Volta Criteria Library user manual

Table Of Contents

1 References 3
2 Volta Criteria 3
3 Volta Criteria license 3
4 System Requirements 3
5 How to use Volta Criteria 4
5.1 Import libraries 4
5.2 Configuring Volta Criteria 4
5.3 Using the library for writing CQL statements 4
5.3.1 Construction of an INSERT statement 5
5.3.2 Construction of an UPDATE statement 7
5.3.3 Construction of a DELETE statement 8
5.3.4 Construction of a SELECT statement 9
5.4 Statements execution 13

1 References

[Ref.01] Volta Criteria distribution package (volta-criteria-x.y.z.* https://sourceforge.net/projects/voltacriteria/files/?source=navbar)

[Ref.02] Datastax official documentation relating to the latest version of java driver for Apache Cassandra (http://www.datastax.com/documentation/developer/java-driver/2.0/java-driver/whatsNew2.html)

2 Volta Criteria

Volta Criteria is a java library that allows you to create queries and CQL without coding (therefore you do not need to know the syntax) CQL. It is designed for those who want to integrate Cassandra database within their java application, simply by using objects.

This will make it more simplified writing queries, limiting as far as possible typing errors, otherwise found only at runtime, and requiring compliance with the constraints of language CQL at compile time.

The library does not require the connection to the database and the execution of the statements, in order not to limit the choice of java driver to use, for example, the official Datastax driver [Ref.02].

3 Volta Criteria license

The license applied to Volta Criteria is "Apache Software License 2.0" (http://www.apache.org/licenses/LICENSE-2.0.html).

4 System Requirements

To use Volta Criteria, a JRE (Java Runtime Environment) version 7 or greater (http://www.java.com) is required..

5 How to use Volta Criteria

The following are the steps needed to use the library Volta Criteria inside a standalone java project.

5.1 Import libraries

To use the library Volta Criteria you must import only the jar file volta-criteria-x.y.z.jar.

N.B. x.y.z indicates a generic version of the library Volta Criteria.

5.2 Configuring Volta Criteria

The use of the library Volta Criteria requires no specific configuration.

5.3 Using the library for writing CQL statements

To use the library Volta Criteria, simply import it into the java class where Volta Criteria methods are invoked. The reference class is `it.sogetel.voltacriteria.criteria.CriteriaFactory` whose static method *create* returns an object of type `it.sogetel.voltacriteria.criteria.generic.Criteria` containing the desired statement.

The create method has as input parameters the type of statement, represented by a static enum, and the name of the table on which to execute the statement.

The accepted values as type of statement are the following:

- CriteriaFactory.QueryType.*SELECT*;
- CriteriaFactory.QueryType.*INSERT*;
- CriteriaFactory.QueryType.*UPDATE*;
- CriteriaFactory.QueryType.*DELETE*.

For example, writing:

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.SELECT,
"news");

System.out.println(criteria.toString());
```

you will then have as output:

```
SELECT * FROM news;
```

You can refine the statement contained in the object type `it.sogetel.voltacriteria.criteria.generic.Criteria` through the following methods:

Method	Description
addAllowFiltering()	Adds the ALLOW FILTERING clause, where required
addAssignment(Parameter, Assignment)	Inserts fields and values to be assigned in the INSERT and UPDATE
addLimit(int)	Adds the LIMIT int clause, where required
addOption(Option)	Adds the specified option between TTL and TIMESTAMP
addOrder(Order)	Adds the ORDER BY clause
addParameter(Parameter)	Specifies in which columns to apply the DELETE
addProjection(Projection)	Adds a field to the list of projections of the SELECT
addRestriction(Criterion)	Adds a condition to the list of restrictions of the statement

Below are explained in detail the methods of construction of different types of statements.

N.B. The use of any methods not relevant to the type of statement is not inhibited, however, these will not be considered in the construction phase of the query.

5.3.1 Construction of an INSERT statement

To define a statement of insert data, you must use the methods `addAssignment` and `addOption` (optional).

Through `addAssignment` you can specify the individual columns with values of INSERT. The method takes as input a parameter of type `it.sogetel.voltacriteria.criteria.parameter.Parameter` containing the column name and a parameter of type `it.sogetel.voltacriteria.criteria.assignment.Assignment` containing the value for the column. So, writing:

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.IN
SERT, "news")

.addAssignment(Parameter.value("category"),Assignme
nt.value("sport"))

.addAssignment(Parameter.value("name"),
Assignment.value("noname"));


System.out.println(criteria.toString());
```

