If we insert a new record as follows:

```
insert into examples (num) values ("5")
```

OrientDB inserts the record and interprets it like a string. In fact, if we execute the query again, we will obtain the same result: 9.

OrientDB will skip non-numeric values.

Embedded documents

A particular field type is the embedded document. This type allows us to associate any valid document object expressed as a valid JSON string with a property. Let's see the code:

```
orientdb> insert into examples (embedded) values
({"name":"James","age":23})
Inserted record 'Examples#12:4{embedded:{age=23, name=James}} v0' in
0,001000 sec(s).
```

Execute the following query to see the result:

```
orientdb> select from Examples
---+---------+-------------------
  #| RID     |num
---+---------+-------------------
  0|    #12:0|4
  1|    #12:1|3
  2|    #12:2|2
  3|    #12:3|5
  4|    #12:4|null               |{age=23, name=James}
---+---------+-------------------+-------------------
5 item(s) found. Query executed in 0.028 sec(s).
```

The last field has no column name due to a limitation of the console tool, because at the start of the execution, it cannot know the names of all the fields that will be retrieved.

```
orientdb> select from 12:4
---+---------+-------------------
  #| RID     |embedded
---+---------+-------------------
  0|    #12:4|{age=23, name=James}
---+---------+-------------------
1 item(s) found. Query executed in 0.018 sec(s).
```

A very important thing to point out about the embedded records is that they have no RIDs and they live within the scope of the parent record. If you delete the parent record, its embedded records get deleted. The best thing about the embedded records is that you can use their own fields in projections and selections in query statements. For example:

```
orientdb> select from Examples where embedded.age=23
---+---------+--------------------
 #| RID     |embedded
---+---------+--------------------
 0|    #12:4|{age=23, name=James}
---+---------+--------------------
1 item(s) found. Query executed in 0.111 sec(s).
```

You can of course embed a document inside an embedded document.

Containers

Containers are special fields that can contain a set of other fields. There are three kinds of containers, and each of them can contain embedded documents or RIDs that point to non-embedded records. The RIDs contained in record fields are also called links. This is because they establish a physical link between records. The containers could be as follows:

- `set`: This is an unordered set of elements. An element cannot be inserted more than once.
- `list`: This is an ordered sequence of elements. An element could be present more than once.
- `map`: This is a set of key/value pairs, where keys are strings and values could be any of the allowed values, even other containers.

Like other field types, OrientDB also tries to guess the container type.

To identify a container value you must use the square brackets, similar to the JSON syntax. So we can write the following statement:

```
orientdb> insert into Examples (embset) values ([{"name":"John","age":
24},{"name":"Barbara","age":23}])

Inserted record 'Examples#12:5{embset:[2]} v0' in 0,078000 sec(s).
```

Let's see it:

```
orientdb> select from Examples where embset is not null
---+---------+--------------------
 #| RID     |embset
---+---------+--------------------
 0|    #12:5|[2]
---+---------+--------------------
1 item(s) found. Query executed in 0.014 sec(s).
```

The console does not show the contents of the container field but only its cardinality. To show its content we can use the `flatten()` function as follows:

```
select flatten(embset) from Examples where embset is not null
```

> The `flatten()` function must be the only parameter in the projection. You cannot mix record fields and the `flatten()` function in the same projection.

Schema-full classes and the mixed-mode schema

So far, we have seen how to use the classes in the schema-less mode. OrientDB gives us the possibility to specify the field type to create a constraint in the schema. We can even declare whether a class must be schema-full, that is, every field must be declared; or mixed-mode, that is, in the `insert` commands, we can use the non-declared fields and at the same time, we can also have a check by the OrientDB engine against the declared ones. To create a class field or property, we need to use the `create property` command:

```
create property <class>.<property> <type> [<linked type>|<linked
class>]
```

Where:

- `class` is the class name
- `property` is the property name
- `type` is the property type
- `linked_type` is used only if the type is a link or a container
- `linked_class` is used only if the type is a link or a container

In our minimal blog example, we can code as follows:

```
create property Posts.pubDate date
```

This means that we want to create a `date` property on the `Posts` class.

Now if we try to insert an invalid value (that is an invalid date) in the `pubDate` field, OrientDB will throw an exception. Let's try it:

```
orientdb> insert into Posts (title,text,pubDate) values ("This is the
title","This is the text","This is the date")

Error: com.orientechnologies.orient.core.exception.
OQueryParsingException: Error on conversion of date 'This is the date'
using the format: yyyy-MM-dd HH:mm:ss
orientdb>
```

OrientDB cannot cast the string value, `This is the date` in a valid date value, and suggests the correct string format:

```
orientdb> insert into Posts (title,text,pubDate) values ("This is the
title","This is the text","2013-01-29 13:24:45")
Inserted record 'Posts#9:12{title:This is the title,text:This is the
text,pubDate:Tue Jan 29 00:00:00 CET 2013} v0' in 0,001000 sec(s).
```

Note that the time in the server response is gone! This is because we chose a `date` type instead of `datetime`, so pay attention when you choose the fields' data type. In our example, our `Posts` class is now in the mixed-mode schema because we inserted records with fields that were not previously declared. If we want to avoid this, we could force the schema-full mode for the `Posts` class:

```
ALTER CLASS Posts STRICTMODE true
```

Now it is impossible to execute the commands as follows:

```
orientdb> insert into Posts (title,text,pubDate) values ("This is the
title","This is the text","2013-01-29 13:24:45")
Error: com.orientechnologies.orient.core.exception.
OCommandExecutionException: Error on execution of command: OCommandSQL
[text=insert into Posts (title,text,pubDate) values ("This is the
title","This is the text","2013-01-29 13:24:45")]
Error: com.orientechnologies.orient.core.exception.
OValidationException: Found additional field 'title'. It cannot be
added because the schema class 'Posts' is defined as STRICT
```

We must declare each field we intend to use. Note the old records are still there. The `ALTER CLASS` command is not retroactive. So we have to execute the following:

```
orientdb> create property Posts.title string
Property created successfully with id=1
orientdb> create property Posts.text string
Property created successfully with id=2
```

After we have created the properties, we can declare more constraints on them using the `alter property` command. We can declare a field as mandatory, or not-nullable, or declare a range for the number fields:

```
alter property <class>.<property> <attribute-name> <attribute-value>
```

As stated in the OrientDB wiki page:

Class is the class owner of the property

- `property` is the name of the property to be altered
- `attribute-name` is the attribute name to be altered

  Supported attribute names are as follows:

  - ° `linkedclass` is the linked class name if the field is a link. To remove the attribute, set it to NULL.

  - ° `linkedtype` is the linked type name between those supported. To remove the attribute, set it to NULL.

  - ° `min` is the minimum value constraint. It accepts strings, numbers, or dates as values according to the field type. To remove the attribute, set it to NULL.

  - ° `mandatory` is a boolean value, set it to `true` if the property is mandatory.

  - ° `max` is the maximum value constraint. It accepts strings, numbers, or dates as values according to the field type. To remove the attribute, set it to NULL.

  - ° `name` is the class name.

  - ° `notnull` is a boolean value. Set it to `false` to allow null values.

  - ° `regexp` defines a regular expression as constraint. The field value will be validated against the specified `regexp` parameter. To remove the attribute, set it to NULL.

  - ° `type` is the new type between those supported. To remove the attribute, set it to NULL. These changes do not affect data that is already saved, so keep in mind that the new type should be compatible with the previous one.

  - ° `attribute-value` is the new attribute value to set according to the attribute-name.

You can also define your own custom attributes which are useful, for example, to represent UML stereotypes. Syntax is `name = value`. For example: `stereotype = icon`.

> You cannot alter more than one attribute at a time.

Relationships

The concept of relationship in OrientDB is different from the world of relational databases. Since OrientDB is not a relational database, the relational `JOIN` operator makes no sense. Furthermore, when you build a relationship in OrientDB, you can establish a physical connection between the documents. This means that a `JOIN` operation has no computational cost to retrieve a related document from a given one. This makes OrientDB very fast when managing relationships between millions of records. In OrientDB, there are two kinds of relationships, which are as follows:

- **Referenced**: They are used to create associations between documents
- **Embedded**: They are stronger relationships, similar to a composition in a UML diagram

The embedded relationship is useful to model links between documents that have a life cycle dependency, that is, the embedded document cannot exist without the embedding one.

One-to-one and one-to-many relationships

To create this kind of relationship you have to create a field of the `link` data type. This data type in fact accepts RIDs as values. In this way you can physically create a link between documents. In our example of a minimal blog, we can have a 1:n relationship between authors and posts, where one author will write several posts. We can model this by executing the following command:

```
orientdb> create property Posts.author link Authors
Property created successfully with id=3
orientdb> alter property Posts.author mandatory=true
Property updated successfully
orientdb> alter property Posts.author notnull=true
Property updated successfully
```

The `Posts` class schema should appear as follows:

```
orientdb> desc posts
Class................: Posts
Default cluster......: posts (id=9)
Supported cluster ids: [9]
Properties:
--------+------+-----------------+---------+--------+---------+---+-
--+
 NAME   |TYPE  | LINKED TYPE/CLASS|MANDATORY|READONLY| NOT
NULL|MIN|MAX|
--------+------+-----------------+---------+--------+---------+---+-
--+
 pubDate|DATE  | null             |false    |false   | false   |   |
|
 author |LINK  | Authors          |false    |false   | false   |   |
|
 title  |STRING| null             |false    |false   | false   |   |
|
 text   |STRING| null             |false    |false   | false   |   |
|
--------+------+-----------------+---------+--------+---------+--
orientdb>
```

You may want to do exactly the opposite, that is, to reference all the posts written by an author within the `Author` class. In this case, you can use the link containers, field data type, which are `linklist`, `linkset`, and `linkmap`. This is an important choice to be made during the analysis phase of your application. OrientDB is quick in following the links between documents, but this is only true in one direction. So in our blog example, if we think that the application will perform queries from posts to obtain their authors, we can use the `link` field in the `Posts` class. If vice versa, the application more often will execute queries on authors to know their posts. We can consider using a `linklist` field in the `Authors` class.

> We can use both approaches but we must pay attention to the management of inversion relationships because OrientDB does not manage them. It is our task to keep the relationships in a consistent state.

Embedded relationships

The embedded 1:1 relationship is realized by embedding a document in a field of its owner. The embedded documents have no RIDs and follow the life cycle of the embedding document. To realize a 1:n embedded relationship, you can use one of the embedded container field type, that is, `embeddedlist`, `embeddedset`, or `embeddedmap` (if you want to associate a key to any embedded record). In our blog example, we can imagine that the comments of an article are embedded into the article itself. In fact, if a post is deleted, the comments also should be deleted. In our case, the code as follows:

```
orientdb> create property Post.comments embeddedlist
Property created successfully with id=3
```

I chose the `embeddedlist` type because I want to preserve the insert order in the list. If I had chosen the `embeddedset` type, this order could not be guaranteed.

Many-to-many relationships

In a many-to-many relationship, a graph database expresses all its potential and shows a great difference compared to the relational ones. This relationship can exist only between records that are not embedded.

Referenced relationships

The n:m relationship (also known as many-to-many relationship) between two entities are modeled in a relational database using a third table that contains the primary keys of both tables. In OrientDB this approach is not necessary. In fact, you can use the link containers fields similar to the ones in the one-to-many relationship; in this case, on both sides of the relationship. Always keep in mind that OrientDB doesn't support inversion relationships. In our blog example, we can assume that in a category there can be many posts, and that a post can belong to many categories. In this case, we could create a many-to-many relationship in the following way:

```
orientdb> create property Posts.categories linklist Categories
Property created successfully with id=5
orientdb> create property Categories.posts linklist Posts
Property created successfully with id=1
```

Traversing the relationships

If you have an embedded or referenced many-to-one or one-to-one relationship, you could traverse the relationship just using the '.' object notation exactly like you do with objects, or if you have experience with Hibernate (in **Hibernate Query Language** (**HQL**)). So in our minimal blog database, to select the `Author` of a `Post`, we would write the following:

```
select author.name from Post where @rid=#xx:yy
```

Where:

- `author`: Is the `Posts` field name of the link to the `Author` class
- `name`: Is the field of the `Author` class which contains the `Author` name
- `@rid`: Is the implicit field representing the record ID
- `#xx:yy`: Is the RID value of the post, which we are selecting

For performance reasons, we could write the same statement in the following way:

```
select author.name from #xx:yy
```

In OrientDB, there is also the `Traverse` command. It is a powerful command especially used to perform queries on graphs, but it is also useful in a document database.

