Appendix

In this appendix, many additional pieces of information are grouped together from a quick reference to the most used commands, SQL and Java, to an overview of the several configuration parameters available.

Extended SQL quick reference

In this section, there is a quick reference to the most common SQL commands. For each of them the syntax and a brief description is provided if needed.

Schema manipulation

Here, the basic commands to manipulate the database schema are presented. In the first part there are commands to create elements such as classes, properties, indices, and so on. In the second part, there are commands used to alter the characteristics of the already created elements. Finally, in the third part, there are specific commands to drop and remove the existing resources.

Create commands

The following commands allow us to create elements and resources both on a remote server and in a local filesystem.

The create database command

This command is used to create a new database using the supplied parameters. It can be used to create a new database against a remote OrientDB server or to create a new one in a local filesystem:

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

Where:

- `database-url`: This is the URL of the database to be created in the `"<mode>:<path>"` format. In our case, this is `"remote:localhost/<database name>"`.
- `user`: This is the root username.
- `password`: This is the root password.
- `storage-type`: This indicates whether 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`: This is the type of the database. The possible values are `document` or `graph`. By default it is `document`.

The create class command

This command is used to create a new class in a database. You must be connected with a database and have the right credentials to perform this command:

```
CREATE CLASS <class> [EXTENDS <super-class>] [ABSTRACT|CLUSTER <cluster-id>*]
```

Where:

- `class`: This is the name of the class that will be created
- `super-class`: This is the optional class name to extend
- `cluster-id`: This may be one or a list (comma-separated) of cluster IDs to be used to store the class's records

If you don't specify at least one cluster, a new one with the same name of the class will be created. If a cluster with the same name of the class already exists, the default cluster will be used. Abstract classes belong to a dummy cluster with ID -1. If you extend the `ORestricted` class, your new class will have the record access security level feature. If you extend `OGraphVertex` (available only in graph databases), the documents belonging to this new class will be used as vertices. The `OGraphVertex` class can also be indicated as `V`. If you extend `OGraphEdge` (available only in graph databases), the documents belonging to this new class can be used as edges. The `OGraphEdge` class can also be indicated as `E`.

The create property command

The `Create Property` command creates a property against a class schema:

```
CREATE PROPERTY <class>.<property> <type> [<contained-type>|<linked-class>]
```

Where:

- `class`: This is the class name
- `property`: This is the name of the new property
- `type`: This is the data type of the new property
- `contained-type`: This is the contained type of the values (if the property will be a container)
- `linked-class`: This is the name of the contained class (if the property will be a container of links to other classes)

The `type` parameter can be:

- boolean
- integer
- short
- long
- float
- double
- date
- string
- binary
- embedded
- link
- byte
- Containers:
    - `embeddedlist`
    - `embeddedset`
    - `embeddedmap`
    - `linklist`
    - `linkset`
    - `linkmap`

`Contained-type` can be:

- boolean
- integer

- short
- long
- float
- double
- date
- string
- binary
- embedded
- link
- byte

The create datasegment command

This command is used to create new ODA and ODH files. The data segment files are where the records are physically stored. You can assign clusters to them:

```
create datasegment <datasegment-name> <datasegment-location>
```

Where:

- `datasegment-name`: This is the name of the new data segment.
- `datasegment-location`: Since each data segment corresponds to two files, you can choose the path to store them. If omitted, the database path is used.

The create cluster command

This command is used to create a new cluster on the filesystem. Each cluster has two files: an OCL file and an OCH file:

```
CREATE CLUSTER <name> <type> [<data-segment>|default] [<path>|default]
[<position>|append]
```

Where:

- `name`: This is the name of the cluster.
- `type`: This can be `PHYSICAL` or `MEMORY`. In a `PHYSICAL` database, you can create an in-memory cluster which is faster than the physical ones, but volatile.
- `data-segment`: This can be a data segment or default. To create a new data segment file, see `create datasegment`.

- `path`: Since each cluster corresponds to two files, you can choose their path. If omitted, the database path is used.
- `position`: This specifies a cluster ID to be replaced inside the specified data segment, or default to append the new cluster.

The create index command

There are two kinds of indices:

- An automatic one, that is, a typical index bound to a property's class which can be used to speed up queries or to create constraints like uniqueness.
- A manual one, which can be used to create manual indices or dictionaries to store key-values pairs. By default, any new OrientDB database has one such kind of index named `Dictionary`.