will result in output:

```
INSERT INTO news (category, name) VALUES ('sport',
'noname');
```

Invoking the method `addOption` you can add the clauses TTL and TIMESTAMP to the insert statement. You need to pass to the method a parameter of type `it.sogetel.voltacriteria.criteria.options.Options` in order to specify which of the two clauses to use and the relative value. So if you add the call to this method to the code example above:

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.IN
SERT, "news")

.addAssignment(Parameter.value("category"),
Assignment.value("sport"))
```

```
.addAssignment(Parameter.value("name"),
Assignment.value("noname"))
.addOption(Options.ttl(1234567890));

System.out.println(criteria.toString());
```

the output will be:

```
INSERT INTO news (category, name)
VALUES ('sport', 'noname')
USING TTL 1234567890;
```

5.3.2 Construction of an UPDATE statement

Similarly to the insert statement, also in the update statement you must use the method `addAssignment` (see the paragraph of INSERT) to specify the columns and the values to be assigned.

To perform an update statement, you must specify the tuples on which the update itself works, inserting conditions to the WHERE clause, using the method `addRestriction` that takes as input an object of type `it.sogetel.cassandra.criteria.restrictions.Restrictions`.
In the UPDATE statement is recognized for purposes of construction of the string, the only method `eq(String propertyName, Object value)`.
The statement:

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.UP
DATE, "news")
.addOption(Options.ttl(1234567890))
.addAssignment(Parameter.value("category"),
Assignment.value("sport"))
.addAssignment(Parameter.value("name"),
Assignment.value("noname"))
.addRestriction(Restrictions.eq("author", "none"));
```

thus obtaining such statement:

```
UPDATE news
USING TTL 1234567890
SET category = 'sport', name = 'noname'
WHERE author='none';
```

5.3.3 Construction of a DELETE statement

Simply call the create method of the library by passing the parameter `CriteriaFactory.QueryType.DELETE` and the name of the affected table to get the delete statement of the entire contents of a table. The call:

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.DE
LETE, "news");
```

return the statement:

```
DELETE  FROM news;
```

Again, you can select the tuples on which to act through the method addRestriction (see the paragraph of UPDATE):

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.DE
LETE, "news")
.addRestriction(Restrictions.eq("category",
"economy"))
.addRestriction(Restrictions.ltMaxTimeUUID("date",
"2014-01-01 00:05+0000"));
```

obtaining:

```
DELETE  FROM news
WHERE category='economy'
AND date < maxtimeuuid('2014-01-01 00:05+0000');
```

You can also specify which columns to clear through `addParameter` method that takes as input an object of type `it.sogetel.voltacriteria.criteria.parameter.Parameter` by which indicate the names of the individual columns. Thus changing the code example above in this way:

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.DE
LETE, "news")
.addParameter(Parameter.value("name"))
.addRestriction(Restrictions.eq("category",
"economy")
.addRestriction(Restrictions.ltMaxTimeUUID("date",
"2014-01-01 00:05+0000")));
```

you obtain the statement:

```
DELETE name  FROM news
WHERE category='economy'
AND date < maxtimeuuid('2014-01-01 00:05+0000');
```

5.3.4 Construction of a SELECT statement

As seen above, a simple call of the create method:

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.SE
LECT, "news");
```

correspond to the query:

```
SELECT *
```

```
FROM news;
```

If you want to select individual fields that the query should return, you can call the method `addProjection`, which takes as input a parameter of type `it.sogetel.voltacriteria.criteria.projections.Projections` by which to specify the columns or functions that must be returned by the statement:

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.SELECT, "news")
.addProjection(Projections.ttl("date"))
.addProjection(Projections.property("name"));
```

obtaining:

```
SELECT TTL(date) as t, name
FROM news;
```

Again, you can specify the selection criteria of the query, specifying the individual conditions by the method `addRestriction` (see the paragraph of UPDATE):

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.SELECT, "news")
.addProjection(Projections.ttl("date"))
.addProjection(Projections.property("name"))
.addRestriction(Restrictions.eq("category",
"sport"))
.addRestriction(Restrictions.ltMaxTimeUUID("date",
"2014-01-01 00:05+0000"));
```

whose output will return:

```
SELECT TTL(date) as t, name
FROM news
WHERE category='sport'
AND date < maxtimeuuid('2014-01-01 00:05+0000');
```

You can also insert a sort clause adding the method `AddOrder` that takes as input an object of type `it.sogetel.voltacriteria.criteria.order.Order` through which you can specify the sort order and the ordering columns. Therefore, by modifying the sample code as follows:

```
String[] orderParam = {"date"};

Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.SE
LECT, "news")
.addProjection(Projections.ttl("date"))
.addProjection(Projections.property("name"))
.addRestriction(Restrictions.eq("category",
"sport"))
.addRestriction(Restrictions.ltMaxTimeUUID("date",
"2014-01-01 00:05+0000"))
.addOrder(Order.desc(orderParam));
```

will have as output:

```
SELECT TTL(date) as t, name
FROM news
WHERE category= 'sport'
AND date < maxtimeuuid('2014-01-01 00:05+0000')
ORDER BY date DESC;
```

To limit the number of rows of query output, you can call the method `addLimit` which specify the maximum number of lines:

```
String[] orderParam = {"date"};


Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.SE
LECT, "news")
.addProjection(Projections.ttl("date"))
.addProjection(Projections.property("name"))
.addRestriction(Restrictions.eq("category",
"sport"))
.addRestriction(Restrictions.ltMaxTimeUUID("date",
"2014-01-01 00:05+0000"))
.addOrder(Order.desc(orderParam))
.addLimit(100);
```

The criteria object will contain the query:

```
SELECT TTL(date) as t, name
FROM news
WHERE category='sport'
AND date < maxtimeuuid('2014-01-01 00:05+0000')
ORDER BY date DESC
LIMIT 100;
```

In the case of very expensive queries, you can add the ALLOW FILTERING clause simply adding the call to the method `addAllowFiltering` without parameters. The code:

```
Criteria criteria =
CriteriaFactory.create(CriteriaFactory.QueryType.SE
LECT, "news")
```

```
.addProjection(Projections.ttl("date"))
.addProjection(Projections.property("name"))
.addRestriction(Restrictions.eq("category",
"sport"))
.addRestriction(Restrictions.ltMaxTimeUUID("date",
"2014-01-01 00:05+0000"))
.addOrder(Order.desc(orderParam))
.addLimit(100)
.addAllowFiltering();
```

then return:

```
SELECT TTL(date) as t, name
FROM news
WHERE category='sport'
AND date < maxtimeuuid('2014-01-01 00:05+0000')
ORDER BY date DESC
LIMIT 100
ALLOW FILTERING;
```

5.4 Statements execution

As stated in the introductory paragraph to the library, it was decided not to include within the library connection methods to the database Cassandra and methods to execute statements. So you can choose, as needed, the driver by which you want to perform operations; will just pass the return of the `toString()` method of the `Criteria` object to the execution method.

Using the driver provided by Datastax [Ref.02], for example, you will operate as follows:

```
private Session session;
session = cluster.connect();
```


```
session.execute(criteria.toString());
```