- It can be used to traverse many-to-one or many-to-many relationships because it can traverse all the links present in container fields, in addition to the linked one.
- In fact, it retrieves all the linked documents crossing the defined relationships among them with just one statement. The syntax is:

  ```
  traverse [<fields>|*|any()|all()] from <target> [while
  <condition>] [limit <max-records>]
  ```

  Where:

    ° `fields`, `*`, `any()`, `all()`: Here we can specify the name of the properties that contain the links. If we use `*`, `any()`, or `all()`, all the fields are traversed.

    ° `target`: It could be a class name, a list of clusters, a RID (or a list of RIDs), or even another traverse command or a select statement.

    ° `condition`: It is similar to a `where` clause in the `select` statement. This is the condition to be applied to evaluate whether it follows a link or not.

    ° `max-records`: It is the maximum number of records to be retrieved.

- OrientDB manages the recursions and keeps track of the already visited documents to avoid retraversing.

- Suppose that we want to know all the authors who have written at least one post, starting from the categories class, we write the following statement:

  ```
  traverse posts from Categories
  ```

To execute the preceding statement, first let's populate our database.

- Clean up the Posts class as follows:

  ```
  truncate class Posts
  ```

- Populate the Categories class as follows:

  ```
  insert into categories (name) values ("cars") ("bike")
  ("motorbike")
  ```

- Populate the Authors class as follows:

  ```
  insert into Authors (name) values ("john") ("mary") ("jane")
  ```

- Retrieve the Authors' RIDs:

  ```
  orientdb> select from authors
  ---+---------+--------------------
    #| RID     |name
  ---+---------+--------------------
    0|    #15:0|john
    1|    #15:1|mary
    2|    #15:2|jane
  ---+---------+--------------------
  3 item(s) found. Query executed in 0.098 sec(s).
  ```

> Note that the RIDs could be different on your system depending on the execution order of the class creation statements.

- Populate the Posts class:

  ```
  insert into Posts (title,text,pubDate,author) values ("A very good
  car","This is a very good car",sysdate(),#15:0) ("A smart bike","A
  bike very cool!",sysdate()
  ,#15:1) ("Another supercar","Remember the old days?...",
  sysdate(),#15:0)
  ```

- The sysdate() function returns the current server date. The last value of each tuple is the Author RID, so adjust it if in your system the Author class has a different ID other than 15.

Now we have three posts, let's see them. We need their RIDs to put in the `Categories` class. This is because we have chosen to model the database in the following way:

```
orientdb> select from posts


---+---------+-------------------+-------------------+-------------
------+--------------------
  #| RID     |title              |text               |pubDate
|author
---+---------+-------------------+-------------------+-------------
------+--------------------
  0|     #9:0|A very good car    |This is a very good car|2013-02-05
23:00:00 |#15:0
  1|     #9:1|A smart bike       |A bike very cool!  |2013-02-05
23:00:00 |#15:1
  2|     #9:2|Another supercar   |Remeber old days?...|2013-02-05
23:00:00 |#15:0
---+---------+-------------------+-------------------+----------
3 item(s) found. Query executed in 0.064 sec(s).
```

Now let's insert the `Posts` into the categories:

```
update Categories add posts = #9:0 where name="cars"
update Categories add posts = #9:2 where name="cars"
update Categories add posts = #9:1 where name="bike"
```

Our database is ready, so we can execute a `traverse` command:

```
traverse all() from Categories
orientdb> traverse all() from categories


---+---------+-------------------+--------------------
  #| RID     |name               |posts
---+---------+-------------------+--------------------
  0|     #8:0|cars               | [2]
  1|     #9:0|null               |null               |A very good
car     |This is a very good car|2013-02-05 23:00:00 |#15:0
  2|    #15:0|john               |null               |null
|null               |null               |null
  3|     #9:2|null               |null               |Another
supercar    |Remeber old days?...|2013-02-05 23:00:00 |#15:0
```

```
   4|    #8:1|bike                |[1]                 |null
|null               |null                |null
   5|    #9:1|null                |null                |A smart bike
|A bike very cool!   |2013-02-05 23:00:00 |#15:1
   6|   #15:1|mary                |null                |null
|null               |null                |null
   7|    #8:2|motorbike           |null                |null
|null               |null                |null
---+---------+--------------------+--------------------+----------
8 item(s) found. Traverse executed in 0.217 sec(s).
```

Let's see what happened. The `traverse` command starts from the `Categories` class and for each record, it follows the available links. However, the exploration of the links does not stop at the linked record, but goes further to its link and so on. In fact, you can see in the server reply that all nodes were traversed. `Traverse` provides some context variables to help us. They are as follows:

- `$parent`:It accesses the parent of the actual record
- `$current`: It retrieves the current record
- `$depth`: It retrieves the current depth of nesting
- `$path`: It retrieves the path of the current record from the root of traversing

For example to see how the recursion of the `traverse` command works, try to execute the following code:

```
orientdb> select $path from (traverse all() from categories)
---+---------+--------------------
  #| RID     |$path
---+---------+--------------------
  0|    #-2:1|#8:0
  1|    #-2:2|#8:0.posts.#9:0
  2|    #-2:3|#8:0.posts.#9:0.author.#15:0
  3|    #-2:4|#8:0.posts.#9:2
  4|    #-2:5|#8:1
  5|    #-2:6|#8:1.posts.#9:1
  6|    #-2:7|#8:1.posts.#9:1.author.#15:1
  7|    #-2:8|#8:2
---+---------+--------------------
8 item(s) found. Query executed in 0.028 sec(s).
```

To select only the authors who have written at least one article execute the following code:

```
orientdb> select from (traverse all() from categories) where @
class="Authors"
---+---------+--------------------
  #| RID     |name
---+---------+--------------------
  0|    #15:0|john
  1|    #15:1|mary
---+---------+--------------------
2 item(s) found. Query executed in 0.016 sec(s).
```

To select only the authors who have written at least one article in the category cars, execute the following code:

```
orientdb> select from (traverse * from (select from categories where
name="cars")) where @class="Authors"
---+---------+--------------------
  #| RID     |name
---+---------+--------------------
  0|    #15:0|john
---+---------+--------------------
1 item(s) found. Query executed in 0.027 sec(s).
```

SQL functions

OrientDB supports many functions to use in the select statement in both projections and selections. You can find a complete and updated list of these functions in the official documentation wiki page at `https://github.com/nuvolabase/orientdb/wiki/SQL-Where`.

This page is kept up to date as new functions are developed. One important thing to note is that you can extend the SQL parser capabilities by implementing your own functions. Once you write a new function, you will be able to use it in your SQL statements. You can write functions in Java or in JavaScript. However, there are some conceptual differences between these two approaches. The first one is written in Java in your application and basically expands the parser syntax; the second one is deployed in a special cluster into the database itself and is available to any client, not only in your application. It could be used via the console tool via REST API calls, and so on. You could see them like a sort of stored procedures. To write a custom SQL function in Java, you must register it to the database engine.

This is the code of the `coalesce()` function that I wrote for OrientDB and that is available since Version 1.3.0:

```
OSQLEngine.getInstance().registerFunction("coalesce",
  new OSQLFunctionAbstract("coalesce", 1, 1000) {

    @Override
    public String getSyntax() {
      return "Returns the first non-null parameter or null
if all parameters are null. Syntax: coalesce(<field|value>
[,<field|value>]*)";
      }

    @Override
    public Object execute(OIdentifiable iCurrentRecord, ODocument
iCurrentResult,final Object[] iParameters, OCommandContext iContext)
{
      int length=iParameters.length;
      for (int i=0;i<length;i++){
        if (iParameters[i]!=null) return iParameters[i];
      }
      return null;
    }
  }
);
```

Basically, you must extend `OSQLFunctionAbstract` and override the `execute()` method. To the constructor, you have to pass the name of the function and the minimum and maximum number of parameters that the function will accept. The `coalesce()` function returns the first non-null value among its parameters, and could accept at least one parameter, up to 1,000.

The graph database

Until now, we have seen the document database features. Let's see what they are while we operate on graph databases. The first thing to do is to create a new database. We must specify that we want a graph database instead of a normal document database as follows:

```
orientdb> create database remote:localhost/cities root <root_pass>
local graph
Creating database [remote:localhost/cities] using the storage type
[local]...
Disconnecting from remote server [remote:localhost/minimalblog]...OK
Disconnecting from the database [minimalblog]...OK
```

```
Connecting to database [remote:localhost/cities] with user 'admin'...
OK
Database created successfully.
Current database is: remote:localhost/cities
```

I created a new database called `cities`. What I want to do is to build a graph whose vertices are European cities, connected by edges whose weights are the distances in km among them.
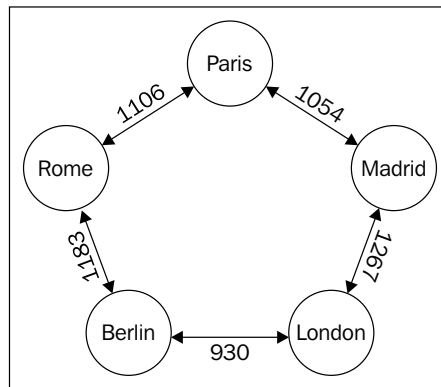
We have created a database, so let's inspect it by using the info command. We can note that there are two new classes; with their respective clusters as follows:

- `OGraphVertex`: This is used to declare the vertices of the graph
- `OGraphEdge`: This is used to connect the vertices between them

Each vertex has a pair of predefined fields **in** and **out**, which are linkset. Each edge has a pair of fields **in** and **out** which are links. This means that each vertex could be linked to many edges, but each edge must have just two linked vertices:
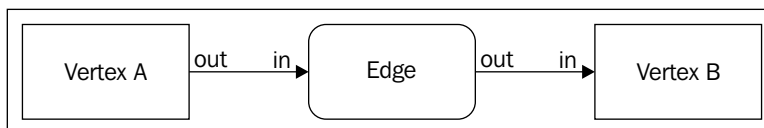
Both `OGraphVertex` and `OGraphEdge` are no more than Documents classes which have special meaning for the OrientDB engine. As their records are like other documents, we can of course declare new properties on them. We can also create new classes that extend them, creating more specialized vertices and edges. Furthermore, we can use two letters as aliases to indicate the `OGraphVertex` and `OGraphEdge` classes. They are V and E. We have already made this introduction, so let's populate our database.

First of all, let's create the nodes. To create a vertex we must use the following command:

```
create vertex
```

Don't use the `insert` statement!

```
orientdb> create vertex set city="Rome"
Created vertex 'V#8:0{city:Rome} v0' in 0,058000 sec(s).
orientdb> create vertex set city="London"
Created vertex 'V#8:1{city:London} v0' in 0,001000 sec(s).
orientdb> create vertex set city="Berlin"
Created vertex 'V#8:2{city:Berlin} v0' in 0,001000 sec(s).
orientdb> create vertex set city="Madrid"
Created vertex 'V#8:3{city:Madrid} v0' in 0,001000 sec(s).
orientdb> create vertex set city="Paris"
Created vertex 'V#8:4{city:Paris} v0' in 0,001000 sec(s).
```

Now let's create the edges by using the `create edge` command. Since our graph is bidirectional, we have to create two edges for each pair of nodes as follows:

```
create edge from #8:0 to #8:4 set distance=1106
create edge from #8:4 to #8:0 set distance=1106
create edge from #8:4 to #8:3 set distance=1054
create edge from #8:3 to #8:4 set distance=1054
create edge from #8:3 to #8:1 set distance=1267
create edge from #8:1 to #8:3 set distance=1267
create edge from #8:1 to #8:2 set distance=930
create edge from #8:2 to #8:1 set distance=930
create edge from #8:2 to #8:0 set distance=1183
create edge from #8:0 to #8:2 set distance=1183
```

> Note that the RIDs used in the preceding statements are the output of the previous vertices of the create statements. They could be different in your database.

So now we have five vertices and 10 edges. You can visualize the graph using the OrientDB Studio as follows:

1. Open your browser and open the link `http://localhost:2480/studio`.
2. Select the `cities` database and provide the admin credentials (by default admin/admin).
3. Go to the **Query** tab and execute the query.
4. Select from **V**.
5. Click on the first value of the column `@rid`.
6. Now click on the **Graph** button.