To create an index:

```
CREATE INDEX <name>|<class>.<property> [ON <class-name> (prop-names)]
<type> [<key-type>]
```

Where:

- `name`: This is the index name. It can be in the `<class>.<property>` form to create an index against a `class` property or any string to create a dictionary.
- `class-name`: The `ON` notation is a way to declare an index bound to a class property. If you use the `ON` notation, you can give a name to the index.
- `prop-names`: This can be one or a list of (comma-separated) properties on which the index is to be created.
- `type`: This can be `UNIQUE`, `NOTUNIQUE`, `FULLTEXT`, or `DICTIONARY`. If you want to perform queries on string field using the `like` operator and want to use an index, you have to declare it as `FULLTEXT`. Otherwise, it will not be used. `DICTIONARY` is similar to `UNIQUE`, but in case the key already exists, the old record is replaced by the new one.
- `key-type`: This is the data type of the field to be indexed. In the case of a composite index, this can be a comma-separated list of field types.

Examples:

Create a unique index for the `username` property of the `user` class:

```
create index user.username unique
```

Create a unique index against the `username` property of the `user` class using the `ON` notation:

```
create index idx_username on user (username) unique
```

Create a dictionary to store key-values pairs:

```
create index my_dictionary dictionary
```

The create vertex command

This command is used to create a new vertex in a graph database. A vertex is a special document that belongs to the `OGraphVertex` class or its derived class:

```
CREATE VERTEX [<class>] [CLUSTER <cluster>] [SET <field> =
<expression>[,]*]
```

Where:

- `class`: This is the vertex class. By default this is `OGraphVertex`, but you can specify any class that extends it.
- `cluster`: This is the optional cluster name into which we need to store the new vertex.
- `field and expression`: These are one or a list of (comma-separated) properties and their values are to be set during creation time.

Create edge

This command is used to create an edge between two vertices or between two sets of vertices. In this case you can specify two sets of RIDs or two queries:

```
CREATE EDGE [<class>] [CLUSTER <cluster>] FROM <rid>|(<query>)|[<rid>]*
TO <rid>|(<query>)|[<rid>]* [SET <field> = <expression>[,]*]
```

Alter

The following commands allow us to alter elements and resources created previously.

Alter database

This command is used to alter an attribute of the current database:

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

Where:

- `attribute-name`: This is the attribute to alter
- `attribute-value`: This is the new value to be assigned

There is only a space character between the attribute name and its value.

The attributes list is as follows:

- `TYPE`: This changes the database from document to graph. The only allowed value is `GRAPH`
- `STATUS`: This changes the status of the database, it can be `OPEN`, `CLOSED`, or `IMPORTING`
- `DEFAULTCLUSTERID`: This is used for setting any other previously created cluster; by default it's `2`
- `DATEFORMAT`: This is the default date format; by default is `yyyy-mm-dd`
- `DATETIMEFORMAT`: This is the default date and time format; by default it is `yyyy-mm-dd HH:mm:ss`
- `TIMEZONE`: The default value is `UTC`
- `LOCALECOUNTRY`: The default value is the system locale country
- `LOCALELANGUAGE`: The default is the system locale language
- `CHARSET`: The default value is `UTF-8`
- `CUSTOM`: We can define our own attribute key, in this case the attribute value becomes a string in the form `key=value`, where `key` is the name of your custom attribute

The alter class command

This command allows us to alter an attribute of a class belonging to the current database:

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

Where:

- `class`: This is the class to be altered
- `attribute-name`: This is the attribute to be altered
- `attribute-value`: This is the new value to be assigned

The complete list of attributes is as follows:

- `NAME`: This is the class name; you can even modify the class name.
- `SHORTNAME`: This is a short name (an alias) for the class. Assign `NULL` to remove it.