You will see a graph. Try to make the following selections:

1. In the **Depth** field enter `10`.
2. Under **Fields** select **city** and **distance**.
3. Now click on **Reload** and then on **Circular Layout**.

You should see something like the following screenshot:
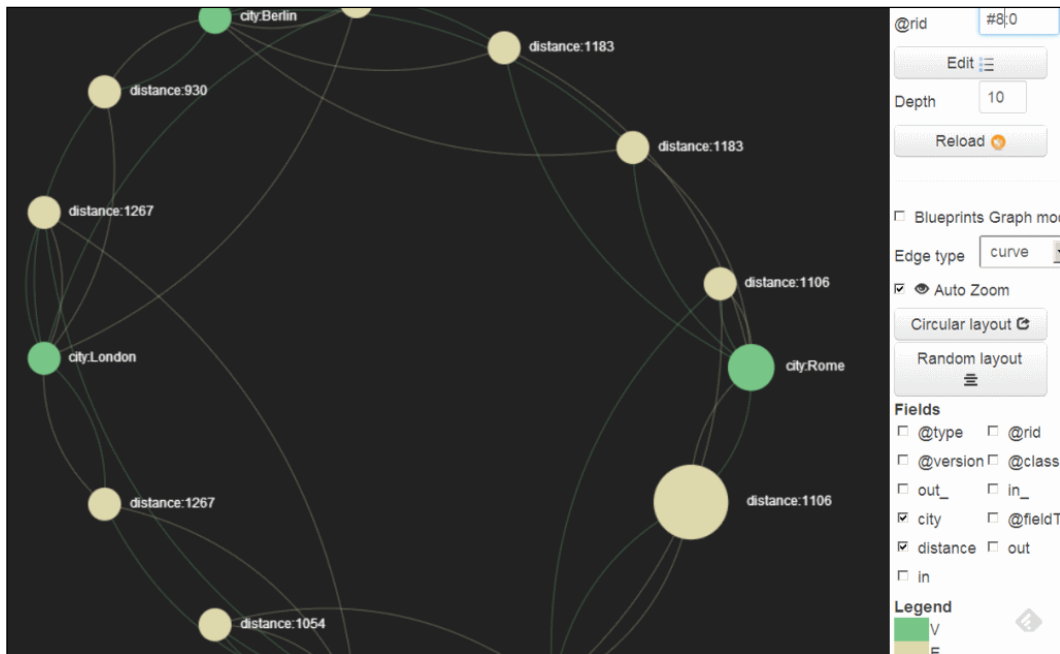
You can use the `Traverse` command to perform operations on the database. For example, to find all the cities linked to a given one:

```
select city from (traverse V.out, E.in from #8:0 while $depth < 3 )
where @class="OGraphVertex" and $dept > 1
```

Here, `#8:0` is the starting vertex RID. The nested `traverse` command returns all vertices and edges starting from `#8:0` and stops the traversal operation when it reaches the depth of two. The external `where` clause filters only the vertices (the traverse returns the edges too), and excludes the root node. OrientDB provides the `dijkstra()` function to calculate the minimal path between two nodes:

```
select flatten(dijkstra(#8:0,#8:3,"distance"))
```

Now we will print the node traversed to go from `#8:0` to `#8:3`, covering the minimum distance.

Using the JDBC driver

The JDBC driver is a separated open source project. You can find it at `https://github.com/nuvolabase/orientdb-jdbc` and it is maintained by the same team that developed the OrientDB engine. Currently, the JAR file can be downloaded at the `typesafe maven` repository which is located in `http://repo.typesafe.com/typesafe/snapshots/com/orientechnologies/orientdb-jdbc/1.0-SNAPSHOT/`.

To use the driver, you must deploy its JAR file in `CLASSPATH`. After this step, you can use it like any other JDBC driver to execute SQL queries and commands. For example:

```
/* declare admin credentials */
Properties credentials = new Properties();
credentials.put("user", "admin");
credentials.put("password", "newadminpass");

/* obtain a connection */
Connection conn = (OrientJdbcConnection) DriverManager.getConnection("
jdbc:orient:remote:localhost/minimalblog", credentials);

/* perform a query */
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT title, text, pubDate from
Posts order by pubDate desc");

/* consume the resultset */
```

```
SimpleDateFormat dateFormatter = new SimpleDateFormat("MMMMM dd,
yyyy");
while (rs.next()) {
  String title = rs.getString("title");
  String text = rs.getString("text");
  Timestamp timestamp = rs.getTimestamp("pubDate");
  String date= dateFormatter.format((java.util.Date) timestamp);
  System.out.printf("%s\n%s\n%s\n\n\n", date, title, text);
}
rs.close();
stmt.close();
conn.close();
```

Note that we know the fields' type because `Posts` is a schema-full class.

Other language drivers such as PHP

Currently, there is more than one library project that implements drivers for the PHP language. Two of them are: OrientDB-PHP at `https://github.com/AntonTerekhov/OrientDB-PHP` and orientdb-odm at `https://github.com/doctrine/orientdb-odm`.

I mentioned both of them because they have different approaches to implementing the data exchange between the PHP page and the OrientDB server. The first one uses the OrientDB binary protocol, the same used by the console tool, and the second one uses the OrientDB REST APIs, and could be useful when you can only use HTTP. Take a look at the pages and feel free to choose the one that best fits your needs.

The native Java API

Of course, the "preferred" way to communicate and interact with OrientDB is using its native JAVA APIs. In *Appendix* (available at `http://www.packtpub.com/sites/default/files/downloads/9956OS_Appendix.pdf`), you will find a quick reference to the common APIs. To use the APIs you must include the following libraries in your classpath:

- `orient-commons-*.jar`
- `orientdb-core-*.jar`
- `orientdb-client-*.jar`
- `orientdb-enterprise-*.jar`
- `orientdb-object-.jar` (only if you use the object interface)
- `javassist.jar` (only if you use the object interface)

- `blueprints-core-*.jar` (only if you use the Tinkerpop interface)
- `blueprints-orient-graph-*.jar` (only if you use the Tinkerpop interface)
- `pipes-*.jar` (only if you use the Tinkerpop interface)
- `gremlin-java-*.jar` (only if you use the Tinkerpop gremlin language)
- `gremlin-groovy-*.jar` (only if you use the Tinkerpop gremlin language)
- `groovy-*.jar` (only if you use the Tinkerpop gremlin language)

Depending on which type of database you need to connect to, you must use a different connection class:

- `ODatabaseDocumentTx` for Document databases
- `OGraphDatabase` for Graph databases
- `OrientGraph` for working with the Tinkerpop blueprints implementation
- `OObjectDatabaseTx` for object databases

Opening a connection

Basically all you have to do to obtain a connection to the server is to use the aforementioned classes.

If you plan to work in a multithreaded environment, you must pay attention when using the connection classes. In fact, these are not thread-safe and each thread needs a new connection instance. Each database instances share the schema, the index manager, and the security manager. The accesses to these resources are synchronized among the instances. However, when you open a connection to a database, the connection instance is set into the current ThreadLocal space. This means that everything related to the management of the correct connection instance is hidden to the developers. To open a connection:

- For a document database we can open a connection as follows:

  ```
  ODatabaseDocumentTx db = new ODatabaseDocumentTx("remote:localho
  st/minimalblog").open("admin", "newadminpass");
  ```

- For a graph database we can open a connection as follows:

  ```
  OGraphDatabase database = new OGraphDatabase("remote:localhost/
  cities").open("admin", "admin");
  ```

- For an object database as follows:

  ```
  OObjectDatabaseTx db = new OObjectDatabaseTx ("remote:localhost/
  minimalblog").open("admin", "newadminpass");
  ```

- A Tinkerpop implementation of the graph database is as follows:

```
OrientGraph graph = new OrientGraph("remote:localhost/
cities","admin","admin");
```

Connection pools

In a multithreaded environment (such as a web application), you can use the connection pool instead of explicitly declaring, opening, and closing connections. This is useful since the OrientDB connection pooling class can manage multiple connections at the same time, and assign and reuse a previously opened connection, which means better performance:

```
ODatabaseDocument database = ODatabaseDocumentPool.global().
acquire("remote:localhost/minimalblog", "admin", "newadminpass");
```

For `Graph` databases:

```
OGraphDatabase database = OGraphDatabasePool.global().
acquire("remote:localhost/cities", "admin", "admin");
```

To close a connection, just use the `close()` method:

```
database.close();
```

If you are using a connection pool, the connections will not be closed but will become available to other threads. By default, the number of connections available in the pool is 20.

Executing SQL queries

One of the most useful APIs is certainly the one which allows you to execute SQL statements just like in the console. In this way, you can try the queries in the console and then put them in your Java code. To execute queries, you have to use the `OSQLSynchQuery` class:

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

The `ODocument` class is the main class that you will use to manipulate the records of an OrientDB database. It maps a single record (or just a document) of any class. When you write a query, you can also use prepared statements for both positional parameters and with named parameters:

```
OSQLSynchQuery<ODocument> queryPos = new OSQLSynchQuery<ODocument>("se
lect from posts where @rid=?");
```

Or:

```
OSQLSynchQuery<ODocument> queryPar = new OSQLSynchQuery<ODocument>("se
lect from posts where @rid=:recordId");
```

You will pass the parameter's value when you execute the queries.

> When you need to retrieve a single record, it is much faster to avoid the `where` clause. You can select a record directly by its RID as follows:
> ```
> select from #3:4
> ```

> In Java, the fastest way is to use the `load()` API:
> ```
> ODocument record = database.load(rid);
> ```
> where `database` is the current open connection, and `rid` is an instance of the `ORecordID` class.

To execute a query, you have to invoke the `execute()` method of the `OCommandRequest` class exposed by the database connection class. So if `database` is the name of your object connection, you should execute the following code:

```
List<ODocument> result = database.command(query).execute();
```

If you have positional parameters, you must list them as `execute()` parameters, as in the following case:

```
List<ODocument> result = database.command(queryPos).execute(new
ORecordId("#23:1"));
```

If you have named parameters, you must prepare `Map` to pass to the `execute()` method:

```
Map<String,Object> params = new HashMap<String,Object>();
params.put("recordId", new ORecordId("#23:1"));
List<ODocument> result = database.command(queryPar).execute(params);
```

You will have a list of the `ODocument` objects as a result, and you can cycle through it to extract the record information:

```
for (ODocument doc: result){
  String title = doc.field("title");
   String text = doc.field("text");
   System.out.printf("%s\n%s\n\n\n", title, text);
}
```

> A fluent API is also available, but it is generally much slower than SQL queries. OrientDB does not perform any optimization when executing these APIs and does not use indices to access data.

Executing SQL commands

You can, of course, execute updates and deletions as well. In these cases you have to use the `OCommandSQL` class instead of `OSQLSynchQuery`. So for example, the following code deletes a record belonging to the `Posts` class:

```
OCommandSQL deleteAPost = new OCommandSQL("delete from Posts where @
rid=#23:0");
int recordsDeleted = db.command(deleteAPost).execute();
```

Create, load, update, and delete a document

To work with documents via Java API, the right way is using the `ODocument` class and its methods. To create a new `Post` in our minimal blog example, perform the following steps:

1. Open a connection:

   ```
   ODatabaseDocumentTx db = new ODatabaseDocumentTx("remote:localho
   st/minimalblog").open("admin", "newadminpass");
   ```

2. Create a post:

   ```
   ODocument post = new ODocument("Posts");
   post.field( "title", "A post with ODocument" );
   post.field( "text", "is very simple" );
   post.field( "pubDate ", new Date() );
   post.save();
   ```

3. Close the connection:

   ```
   db.close();
   ```

As you can see, there is no need to specify the opened connection object to save the new record. This is completely transparent to the developer. In a similar way, you can update or delete a post. Supposing there is a valid opened connection, then we can update the `pubDate` field of a loaded post as follows:

```
ODocument post = database.load((ORecordId)OSQLHelper.
parseValue("#23:1", null));
post.field( "pubDate ", new Date() );
post.save();
```

To delete it:

```
post.delete();
```

In these examples, we used the `load()` method to load a document from the database by its RID, and the `OSQLHelper` class to convert a string into an RID. When you use a graph database, you must pay attention while you create vertices and edges because there are specific APIs to do these kinds of operations. We can create a vertex as follows:

```
ODocument city = database.createVertex("city");
```

We can create an edge between vertices as follows:

```
ODocument edge= database.createEdge( cityOne, cityTwo);
```

However, note that edges and vertices are always instances of the `ODocument` class. To remove edges and vertices don't use `the ODocument.delete()` method, as this could generate inconsistency in the graph. Use the `removeEdge()` and `removeVertex()` functions of the database class instead.

Object database support

OrientDB uses the `Javassist` library to offer support for object-document mapping. This feature is not available for graph databases but only for document ones. This is a specific design choice made because the wrapping operations have performance costs, and the OrientDB team wants to keep the graph implementation as fast as possible. For our minimal blog example, we must create a package for the models, that is, the classes that will be mapping the ones on the db: one for the categories, one for the posts, and one for authors. When we need to use the object mapping inside our Java project, we need to declare the mapping models as follows:

```
//open a connection
OObjectDatabaseTx db = new OObjectDatabaseTx ("remote:localhost/
minimalblog").open("admin", "newadminpass");
//register the model classes
db.getEntityManager().registerEntityClasses("model");
```

Here, `model` is the package containing the mapping class. At this point we can execute queries, for example:

```
//load a category
OSQLSynchQuery<Categories> getCarCategory = new OSQLSynchQuery<Categor
ies>("select from categories where name=?");
List<Categories> resCategories = db.command(getCarCategory).
execute("cars");
```

This retrieves the `cars` category.

The following code retrieves the author `"john"`:

```
//load an author
OSQLSynchQuery<Authors> getAuthor = new
OSQLSynchQuery<Authors>("select from authors where name=?");
List<Authors> resAuthors = db.command(getAuthor).execute("john");
```

Now we can create a new post as follows:

```
john = resAuthors.get(0);
//get a proxied object instance
Posts post = db.newInstance(Posts.class);
  post.setTitle("Post created via object-document mapping");
  post.setText("This is a very important example");
  post.setPubDate(new Date());
  post.setCategories(resCategories);
  post.setAuthor(john);
  db.save( post );
```

And do not forget to close the connection:

```
db.close();
```

OrientDB supports part of the JPA 2.x specification; so you may want to use the JPA annotation to manage cascade deleting. For example, we could have in the `Authors` class containing a list of `Posts`. We want to ensure that when an author is deleted, their posts also should be deleted:

```
public class Authors {
...
  @OneToMany(orphanRemoval = true)
  //or @OneToMany(cascade = { CascadeType.REMOVE })
  List<Posts> posts;
...
}
```

Remember that OrientDB does not manage the reverse relationship so it is your responsibility to maintain the database's consistency. To work, this annotation example needs a `posts` field to be declared against the `Authors` database class, and each time a new post is created, its RID to be put in this field.

RESTful APIs

OrientDB has an embedded HTTP server and exposes a set of REST APIs. So it can be queried via HTTP protocol and JSON data. The REST protocol uses the main four HTTP methods: GET, POST, PUT, and DELETE. The returned data format is JSON and even JSONP could be used. By default, this feature is on. You can enable or disable it by changing the XML config file. In this file you can change the port as well. The OrientDB Studio uses this feature of this protocol to communicate with the server, so if you disable it, the Studio will not work. In the configurations file you will find a network section which contains the following two listeners:

- One for the binary protocol
- One for the HTTP protocol

In particular you will find:

```
<listener ip-address="0.0.0.0" port-range="2480-2490" protocol="http">
    <commands>
        <command pattern="GET|www GET|studio/ GET| GET|*.htm GET|*.html
GET|*.xml GET|*.jpeg GET|*.jpg GET|*.png GET|*.gif GET|*.js GET|*.
css GET|*.swf GET|*.ico GET|*.txt GET|*.otf GET|*.pjs GET|*.svg"
implementation="com.orientechnologies.orient.server.network.protocol.
http.command.get.OServerCommandGetStaticContent">
            <parameters>
                            <entry name="http.cache:*.htm *.html"
value="Cache-Control: no-cache, no-store, max-age=0, must-
revalidate\r\nPragma: no-cache"/>
                            <entry name="http.cache:default"
value="Cache-Control: max-age=120"/>
            </parameters>
        </command>
    </commands>
    <parameters>
        <parameter name="network.http.charset" value="utf-8"/>
    </parameters>
</listener>
```

You can specify a range of TCP ports. OrientDB will try to bound on the first available in this range. In the parameter section, you can specify the charset you want to use to transfer data. I suggest you to not change other parameters. On the OrientDB official wiki site there is a full explanation of the REST protocol implemented by OrientDB. You can find the details of it at `https://github.com/nuvolabase/orientdb/wiki/OrientDB-REST`.

Transactions

OrientDB supports ACID transaction. This means that inside a transaction, all operations succeeded are committed or all the transactions are rolled back. There is no support for nested transactions, so each database connection instance could have at the most one opened transaction at a time. Transactions are optimistic and managed in the client scope, no locks are created on the server. Since every record has a version number field (the `@version` implicit field) maintained by the database engine when the transaction is committed, OrientDB checks the value of the `@version` field of the modified document. If it finds some mismatch (that is, someone else in the meantime has changed the record), it throws an exception and the transaction will be rolled back. To manage transactions via Java API, there are the following three database methods:

- `begin()`: This method opens a new transaction. If there was an opened transaction, this one is rolled back.
- `commit()`: This method persists the changes. If something goes wrong, the transaction is rolled back.
- `rollback()`: This method rolls back the changes

Transactions within REST calls

You can also perform transactions via the REST protocol. This does not mean that you can execute more calls within a transaction. A REST call is always atomic and leaves the database in an consistent state. Basically, you can execute more than a single operation in just one request. To do so you can use the `batch` command. The endpoint is as follows:

```
http://<orientdb-server>:<port>/batch/<database>
```

The action is POST and the body payload is a JSON string which must be in the following form:

```
{ "transaction" : true,
    "operations" : [
            { "type" : "<cud>",
        "record" : {
                    ....
                }
            }
        ]
}
```

Here, `cud` is a character stating the kinds of operations to perform, which are as follows:

- **c**: It specifies the create operation
- **u**: It specifies the update operation
- **d**: It specifies the delete operation

And `record` contains the fields of the record. For updates and deletes you must specify the RID in the form of `"@rid":"#23:1"`.

Summary

In this chapter we have seen different ways to perform queries and other operations such as updates and deletions against data.

In the next chapter we will see some tips and suggestions to improve performance and to perform a fine tuning of an OrientDB server.

4
Performance Tuning

The database performance is the very first priority for the OrientDB team, and because of this they persistently improve the engine and make modifications and updates based on the feedbacks and needs of users. Because of this, there are many aspects you can consider when you want to tune your OrientDB server. This is because depending on how you use it, there are different interventions you have to do. Instead of making just a list of the available options, I think it's better to start with a general introduction to some basic concepts about how OrientDB uses the memory and how it manages the I/O.

Caching

OrientDB uses two caches: the level 1 cache acts at thread level, while the level 2 cache acts at the JVM level. The following schema describes these concepts:
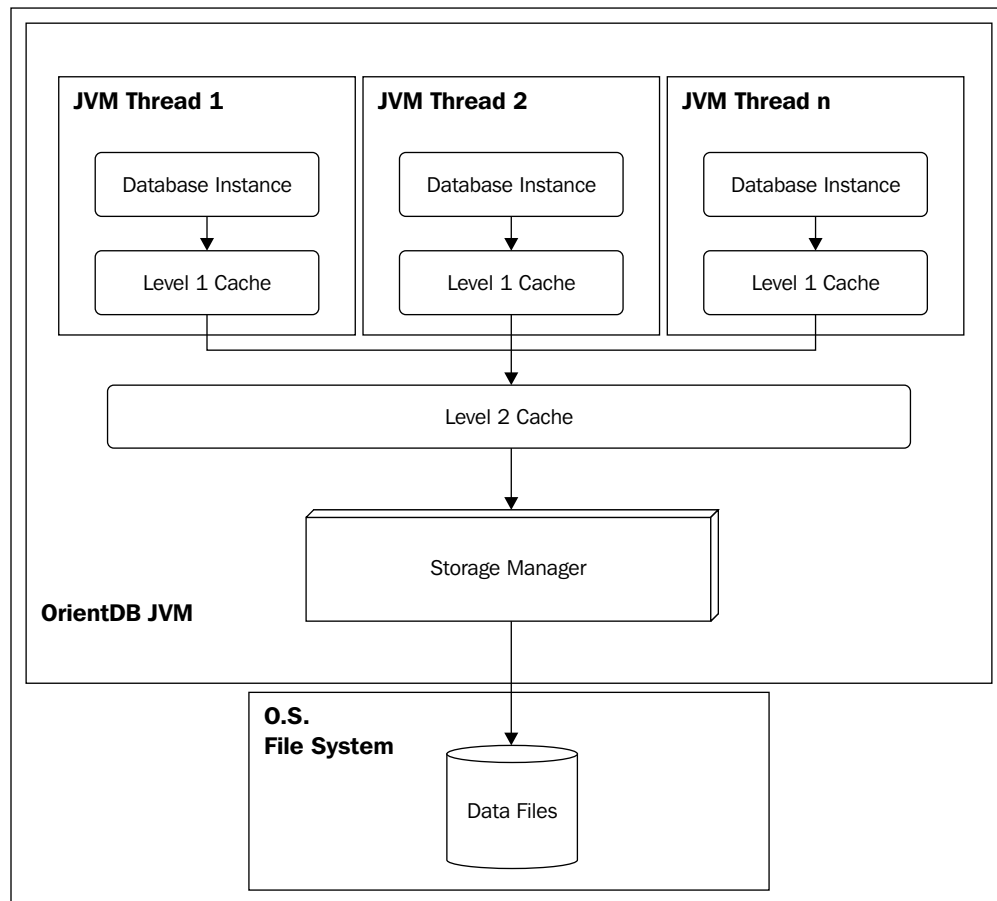
According to how your application works, you can act on the caches to improve performance. For example: if you do many reads you can leave all as it is, but if you are in a multi-threaded scenario and you perform many writes, you may consider disable the level 1 cache. Or, again, if you are in a multi-JVM scenario, you may consider disabling the level 2 cache also. Another consideration is the strategy used to read/write data files. OrientDB uses Java New I/O (NIO, JSR 51) API. This means that OrientDB although is written in Java, it can use the low-level I/O operations of modern operating systems.

Furthermore, OrientDB uses the memory mapped files, because of which access to the filesystem are minimized. Many aspects of OrientDB behavior are configurable through configuration parameters. These parameters can be set in the following ways:

- By using the command line at startup using Java options: you can use the `-D` option to define one or more values for configuration properties:
  ```
  java -Dcache.level1.size=100 .....
  ```

- By putting the configuration value into the XML config file: in the configuration XML, you can put your chosen value inside the `<properties>` section:
  ```
  <properties>
    <entry name="cache.level1.size" value="100" />
  </properties>
  ```

- At runtime, by calling the `OGlobalConfiguration` API:
  ```
  OGlobalConfiguration.CACHE_LEVEL1_SIZE.setValue(iValue)
  ```

- Via console using the `config` command:
  ```
  config set cache.level1.size 100
  ```

  You can see the current configuration as well:
  - At server startup via Java parameter:
    ```
    java -Denvironment.dumpCfgAtStartup=true ...
    ```

  - Via `OGlobalConfiguration` Java API:
    ```
    OGlobalConfiguration.dumpConfiguration(System.out);
    ```

  - Via the console `config` command:
    ```
    config
    ```

There are literally dozens of configuration parameters. In *Appendix* (available at `http://www.packtpub.com/sites/default/files/downloads/9956OS_Appendix.pdf`) you will find an exhaustive list of these parameters.

> When you use the Java API in your client application, many configuration parameters could also be set in the client. This means that you can have up to four caches, for example, two on the client JVM and two on the server.