- `SUPERCLASS`: This is the superclass name to be assigned. We can declare a superclass after the class creation. Assign `NULL` to remove it.
- `OVERSIZE`: This is the `OVERSIZE` factor. This is a multiply factor to apply to record size so as to reserve more space for each record belonging to the class. This is useful to avoid defragmentation upon updates. `0` or `1` means no oversize factor, the default is `2`.
- `STRICTMODE`: Its value can be `true`/`false`. default is `false`, which means that the class is not schema-full.
- `ADDCLUSTER`: This adds a cluster to the class.
- `REMOVECLUSTER`: This removes a cluster from a class (the cluster will not be deleted).
- `CUSTOM`: This allows us to set custom class-level properties in the form of `<name>=<value>` (without spaces between them). This is useful to model, for example, `UML stereotypes`. Example: `ALTER CLASS Person CUSTOM stereotype=customer`.

The alter cluster command

This command alters an attribute of a cluster belonging to the current database:

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

Where:

- `cluster-name`, `cluster-id`: This is the cluster to be altered
- `attribute-name`: This is the attribute to be altered
- `attribute-value`: This is the new value to be assigned

The complete list of attributes is as follows:

- `NAME`: This is the name of the new cluster
- `DATASEGMENT`: This is the new data segment to be used for this cluster

The alter property command

This command alters an attribute of a class's property:

```
ALTER PROPERTY <class>.<property> <attribute-name> <attribute-value>
```

Where:

- `class`: This is the class name
- `property`: This is the property name
- `attribute-name`: This is the attribute to be altered
- `attribute-value`: This is the new value to be assigned

The complete list of attributes is as follows:

- `LINKEDCLASS`: This is the linked class name (can be used only for a link property type). It accepts a string as value. NULL to remove it.
- `READONLY`: This is a Boolean value. If `true`, the property can only be read. It can be written only during the record creation.
- `MIN`: This is the minimum value as constraint. `NULL` is used to remove it. The meaning of "minimum" depends on the property data type:
    - **String**: Minimum length
    - **Number**: Minimum value
    - **Date and Time**: Minimum time in milliseconds
    - **Binary**: Minimum size of the byte array (the binary data is internally represented by an array of bytes)
    - **List, Set, Collections, Maps**: Minimum number of items
- `MANDATORY`: This is a Boolean value; its value is true if the property is mandatory.
- `MAX`: This is the maximum value constraint; its value is `NULL` to remove the constraint (the meaning of "maximum" depends on the property data type).
- `NOTNULL`: This is a Boolean value. If set to true, the property can't be null.
- `REGEXP`: This is a regular expression specified as constraint. `NULL` to remove it.
- `TYPE`: This is a new data type among those supported. `NULL` to remove it. Note that if a property already has a `TYPE` value, OrientDB will try to perform a casting to the new type. Some conversions are not allowed and the `ALTER` command will fail.
- `CUSTOM`: This sets custom properties. Its syntax is `"<name>=<value>"` (without spaces between them). This is useful to model, for example, UML stereotypes. Example: `ALTER PROPERTY Person.avatar CUSTOM stereotype=icon`.

Drop

The following commands allow us to drop elements and resources both on a remote server and in a local filesystem.

The drop database command

This command drops an existing database (all the data will be lost):

```
DROP DATABASE [<database-url> <server-username> <server-userpassword>]
```

Where:

- `database-url`: This is the URL of the database to be created in the `"<mode>:<path>"` format. In our case, it is `"remote:localhost/<database name>"`.
- `server-username`: This specifies the username, generally root, which has the privilege to drop the database.
- `server-userpassword`: This specifies the username and password.

Without parameters, the current database is dropped, otherwise the specified one is dropped.

The drop class command

This command removes a class' definition from the database schema:

```
DROP CLASS <class>
```

Where:

- `class`: This is the class to be dropped

However, in this case, OrientDB will raise an exception that some subclasses still exist and it will not remove the class.

The drop cluster command

This command deletes a cluster from the database (all the data belonging to the cluster will also be deleted):

```
drop cluster <cluster>
```

Where:

- `cluster`: This is the cluster to be dropped

A cluster can be dropped if and only if none of the classes use it.

The drop index command

This command is used to remove an index from the database. If the index is built on a class's property, the data is not affected. If the index is a manual index, the data stored in it will be lost:

```
DROP INDEX <index-name>|<class>.<property>
```

Where:

- `index-name`: This is the name of the index to be dropped
- `class.property`: If the index has no name, you can still drop it by using the class and property name on which it is built

The drop property command

This command is used to remove a property definition from a class schema:

```
DROP PROPERTY <class>.<property>
```

Where:

- `class.property`: This specifies the property to be dropped

Note that this command only changes the schema information related to the specified class. The values of the property are not deleted from the existing records.