In fact, when you use the OrientDB Java libraries in your application to connect to an OrientDB server, the scenario is similar to that shown in the following diagram:
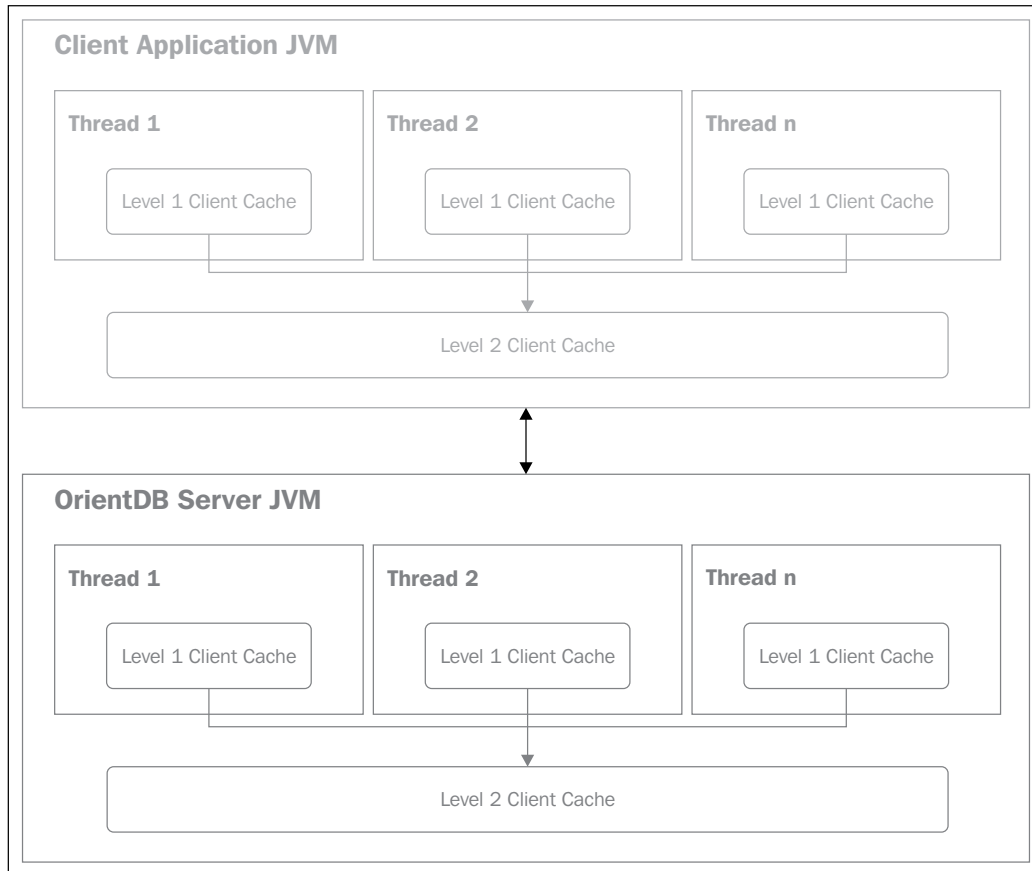
General Optimizations

Optimizations can be made at several levels and can involve different components from the JVM to the application code.

The JVM optimization

The OrientDB online documentation suggests some default parameters to pass to the JVM at startup. They are `-XX:+AggressiveOpts` and `-XX:CompileThreshold=200`. You can, of course, try other values and other options. You can find a complete list in the Oracle JVM official Java HotSpot VM Options page at `http://bit.ly/9PKM2D`.

Memory and cache

OrientDB uses more than one cache: one for each opened connection and one shared among the connections. Furthermore, OrientDB uses the memory mapped files to speed up the data access. This means that the trick here is to find the right balance among the caches, the memory mapping and the heap memory used by the JVM. To set up the memory that will be used by the memory mapped files, you can use the `file.mmap.maxMemory` configuration property. For example, on a 32-bit machine the maximum memory addressable is 4 GB, which means that you can set the heap value and the virtual memory so that their sum is 4 GB. Keep in mind, however, that if your server does not have enough memory, OrientDB can be swapped by O.S. and you can experience some performance degradation. In a 64-bit architecture, by default, OrientDB automatically set the `file.mmap.maxMemory` value as:

```
(maxOsMemory - maxProcessMemory) / 2
```

You can find this instruction in the `OGlobalConfiguration.java` file in the `autoConfig()` method. Furthermore, you can enable/disable level 1 cache, level 2 cache, or both. You can also set the number of records that will be stored in each level as follows:

- `cache.level1.size`: This sets the number of records to be stored in the level 1 caches (default -1, no limit)
- `cache.level2.size`: This sets the number of records to be stored in the level 2 cache (default -1, no limit)
- `cache.level1.enabled`: This is a boolean value, it enables/disables the level 1 cache (default, true)
- `cache.level2.enabled`: This is a boolean value, it enables/disables the level 2 cache (default, true)

Mapping files

OrientDB uses NIO to map data files in memory. However, you can change the way this mapping is performed. This is achieved by modifying the file access strategy.

- **Mode 0**: It uses the memory mapping for all the operations.
- **Mode 1 (default)**: It uses the memory mapping, but new reads are performed only if there is enough memory, otherwise the regular Java NIO file read/write is used.
- **Mode 2**: It uses the memory mapping only if the data has been previously loaded.

- **Mode 3**: It uses memory mapping until there is space available, then use regular JAVA NIO file read/write.
- **Mode 4**: It disables all the memory mapping techniques.

To set the strategy mode, you must use the `file.mmap.strategy` configuration property.

Connections

When you have to connect with a remote database you have some options to improve your application performance. You can use the connection pools, and define the timeout value to acquire a new connection. The pool has two attributes:

- `minPool`: It is the minimum number of opened connections
- `maxPool`: It is the maximum number of opened connections

When the first connection is requested to the pool, a number of connections corresponding to the `minPool` attribute are opened against the server. If a thread requires a new connection, the requests are satisfied by using a connection from the pool. If all the connections are busy, a new one is created until the value of `maxPool` is reached. Then the thread will wait, so that a connection is freed. Minimum and maximum connections are defined by using the `client.channel.minPool` (default value 1) and `client.channel.maxPool` (default value 5) properties. However, you can override these values in the client code by using the `setProperty()` method of the `connection` class. For example:

```
database = new ODatabaseDocumentTx("remote:localhost/demo");
database.setProperty("minPool", 10);
database.setProperty("maxPool", 50);
database.open("admin", "admin");
```

You can also change the connection timeout values. In fact, you may experience some problem, if there are network latencies or if some server-side operations require more time to be performed. Generally these kinds of problems are shown in the logfile with warnings:

```
WARNING: Connection re-acquired transparently after XXXms and Y
retries: no errors will be thrown at application level
```

You can try to change the `network.lockTimeout` and the `network.socketTimeout` values. The first one indicates the timeout in milliseconds to acquire a lock against a channel (default is `15000`), the second one indicates the TCP/IP socket timeout in milliseconds (default is `10000`). There are some other properties you can try to modify to resolve network issues. These are as follows:

- `network.socketBufferSize`: This is the TCP/IP socket buffer size in bytes (default 32 KB)
- `network.retry`: This indicates the number of retries a client should do to establish a connection against a server (default is 5)
- `network.retryDelay`: This indicates the number of milliseconds a client will wait before retrying to establish a new connection (default is 500)

Transactions

If your primary objective is the performance, avoid using transactions. However, if it is very important for you to have transactions to group operations, you can increase overall performance by disabling the transaction log. To do so just set the `tx.useLog` property to `false`.

> If you disable the transaction log, OrientDB cannot rollback operations in case JVM crashes.

Other transaction parameters are as follows:

- `tx.log.synch`: It is a Boolean value. If set, OrientDB executes a synch against the filesystem for each transaction log entry. This slows down the transactions, but provides reliability on non- reliable devices. Default value is false.
- `tx.commit.synch`: It is a Boolean value. If set, it performs a storage synch after a commit. Default value is true.

Massive insertions

If you want to do a massive insertion, there are some tricks to speed up the operation. First of all, do it via Java API. This is the fastest way to communicate with OrientDB. Second, instruct the server about your intention:

```
db.declareIntent( new OIntentMassiveInsert() );

//your code here....

db.declareIntent( null );
```

Here db is an opened database connection.

By declaring the OIntentMassiveInsert() intent, you are instructing OrientDB to reconfigure itself (that is, it applies a set of preconfigured configuration values) because a massive insert operation will begin. During the massive insert, avoid creating a new ODocument instance for each record to insert. On the contrary, just create an instance the first time, and then clean it using the reset() method:

```
ODocument doc = new ODocument();
for(int i=0; i< 9999999; i++){
  doc.reset(); //here you will reset the ODocument instance
  doc.setClassName("Author");
  doc.field("id", i);
  doc.field("name", "John");
  doc.save();
}
```

> This trick works only in a non-transactional context.

Finally, avoid transactions if you can. If you are using a graph database and you have to perform a massive insertion of vertices, you can still reset just one vertex:

```
ODocument doc = db.createVertex();
...
doc.reset();
...
```

Moreover, since a graph database caches the most used elements, you may disable this:

```
db.setRetainObjects(false);
```

Datafile fragmentation

Each time a record is updated or deleted, a hole is created in the datafiles structure. OrientDB tracks these holes and tries to reuse them. However, many updates and deletes can cause a fragmentation of datafiles, just like in a filesystem. To limit this problem, it is suggested to set the `oversize` attribute of the classes you create. The `oversize` attribute is used to allocate more space for records once they are created, so as to avoid defragmentation upon updates. The `oversize` attribute is a multiplying factor where 1.0 or 0 means no oversize. The default values are 0 for document, and 2 for vertices. OrientDB has a defrag algorithm that starts automatically when certain conditions are verified. You can set some of these conditions by using the following configuration parameter:

- `file.defrag.holeMaxDistance`: It defines the maximum distance in bytes between two holes that triggers the defrag procedure. The default is 32 KB, -1 means dynamic size. The dynamic size is computed in the `ODataLocal` class in the `getCloserHole()` method, as `Math.max(32768 * (int) (size / 10000000), 32768)`, where size is the current size of the file.

The profiler

OrientDB has an embedded profiler that you can use to analyze the behavior of the server. The configuration parameters that act on the profiler are as follows:

- `profiler.enabled`: This is a boolean value (enable/disable the profiler), the default value is `false`.
- `profiler.autoDump.interval`: It is the number of seconds between profiler dump. The default value is 0, which means no dump.
- `profiler.autoDump.reset`: This is a boolean value, reset the profile at every dump. The default is `true`.

The dump is a JSON string structured in sections. The first one is a huge collection of information gathered at runtime related to the configuration and resources used by each object in the database. The keys are structured as follows:

- `db.<db-name>`: They are database-related metrics
- `db.<db-name>.cache`: They are metrics about databases' caching
- `db.<db-name>.data`: They are metrics about databases' datafiles, mainly data holes
- `db.<db-name>.index`: They are metrics about databases' indexes
- `system.disk`: They are filesystem-related metrics

- `system.memory`: They are RAM-related metrics
- `system.config.cpus`: They are the number of the cores
- `process.network`: They are network metrics
- `process.runtime`: They provide process runtime information and metrics
- `server.connections.actives`: They are number of active connections

The second part of the dump is a collection of chronos. A chrono is a log of an operation, for example, a create operation, an update operation, and so on. Each chrono has the following attributes:

- `last`: It is the last time recorded
- `min`: It is the minimum time recorded
- `max`: It is the maximum time recorded
- `average`: It is the average time recorded
- `total`: It is the total time recorded
- `entries`: It is the number of times the specific metric has been recorded

Finally, there are sections about many counters.

Query tips

In the following paragraphs some useful information on how to optimize the queries execution is given.

The explain command

You can see how OrientDB accesses the data by using the `explain` command in the console. To use this command simply write `explain` followed by the `select` statement:

```
orientdb> explain select from Posts
```

A set of key-value pairs are returned. Keys mean the following:

- `resultType`: It is the type of the returned resultset. It can be `collection`, `document`, or `number`.
- `resultSize`: It is the number of records retrieved if the `resultType` is collection.
- `recordReads`: It is the number of records read from datafiles.
- `involvedIndexes`: They are the indices involved in the query.

- `indexReads`: It is the number of records read from the indices.
- `documentReads`: They are the documents read from the datafiles. This number could be different from `recordReads`, because in a scanned cluster there can be different kinds of records.
- `documentAnalyzedCompatibleClass`: They are the documents analyzed belonging to the class requested by the query. This number could be different from `documentReads`, because a cluster may contain several different classes.
- `elapsed`: This time is measured in nanoseconds, it is the time elapsed to execute the statement.

As you can see, OrientDB can use indices to speed up the reads.

Indexes

You can define indexes as we do in a relational database using the `create index` statement or via Java API using the `createIndex()` method of the `OClass` class:

```
create index <class>.<property> [unique|notunique|fulltext] [field
type]
```

Or for composite index (an index on more than one property):

```
create index <index_name> on <class> (<field1>,<field2>)
[unique|notunique|fulltext]
```

> If you create a composite index, OrientDB will use it also when in a `where` clause you don't specify a criteria against all the indexed fields. So you can avoid this to build an index for each field you use in the queries if you have already built a composite one. This is the case of a partial match search and further information about it can be found in the OrientDB wiki at `https://github.com/nuvolabase/orientdb/wiki/Indexes#partial-match-search`.

Generally, the indexes don't work with the `like` operator. If you want to perform the following query:

```
select from Authors where name like 'j%'
```

And you want use an index, you must define on the field `name` a `FULLTEXT` index.

`FULLTEXT` indices permit to index string fields. However keep in mind that indices slow down the insert, update, and delete operations.

Looking for @rid values

If you perform a query where you have the `@rid` value in the `where` condition, you can speed up your statement by omitting the `where` keyword.

For example, instead of:

```
select from posts where @rid = #15:2
```

Execute the following statement:

```
select from #15:2
```

If you have a set of RIDs:

```
select from [#15:2, #15:3, #15:3]
```

Summary

In this chapter we have seen some strategies that try to optimize both the OrientDB server installation and queries.

In the next chapter we are going to explore some advanced features of OrientDB, such as triggers, stored functions, hooks, and clustering.

5
Advanced Features

The information described in this chapter is related to the latest source code snapshot available on the GitHub repository at the time of writing this book. Since OrientDB is a very active project with many contributors, many features are added constantly and can make documentation obsolete and deprecated. The features listed in this chapter are as follows:

- **Embedded mode**: The OrientDB engine can be embedded in your own Java application. This means that you can bring all the power of a graph database directly into your application. This is the most powerful use of OrientDB, because to use its functionalities, you don't need to connect to a remote server, so you have no network latency or bottleneck.

- **Server-side code**: You can write your own functions in JavaScript and invoke them in SQL statements via REST or by using the Java API. In this way you can even put business logic into the server.

- **Hooks**: Hooks help you in extending the core of the engine, implementing new functionality depending on your needs.

- **Triggers**: As in a classical relational database, you can define functions that are automatically executed on specific events such as insert, update, and delete, not only at class level, but even at record level. This means that you can define different triggers for different records of the same class.

- **Clustering**: You can deploy a cluster of OrientDB nodes for load balancing purposes or to set up a high availability environment.

Embedded mode

You can embed the OrientDB core into your own Java project and use a local database without the need for a remote DB server. To use OrientDB in embedded mode you must include the following libraries in your project:

- `orient-commons-xxxx.jar`
- `orientdb-core-xxxx.jar`
- `orientdb-server-xxxx.jar`

You can use the OrientDB Java API to interact with the database and perform any kind of operation against it. You can access the database in the `./db` directory as follows:

```
final OGraphDatabase db = new OGraphDatabase ("local:./db") ;
if (!db.exists()) {
  //the db does not exist, let's create it
db.create();  //after the creation, the db is automatically opened
} else db.open("admin","admin");
```

Remember to close the connection when you finish using it.

In a thread you can retrieve an already opened connection as follows:

```
OGraphDatabase db = new OGraphDatabase   ((ODatabaseRecordTx)
ODatabaseRecordThreadLocal.INSTANCE.get());
```

Keep in mind, however, that connections are not thread safe, as stated in the official wiki page located at `https://github.com/orientechnologies/orientdb/wiki/Document-Database#multi-threading`.

Server-side code

One of the most powerful features of OrientDB is the possibility to write your own custom server-side code. You can imagine these pieces of code as being similar to that of stored procedures. The language used is JavaScript. In fact, OrientDB embeds the Rhino interpreter. To write your own server-side function you can use the OrientDB Studio which has an IDE, or you can use the console tool. In fact, the JavaScript functions are stored in the special `OFunction` class and you can insert, update, or delete functions like any other records belonging to a class.

Server-side function features

Server-side functions have a set of interesting characteristics. Some of them are as follows:

- They can be written in JavaScript or SQL. You can write your own functions in plain JavaScript or in SQL statements.
- They can be invoked via Java API, REST calls, and OrientDB Studio.
- They call each other. The JavaScript functions can call each other, and therefore you can build your server-side logic.
- They support recursion.
- They have automatic mapping of parameters by position and name. You can pass, of course, parameters to the functions and use both position and name notation.

Creating a function

To create a function, the most convenient way is to use the OrientDB Studio. After you are logged in as administrator, you have to click on the **Functions** tab on the toolbar:
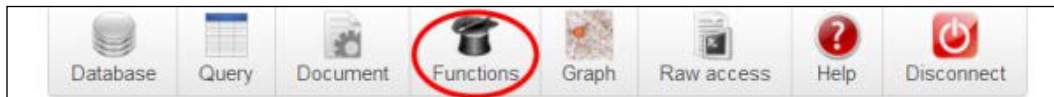
You should see the following IDE (you may be prompted to re-enter your admin credentials):
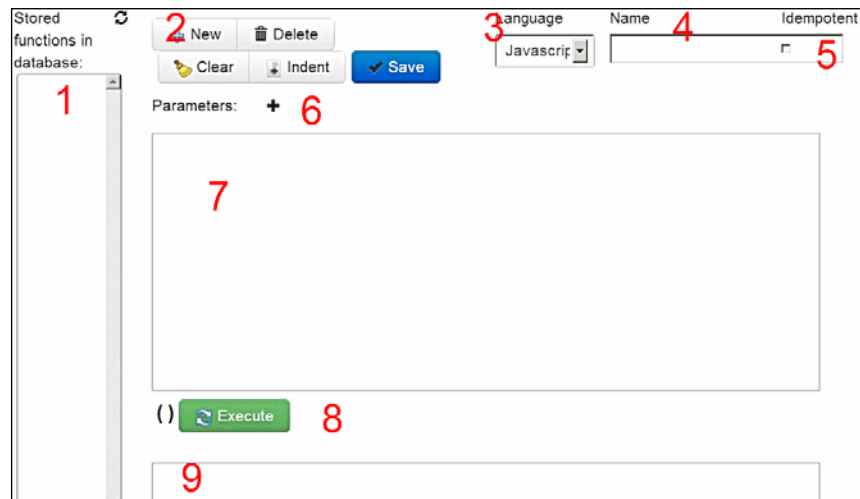
The IDE is split into several sections:

- **Section 1**: This is the list of the stored functions. Initially it is empty.
- **Section 2**: This is the toolbar you can use to create, update, and save your functions.
- **Section 3**: In this section, you can choose the functions' language. JavaScript and SQL are now supported, and more languages are coming soon.
- **Section 4**: This is the name of your stored function.
- **Section 5**: The idempotent checkbox specifies if the function is or is not idempotent; that is, whether the function has any side effects. This is a very important flag because if you want to call this function via HTTP, you check it as **Idempotent**. In fact, this is used to avoid calling non-idempotent functions using the HTTP GET method.
- **Section 6**: Here you can specify the function's parameters.
- **Section 7**: This is the body of the function.
- **Section 8**: This is the **Run** button. If the function accepts parameters, you can insert them between the brackets.
- **Section 9**: This is the output.

Let's try to write a simple function. Let's make a function that accepts two numbers and returns the sum. You have to specify the name of the function, the name of the two parameters by clicking twice on the **+** symbol next to the **Parameters** label, and write the body of the function, for example:

```
return a+b;
```

Here, a and b are the parameters. Then you need to enable the **Idempotent** checkbox and click on the **Save** button.
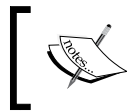
> Note that every time you make a change to a property of the function (the body, a parameter, or something else) you must save it. Otherwise the server always executes the previous version.

You should have something similar to the following screenshot:

Now specify the two parameter values, for example 1 and 2, and click on the **Execute** button. The result of the function call should appear:

```
{"result":[{"@type":"d","@version":0,"value":"12"}]}
```

The first thing we can note is that the result is in JSON format. The `@type` property specifies which kind of result we have; in this case `d` means `document`, the `@version` value is always `0`, the `value` property is the actual result of the function. But here, seemingly there is a problem, because 1 + 2 is not 12. This happens because all the parameters are treated like strings. So we need to modify our function body:

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

Click on the **Save** button, and then rerun the function. Now the result is a bit different:

```
{"result":[{"@type":"d","@version":0,"value":3.0,"@
fieldTypes":"value=d"}]}
```

Now `value` is correct, and we have the new field `@fieldTypes` that specifies that the `value` field is a number. If you open a console and perform a query on the `OFunction` class, you will see your function stored in it:

```
orientdb> select from OFunction

-+----+---------+----+------------------------+---------+---------
#|RID |language  |name|code                    |idempotent|parameters

-+----+---------+----+------------------------+---------+---------
0|#6:0|Javascript|sum |return Number(a)+Number(b);|true      |[2]

-+----+---------+----+------------------------+---------+---------
1 item(s) found. Query executed in 0.024 sec(s).
```

Let's make an example with the SQL language. Let's make a function that returns all the DB users:

1. Click on the **New** button.
2. Select **SQL** as the language.
3. Choose a name, for example `getUsers`.
4. Select the **Idempotent** checkbox.
5. Write the SQL statement as the function body:
   ```
   select from ouser
   ```
6. Save the function.
7. Run the function by clicking on the **Execute** button.

8. In the text area you should see the result in JSON format.

Usage

You can call a server-side function via Java API or HTTP Rest. In fact, every stored function is exposed by the embedded REST server.

Java API

The functions are available through the Function Library APIs. To use the `sum()` function declared previously, you can write some code similar to the following example:

```
OFunction sum = db.getMetadata().getFunctionLibrary().
getFunction("sum");
Number result = (Number) sum.execute(1, 2);
```

Here, `db` is an open database connection.

RESTful calls

Each stored function is automatically exposed by the embedded REST server. To call an idempotent function, you have to do an HTTP GET request to the URL:

```
http://<server-host>:<server-port>/function/<db-name>/<function-
name>/<parameters>
```

In our example it is:

```
http://localhost:2480/function/function_demo/sum/1/2
```

If the function is not idempotent you cannot use the GET action, but you can use POST.

> OrientDB implements the basic auth protocol. This means that if you want to do a REST call, you must supply the authorization header encoded in base64.

Special variables

Inside the JavaScript server-side functions some implicit variables are available so that you can easily interact with the database. They are as follows:

- `db`: The current document database instance
- `gdb`: The current graph database instance (if the current database is a graph db)

The preceding objects have the following useful methods:

- `query()`: This method performs a SQL query against the current database instance. You can pass the query and the parameters as well. For example:
  ```
  db.query ("select from ouser where name=?","admin");
  ```

- `begin()`, `commit()`, `rollback()`: These methods are used to perform transaction-related operations.

- `save()`: This method persists an object. You have to pass the object to persist. The object could have the special OrientDB fields such as `@class` to specify the class, `@version` to allow OrientDB to perform a check on concurrent modifications. For example:

```
db.save ({"@class":"author", name: "Jenny"});
```

  If you enable the e-mail plugin, you will also have a method that is used to send e-mails (this will be discussed later).

- `send()`: This method sends an e-mail using the specified parameters.

  If you plan to call your functions via REST, you have some other implicit variables.

- `request`: This is the HTTP `request` object. Its methods are as follows:
    - `getContent()`: This method returns the request content
    - `getUser()`: This method returns the current user
    - `getContentType()`: This method returns the content type specified into the request
    - `getHttpVersion()`: This method returns the request's HTTP version
    - `getHttpMethod()`: This method returns the HTTP request method
    - `getIfMatch()`: This method returns the request's `IF-MATCH` header
    - `isMultipart()`: This method checks whether the request has a multipart
    - `getArguments()`: This method returns the request's arguments passed in REST form. For example, `/1/2`
    - `getArgument(<position>)`: This method returns the request's argument by position, or null if not found
    - `getParameters()`: This method returns the request's parameters
    - `getParameter(<name>)`: This method returns the request's parameter by name, or null if not found
    - `hasParameters(<name>*)`: This method returns the number of parameters found between those passed
    - `getSessionId()`: This method returns the session ID
    - `getURL()`: This method returns the request's URL

- `response`: This is the HTTP response object. Its methods are as follows:
    - `setHeader(<header>)`: This method sets the response additional headers. To specify multiple headers you use the line breaks.

- ° `setContentType(<contentType>)`: This method sets the response content type. If it is null or you don't set it, it will be automatically guessed.

- ° `setCharacterSet(<characterSet>)`: This method sets the response character set.

- ° `writeStatus(<httpCode>, < reason>)`: This method sets the response status as the HTTP code and reason.