Data inquiry

The following commands allow us to query the data stored in a database.

The select command

This command is used to return records according to the specified criteria:

```
SELECT [<projections>] from <target> [WHERE <conditions>] [ORDER BY
<order-fields> [ASC|DESC] [SKIP <skip-records>] [LIMIT <max-records>]
```

Where:

- `projections`: This specifies what you want to extract from the query. They are optional; if no projection is given, all fields are returned.
- `target`: This can be a class, a cluster (use the prefix `cluster:`), an index (use the prefix `index:`), an RID, or a set of RIDs (include them between square brackets and separate them with a comma).
- `conditions`: These are the optional conditions used to filter the result set.

- `order-fields`: This is the optional list of fields used as sorting criteria.
- `skip-records`: This specifies the number of records to be skipped and is useful for pagination.
- `max-records`: This specifies the maximum number of records to return and is useful for pagination.

Projections

You can specify the fields returned by the query, or you can use functions or aggregate functions that will be applied to the specified fields. You can also navigate through the linked or embedded objects via the dot notation. If you have some experience with languages, for example **Hibernate Query Language** (**HQL**) you will be fine with this. Keep in mind that if you declare a projection in your query, the returned records do not physically exist on any data file (they are built in memory and returned). Because of this, these records do not have RIDs, and OrientDB assigns them fake RIDs. To know the effective RID of the record that has generated a row in the result set, you can use the `@RID` meta field. The complete list of meta fields is as follows:

- `@this`: This is the record itself
- `@rid`: This is the RID of the record
- `@rid_id`: This is the cluster part of the RID
- `@rid_pos`: This is the record part of the RID
- `@version`: This is the version number of the record
- `@version_time`: This is the timestamp of the current version of the record (available only in a distributed environment)
- `@version_mac`: This is the MAC address associated to the current version of the record (available only in a distributed environment)
- `@class`: This is the class of the record
- `@type`: This can be document or byte for binary data
- `@size`: This is the size of the record in bytes
- `@fields`: This is the collections of the record's fields
- `@raw`: This is the internal raw representation of the record

Conditions

Conditions are the criteria used to filter records. Similar to any RDBMS, the conditions are specified after the `where` keyword. You can find a complete list of operators in the wiki at:

```
https://github.com/nuvolabase/orientdb/wiki/SQL-Where
```

Since OrientDB is a very active project with many contributors, the list of available functions and operators grows constantly. There are two special operators which are very useful in a NoSQL context. They are `any()` and `all()`. In fact, since in a schema-less class you may not know at design time which fields it has, you cannot write a query. In this case, you can use `any()` to indicate that at least one property has to meet the criteria or you can use `all()` to indicate that all properties have to meet the criteria. Following are some examples:

```
select from authors where any() like 'j%';
select from authors where all() is not null;
```

The LET keyword

The `LET` keyword allows us to declare context custom variables inside a `SELECT` statement. Context variables can be used in projections, conditions, and subqueries. They are destroyed once the statement is finished:

```
SELECT @rid, $city.name FROM Profile LET $city = address.city WHERE
$city.name like '%om%"'
```

In the preceding example, a `$city` variable is declared as `address.city`. Once declared, the new context variable can be used in conditions and also in projections. In fact, it is used to filter by city name and in the result set.

The traverse command

It is used to traverse the graph from a starting node:

```
TRAVERSE [*|<field>*] FROM <source> [WHILE <conditions>] [LIMIT <max-
records>]
```

Where:

- `field`: This specifies a field or a list of fields to follow to explore the graph. You can use the `any()` and the `all()` operators.
- `source`: This can be a class, a cluster (use the prefix `cluster:`), an index (use the prefix `index:`), an RID, a set of RIDs (include them between square brackets and separate them with a comma), a subquery similar to a `select` statement, or the `traverse` command.
- `conditions`: This specifies the optional conditions that must be true to continue traversing.
- `max-records`: This specifies the maximum number of records to be returned.