- ° `writeHeaders(<header>, <keepAlive>)`: This method writes the response headers specifying it to use `keepAlive` or not. `keepAlive` is optional, the default is `true`.

- ° `writeLine(<content>)`: Writes a line in the response. A line feed will be appended at the end.

- ° `writeContent(<content>)`: This method writes content directly.

- ° `writeRecords(<records>, <fetchPlan>)`: This method writes a list of records in the response, specifying a `fetchPlan` parameter to serialize nested records. The records are serialized in JSON format. `fetchPlan` is optional. Records are instances of `OIdentifiable`, and generally they are results of a `db.query()` call.

- ° `writeRecord(<record>, <fetchPlan>)`: This method writes a single record in the response, specifying a `fetchPlan` value to serialize nested records. The record is serialized in JSON format. `fetchPlan` is optional. `record` is an instance of `OIdentifiable`, and generally it is the result of a `db.query()` call.

- ° `send(<code>, <reason>,< contentType>, <content>, <additionalHeaders>, <keepAlive>)`: This method sends the complete HTTP response in one call. `additionalHeaders` and `keepAlive` are optional.

- ° `sendStream(<code>,< reason>, <contentType>, <content>,< size>)`: This method sends the complete HTTP response in one call specifying a stream as content.

- ° `flush()`: This method flushes the content of the response.

Finally there is the `util` object that has additional helper methods. At this moment only one method is available:

- ° `exists(<parameter>)`: This method returns `true` if the specified parameter has been passed to the function.

Hooks

Hooks are pieces of Java code that are "injected" at the lowest level of the db engine. They allow us to intercept the CRUD operation before or after they are performed and also allows us to perform additional operations on data before they are persisted, or before they are returned to the client. To write a hook, you have to write a Java class implementing the ORecordHook interface or just extend the ORecordHookAbstract class; both belong to the orientdb-core-xxx.jar library. The ORecordHookAbstract class has several methods and you can override only the ones you are interested in. Otherwise you can implement the ORecordHook interface and then write all the required methods. For example, you can mask the password field from the OUser class.

You may write a hook that intercepts the select from that class and drops the password field from the read records before they are returned to the client. Your code could be something like the following:

```
public class HidePassword extends ORecordHookAbstract {
  @Override
  public void onRecordAfterRead(ORecord<?> iRecord) {
    super.onRecordAfterRead(iRecord);
    if (iRecord instanceof ODocument){
      ODocument doc = (ODocument)iRecord;
      if (doc.getClassName()!=null && doc.getClassName()
.equalsIgnoreCase("OUser")){
        doc.removeField("password");
      }
    }
  }//onRecordAfterRead method
} //HidePassword class
```

To register your hook in the engine you must modify the class, compile its class, and put your .jar file in the ORIENTDB_HOME/lib directory. After this, you must modify the config file by inserting the following line:

```
<hook class="my.best.class.ever.HidePassword" position="REGULAR"/>
```

position indicates the order of the hooks' execution. It can be:

- FIRST: This hook will be the first one to be executed.
- EARLY: This hook will be executed just after the first.
- REGULAR: No order is specified; the hook will be called but there is no guarantee that it will be executed before or after another one.
- LATE: This hook will be executed after the REGULAR ones.
- LAST: This hook will be executed as the last one.

Triggers

Since Version 1.4.0, OrientDB supports triggers both on classes and records. This means that you can define triggers when an operation occurred against a specified class, or even against a specified record. Triggers can intercept the following eight events:

- `onBeforeCreate`
- `onAfterCreate`
- `onBeforeRead`
- `onAfterRead`
- `onBeforeUpdate`
- `onAfterUpdate`
- `onBeforeDelete`
- `onAfterDelete`

They can run a specified stored function if the events occur. Inside the databases there is a new abstract class named `OTriggered`; every class that extends it can use this feature. Let's take a look at an example. We want to log the date and time when a new record is inserted or updated, adding a field to the inserted/updated record. The first thing to do is to write a new stored function:

- **Name**: `logDate`
- **Body**:
  ```
  var now=new Date();
  var nowInString = now.getFullYear() + "/" + now.getMonth() + "/"
  + now.getDate() + " " + now.getHours() + ":" + now.getMinutes() +
  "."+now.getMilliseconds();
  doc.field("__log",nowInString);
  return "RECORD_CHANGED";
  ```

When a function is called in a trigger context, a new implicit variable is available. The `doc` variable is an instance of the `ODocument` class and maps the record that triggers the function. In our case we use the `doc` variable to add a new field, `__log`, into the document. Note also the returned string. This says to OrientDB that the trigger modified the document. Allowed values are as follows:

- `RECORD_NOT_CHANGED`
- `RECORD_CHANGED`

If no value is returned, `RECORD_NOT_CHANGED` is assumed.

> The returned value must be a string.

Next we have to create our new class:

```
create class Logged extends OTriggered
```

Now let's declare the trigger:

```
alter class Logged custom onBeforeCreate=logDate
```

> Don't include the function name between quotes.
> ```
> insert into Logged (text) values ("text")
> Inserted record 'Logged#10:0{text:text,__log:2013/2/30
> 22:23.604} v0' in 0,021000 sec(s).
> ```

Let's verify the inserted record:

```
select from Logged
---+---------+--------------------+--------------------
 #| RID     |text                |__log
---+---------+--------------------+--------------------
 0|    #10:0|text                |2013/2/30 22:23.604
---+---------+--------------------+--------------------
```

Gremlin support

Gremlin is a specialized language for working with graph database engines. OrientDB supports the Gremlin language. To use the Gremlin language you must use the Graphed Edition. In the ORIENTDB_HOME/bin directory you will find the gremlin.sh tool (.bat for Windows platforms). Remember to set the environment variable, ORIENTDB_HOME, before launching the shell tool. Once you have run gremlin.sh (you must run it from the ORIENTDB_HOME/bin directory), you can connect to an OrientDB server, or connect to a local database stored in the local filesystem:

```
orientdb-graphed-1.3.0\bin>gremlin

        \,,,/
        (o o)
-----oOOo-(_)-oOOo-----
gremlin>
```

To connect to the local example database, run the following code:

```
g = new OrientGraph("local:/orientdb-graphed-1.3.0/db/
tinkerpop","admin","admin");
```

To connect to a remote database you must use the following remote protocol:

```
g=new OrientGraph("remote:localhost/tinkerpop","admin","admin");
```

Once you have connected to a database, you will be able to perform any query or operation allowed by Gremlin.

Gephi

Gephi is a graphical tool that allows us to explore and manipulate a graph database in a visual manner. OrientDB supports the JSON streamed Gephi data format. To use Gephi with OrientDB, you must download Gephi from `http://gephi.org/users/ download` and install it. You must also run the OrientDB server. Gephi will use the OrientDB embedded HTTP REST server to retrieve the necessary data. To use Gephi, perform the following steps:

1. Start an OrientDB server instance.
2. Open Gephi.
3. Navigate and click on the **Tools | Plugins** menu.
4. Click on the **Available Plugins** tab.
5. Select the **Graph Streaming** plugin and install it (this is necessary only once).
6. Go to the **Overview** view (click on the **Overview** top-left button).
7. Click on the **Streaming** tab on the left.
8. Click on the green **+** button.
9. Insert as **Source URL**, the query you want to execute. For example, `http://localhost:2480/gephi/tinkerpop/sql/select%20from%20v/100`.
10. Select the JSON format.
11. Enable **Use Basic Authentication** and provide the right credentials (for example, `admin/admin`).

12. Click on the **OK** button.
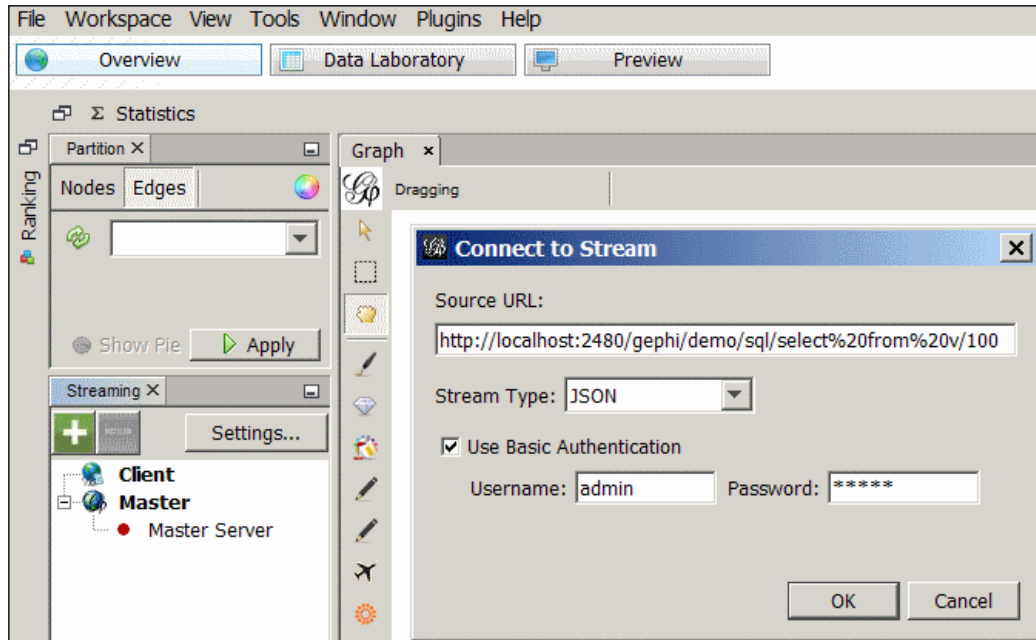
As you can see, the URL provided returns a set of information. You can explore it by trying to run the URL on a browser. The format of the URL is as follows:

```
http://<host>:<port>/gephi/<database>/<language>/<query>[/<limit>]
```

This can be broken down as follows:

- `host`: This is the host where OrientDB is installed.
- `port`: This is the configured port of the OrientDB instance.
- `database`: This is the database to query.
- `language`: This is the language used to write the query — it can be `sql` or `gremlin`.
- `query`: This is the query itself to run against the database.
- `limit`: This is an optional parameter to specify the maximum number of records to return. By default this is `20`. `-1` means no limit, so use it carefully.

Clustering

OrientDB can be deployed in a cluster architecture. It uses the Hazelcast open source project to manage the clustering. OrientDB currently supports the multi-master replication, in which each node of the cluster owns an exact replica of the database and can perform all kinds of operations against it. Of course, OrientDB manages the conflicts that could happen when more than one node wants to update the same information. Currently the clustering implementation is under heavy refactoring by the project team due to the implementation of the auto-sharing support.

> It is very important to know that replication works only for already created databases. Schema-related commands are not executed in a distributed environment. This means that you cannot create a new database in a node and see it in other nodes. Only data that is replicated across the cluster's nodes.

How it works

When OrientDB is configured to work in a cluster (this can be achieved through the configuration file), it tries to join to the cluster sending an IP multicast message in broadcast. If the cluster exists, the new node joins in, otherwise it becomes the first node of a new cluster. In the configuration file, there will be the cluster name and a password as well. The password is used to encrypt the broadcast message, so that the password itself does not cross the network. In this way, you may have several clusters on the same network, because only the cluster that can decrypt the message can reply to the message and allows the new node to join the cluster. Obviously all the nodes belonging to a cluster also have to know the same cluster name and the same password. OrientDB clients always know about the cluster layout, so when a new node joins the cluster, the new configuration is broadcasted to all active clients. In this way, clients know which other nodes can be queried, in case the node they are connected to becomes unavailable.

Replication

When a node has to do some non-idempotent operation against a database, it writes the operation on its own operation log, and then performs it. Because an OrientDB cluster is a multi-master cluster, the node communicates to other nodes that the operation is done. What happens next depends on the cluster configuration. Data propagation, in fact, can be made synchronously or asynchronously. The first one guarantees that databases are always consistent and synchronized, because the first node always waits for the answers of the other nodes before sending back the acknowledgement to the client. This, however, is expensive in terms of response time. The second mode, the asynchronous one, on the contrary is the fastest but does not guarantee the consistency of databases across the cluster because a few milliseconds can elapse between updates. Each operation is tracked by a unique identifier. This is important because if a node leaves the cluster, it loses the new updates eventually made by other nodes. When it rejoins in, it sends to other nodes belonging to the cluster, the identifier of the last operation performed to the other nodes belonging to the cluster before leaving the cluster. Other nodes check this identifier against their operation log and if it is not the same, they resend all newer operations to the node allowing its alignment.