While executing a `traverse` command, some context variables are available to be used in the `while` condition. They are as follows:

- `$depth`: This specifies the current depth of nesting
- `$history`: This is a set of RIDs of all the records traversed to reach the current record
- `$path`: This is a string representation of the path used to reach the current record from the root of the traversing
- `$parent`: This is the parent's context (if any) and is useful in subqueries
- `$current`: This is the current record; to access the upper level record in nested queries, you can use `$parent.$current`

Examples:

```
traverse any() from #22:0 while $depth<3;
select @rid, $path from (traverse any() from #22:0 while $depth<3);
```

Data manipulation

By using the following statements you can manipulate the data stored in a database.

The insert statement

The insert statements allow us to insert data into the database. You can perform an insert statement against a class, a cluster, or an index:

```
INSERT INTO <class>|cluster:<cluster>|index:<index> [<different_cluster>]
[(fields) VALUES (<values>)]|SET <field>:<value>
```

Where:

- `class`: This is the class name, if the insert must be done against a class
- `cluster`: This is the cluster name, if the insert must be done in a cluster
- `index`: This is the manual index name (for example, dictionary), if the insert must be done in an index
- `different_cluster`: This is a cluster name if you want to perform an insert in a class, but using a different cluster from the default one

You can use two kinds of syntax. The first one is similar to standard SQL, where you have to specify the list of fields followed by the `VALUES` keyword and a list of values. The second one involves the use of the `SET` keyword which is similar to the `set` keyword in the standard SQL `update` command.

The update statement

Update statements have three types of syntax: the first one is similar to standard SQL, the second one is useful when you have to update a list or a collection, and the third one is for maps:

```
UPDATE <class>|cluster:<cluster-name>  SET|INCREMENT <field-name>=<field-value> [,<field-name>=<field-value> ]* [WHERE <conditions>] [LIMIT <max-records>]
```

Where:

- `class`: This is the class name
- `cluster-name`: This is the cluster name
- `SET`: This means that the fields will be updated with the specified values
- `INCREMENT`: This means that the fields will be incremented by the specified values (use negative values to decrement)
- `field-name`: This is the name of the property to be updated
- `field-value`: This is the new value (or the increment in case of `INCREMENT`)
- `conditions`: This is used to specify the criteria to be used to filter the records that are to be updated
- `max-records`: This specifies the maximum number of records to be updated

With collections and lists:

```
UPDATE <class>|cluster:<cluster-name> ADD|REMOVE <field-name>=<field-value> [,<field-name>=<field-value> ]* [WHERE <conditions>] [LIMIT <max-records>]
```

Note that `ADD` and `REMOVE` keywords are used instead of `SET` and `INCREMENT`.

With maps:

```
UPDATE <class>|cluster:<cluster-name> PUT|REMOVE <field-name>=<map-key>,<map-value> [,<field-name>=<map-key>,<map-value> ]* [WHERE <conditions>] [LIMIT <max-records>]
```

With maps, there is the `PUT` keyword instead of `ADD`. Furthermore, you can specify a key and a value pair. To remove an item from a map, you have to specify just the key.

The delete command

This command allows us to delete a record from a class, a cluster, or an index entry. The `delete` command supports transactions:

```
DELETE FROM <class>|cluster:<cluster-name>|index:<index> [WHERE
<conditions>]
```

Where:

- `class`: This is the class name
- `cluster-name`: This is the cluster name
- `index`: This is the index name
- `conditions`: This is the criteria to be used to select the records to be deleted

The Truncate statement

The `TRUNCATE` statement acts as a lower level to that of the `DELETE` one. The truncate operations cannot be rolled back.

The Truncate Class command

This command deletes all the records belonging to all clusters defined as part of the specified class:

```
TRUNCATE CLASS <class-name>
```

The Truncate Cluster command

This command deletes all the records belonging to the specified cluster:

```
TRUNCATE CLUSTER <cluster-name>
```

The Truncate Record command

This command is used to truncate a record or a list of records. It is useful when some records are corrupted and cannot be loaded or even deleted:

```
TRUNCATE RECORD <rid>|[<rid>,*]
```

The list of RIDs is a comma-separated list included between square brackets.

In the preceding paragraphs, we have seen some of the most common SQL commands available in OrientDB. In the next part, we will continue with quick references, but now we will focus on the Java API here: `http://www.packtpub.com/sites/default/files/downloads/9956OS_OrientDB_1.5.0.pdf`.