Configuration and setup

The configuration of clusters and nodes takes place entirely in the configuration file of each node. OrientDB distribution comes with three configuration files in the `ORIENTDB_HOME/config` directory:

- `orientdb-server-config.xml`: This is the default config file for a standalone server
- `orientdb-dserver-config.xml`: This is the default config file for distributed deployment
- `orientdb-aserver-config.xml`: This is the default config file for distributed deployment with the auto-sharing feature

The `orientdb-dserver-config.xml` file is the one we use. The third one at the moment is experimental and under development. There are two other files to keep in mind when deploying and configuring a cluster:

- `hazelcast.xml`
- `default-distributed-db-config.json`

The first file to look is `orientdb-dserver-config.xml`. Here you'll find the configuration section for the Hazelcast plugin with its default parameters:

- `alias`: This allows you to specify an alias name for the node; if it is not specified, the node will be called `<ip-address>:<port>`.

- `enabled`: This value is false, and the Hazelcast plugin is disabled for this node.

- `configuration.hazelcast`: This is the external file containing additional parameters for Hazelcast.

- `configuration.db.default`: This is an external configuration file for specific replication parameters.

- `alignment.startup`: This specifies if the node should be aligned at startup.

- `alignment.timer`: This specifies a delay in milliseconds between automatic realignment.

- `conflict.resolver.impl`: This is the class used to manage the conflict that will occur. By default the `com.orientechnologies.orient.server.distributed.conflict.ODefaultReplicationConflictResolver` class is used. You can write your own class. In this case your class must implement the `OReplicationConflictResolver` interface.

The second file to look is the one defined in the `configuration.hazelcast` property; by default it is `${ORIENTDB_HOME}/config/hazelcast.xml`.

The relevant section in this file is the group one. In this section, you must specify the name of the cluster and the password used to encrypt the broadcast message sent to join the cluster. All the nodes belonging to the same cluster must have the same name and password. The third file is, by default, `${ORIENTDB_HOME}/config/default-distributed-db-config.json`. In this file it is possible to define the behavior of the node for each database, class, and other related stuffs. The default configuration file instructs the node to keep all the clusters synchronised except the internal one and the indices (this is because indices are rebuilt locally by each node). However, you can customize this behavior. One important feature for clusters is the possibility to share the second level cache. By default, this cache is shared between connection instances inside the same JVM, however; it is possible to disable this kind of cache and use a new kind of cache that is shared among all the connection instances belonging to the same cluster. To achieve this, you have to uncomment the related line in the `properties` section inside the `orientdb-dserver-config.xml` file.

Sending e-mails through OrientDB

Since Version 1.2.0, OrientDB ships with an e-mail plugin. This plugin allows you to send e-mails directly from the database. This is useful, for example, inside a hook to send an e-mail to the administrator if some conditions occur. To send e-mails you must enable the plugin in the configuration file(s). The default configuration file, `orientdb-server-config.xml`, comes with a default configuration. The plugin accepts a set of parameters:

*   `profile.<profile-name>.mail.smtp.host`: This is the SMTP server to be used
*   `profile.<profile-name>.mail.smtp.port`: This is the TCP port to be used to communicate with the SMTP server
*   `profile.<profile-name>.mail.smtp.auth`: This is set to true if the SMTP server requires authentication
*   `profile.<profile-name>.mail.smtp.starttls.enable`: This is set to true if the server supports TLS
*   `profile.<profile-name>.mail.smtp.user`: This specifies the username to be used to send the e-mails
*   `profile.<profile-name>.mail.smtp.password`: This is the user password
*   `profile.<profile-name>.mail.date.format`: This is the date format to be used

More than one `profile` parameter is supported. You can configure more profiles (for example, more than one SMTP server, user, password, and so on).

Usage

To send a message you must prepare a map with the required fields. They are as follows:

*   `profile`: This is the profile to be used to send the e-mail.
*   `to`: This is the recipient address. It can also be a list of comma-separated addresses.
*   `cc`: This is the carbon copy address. Similar to the `to` field, it can also be a list of comma-separated addresses.
*   `bcc`: This is the blind carbon copy address. Similar to the `to` field, it can also be a list of comma-separated addresses.
*   `subject`: This is the subject of the message.

- message: This is the body of the message. The content could be in HTML.
- attachments: This allows us to attach files. This is an array of filenames.

You can send e-mails either in stored JavaScript functions or via Java code. In JavaScript, the e-mail plugin defined and configured in the config file injects a new implicit variable into the function body, so you can simply write something as follows:

```
mail.send({
  profile: "default",
  to: "admin1@example.com,admin2@example.com",
  cc: "supervisor@example.com",
  subject: "Something happend!",
  message : "Alert! Something happend on OrientDB server!"
});
```

In Java, you must prepare the map and then use the OMailPlugin class (located in the orientb-server-xxxx.jar library):

```
OMailPlugin plugin = OServerMain.server().getPlugin("mail");
Map<String, Object> message = new HashMap<String, Object>();
message.put("profile", "default");
message.put("to",      "admin1@example.com,admin2@example.com");
message.put("cc",      "supervisor@example.com");
message.put("subject", "Something happend!");
message.put("message", "Alert! Something happend on OrientDB
server!");
plugin.send(message);
```

Summary

In this chapter we have seen some of the advanced features shipped with OrientDB. We have seen how to write server-side code, how to deploy a cluster, how to send e-mails, and how to visually display the data organized in a graphical representation.

In the next chapter, there is a quick reference to the most common and useful SQL statements, and also a Java API reference.

Furthermore, several configuration parameters available in OrientDB are listed and explained in *Appendix*. This appendix is available at http://www.packtpub.com/sites/default/files/downloads/9956OS_Appendix.pdf.

Also you can find a list of references at http://www.packtpub.com/sites/default/files/downloads/9956OS_OrientDB_1.5.0.pdf.

Index

mixed-mode 58-60
mode property 30

N

name attribute 60
name property 30
Native Java API
  about 74, 75
  connection, opening 75, 76
  document, creating 78, 79
  document, deleting 78, 79
  document, loading 78, 79
  document, updating 78, 79
  object database 79, 80
  SQL commands, executing 78
  SQL queries, executing 76-78
network 21
notnull attribute 60

O

OCommandRequest class 77
OCommandSQL class 78
ODatabaseDocumentTx class 75
ODocument class 76
ODocument.delete() method 79
ODocument object 77
ofunction.och file 25
OGraphDatabase class 75
OGraphEdge class 70
OGraphVertex class 70
OIdentity class 34
onCreate.fields property 35
onCreate.identityType property 35
one-to-many relationship 61, 62
one-to-one relationship 61, 62
OObjectDatabaseTx class 75
orids.0.ocl file 25
orids.och file 25
OrientDB
  classes 25, 26
  configuring 21
  console 22
  datafiles 24, 25
  emails, sending through 115
  emebedded relationships 61

  Gephi, using with 110, 111
  installing, as daemon 15
  installing, as service 15
  installing, from latest stable release 9
  referenced relationships 61
  security 28
  setting up, on Linux systems 15
  setting up, on Windows systems 16
  URL 9
orientdb-1.3.0.tar.gz file 9
OrientDB architecture
  about 19
  binary drivers 20
  binary protocol 20
  console 20
  core 20
  handlers 20
  Hazelcast 20
  Java API clients 20
  OrientDB Console Studio 20
  REST API 20
orientdb-aserver-config.xml file 113
OrientDB, configuring
  orientdb-dserver-config.xml 22
  orientdb-server-config.xml 21
OrientDB Console Studio 20
orientdb-dserver-config.xml
    file 21, 22, 32, 113
orientdb-graphed-1.3.0.tar.gz file 9
orientdb-odm
  URL 74
OrientDB-PHP
  URL 74
OrientDB Server
  about 7
  Enterprise Edition 9
  Graphed Edition 8
  Key/Value Edition 8
  Standard Edition 7
orientdb-server-config.xml file
  about 21, 32, 113
  handlers 21
  network 21
  properties 22
  storages 21
  users 22
orientdb-server-log.properties parameter 22

Thank you for buying
Getting Started with OrientDB

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.
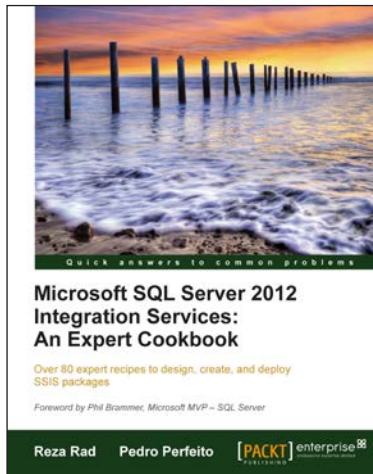
Microsoft SQL Server 2012 Integration Services: An Expert Cookbook

ISBN: 978-1-84968-524-5 Paperback: 564 pages

Over 80 expert recipes to design, create, and deploy SSIS packages

1. Full of illustrations, diagrams, and tips with clear step-by-step instructions and real time examples

2. Master all transformations in SSIS and their usages with real-world scenarios

3. Learn to make SSIS packages re-startable and robust; and work with transactions

4. Get hold of data cleansing and fuzzy operations in SSIS
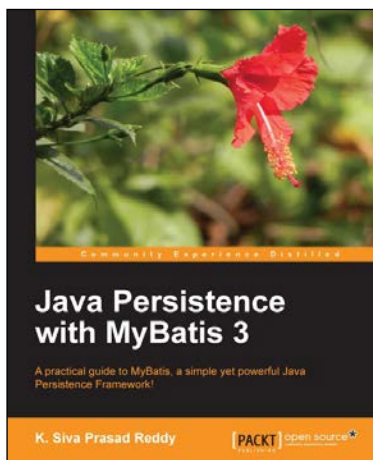
Java Persistence with MyBatis 3

ISBN: 978-1-78216-680-1 Paperback: 132 pages

A practical guide to MyBatis, a simple yet powerful Java Persistence Framework!

1. Detailed instructions on how to use MyBatis with XML and Annotation-based SQL Mappers

2. An in-depth discussion on how to map complex SQL query results such as One-To-Many and Many-To-Many using MyBatis ResultMaps

3. Step-by-step instructions on how to integrate MyBatis with a Spring framework

Please check **www.PacktPub.com** for information on our titles
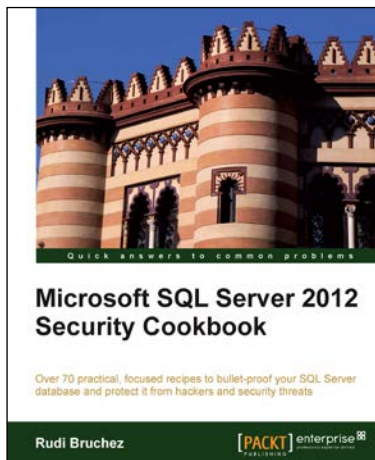
Microsoft SQL Server 2012 Security Cookbook

ISBN: 978-1-84968-588-7   Paperback: 322 pages

Over 70 practical, focused recipes to bullet-proof your SQL Server database and protect it from hackers and security threats

1. Practical, focused recipes for securing your SQL Server database

2. Master the latest techniques for data and code encryption, user authentication and authorization, protection against brute force attacks, denial-of-service attacks, and SQL Injection, and more

3. A learn-by-example recipe-based approach that focuses on key concepts to provide the foundation to solve real world problems
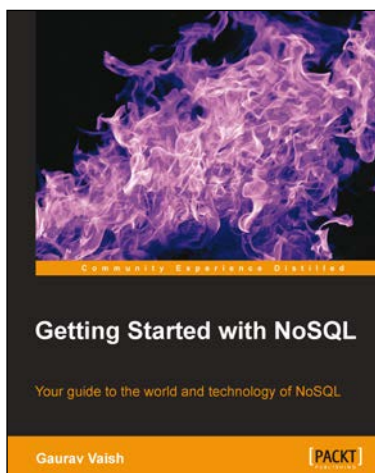
Getting Started with NoSQL

ISBN: 978-1-84969-498-8   Paperback: 142 pages

Your guide to the world and technology of NoSQL

1. First hand, detailed information about NoSQL technology

2. Learn the differences between NoSQL and RDBMS and where each is useful

3. Understand the various data models for NoSQL

4. Compare and contrast some of the popular NoSQL databases on the market

Please check **www.PacktPub.com** for information on our titles