



Prepare Cassandra to receive data from Spark

Before we start connecting to Cassandra, I want to clarify two things about Cassandra database.

1. Cassandra has the concept of a *Keyspace*, which is similar to a database in relational databases. A *Keyspace* is a level where we define data partitioning and replication strategy. So, I don't recommend creating a *Keyspace* on the fly.
2. Cassandra data model design is a conscious decision that completely depends upon the queries that you want to execute on that table. So, I don't recommend creating Cassandra tables on the fly. Most of the time, you will create a Cassandra table and populate the data from Spark.

Great!

1. Connect to Cassandra using the cqlsh client.

|  |  |
| --- | --- |
|  | cqlsh |

1. Execute a CQL statement to test if Cassandra cluster is up and running.

|  |  |
| --- | --- |
|  | SELECT cluster\_name, listen\_address FROM system.local; |

1. Create a *Keyspace* using below CQL command.

|  |  |
| --- | --- |
|  | create KEYSPACE sparkdb WITH replication = {'class': 'SimpleStrategy', 'replication\_factor': 3}; |

1. Create a Cassandra table.

|  |  |
| --- | --- |
|  | create table sparkdb.survey\_results( gender text, sum\_yes int, sum\_no int, primary key (gender)); |

1. Test the table.

|  |  |
| --- | --- |
|  | select \* from sparkdb.survey\_results; |

Great! Now we are ready to jump to your Apache Spark machine and try to connect Cassandra and load some data into this table.

How to write Spark data frame to Cassandra table

Start spark shell and add Cassandra connector package dependency to your classpath. I am using the latest connector as on date. The latest version of Spark uses Scala 2.11, and hence I am using the connector for Scala 2.11.

|  |  |
| --- | --- |
|  | spark-shell --packages datastax:spark-cassandra-connector:2.0.1-s\_2.11 |
|  |  |

The next step is to create a data frame that holds some data. Then we can write that data frame to a Cassandra table. Let's create a data frame. Use following code to create a data frame.

|  |  |
| --- | --- |
|  | //Read CSV into Data Frame |
|  | val df = spark.read |
|  | .format("csv") |
|  | .option("header", "true") |
|  | .option("inferSchema", "true") |
|  | .option("nullValue", "NA") |
|  | .option("timestampFormat", "yyyy-MM-dd'T'HH:mm:ss") |
|  | .option("mode", "failfast") |
|  | .load("/home/prashant/spark-data/mental-health-in-tech-survey/survey.csv") |
|  |  |
|  | df.createOrReplaceTempView("survey\_tbl") |
|  |  |
|  | val dfout = spark.sql( |
|  | """select gender, sum(yes) sum\_yes, sum(no) sum\_no |
|  | from (select case when lower(trim(gender)) in ('male','m','male-ish','maile','mal','male (cis)', |
|  | 'make','male ','man','msle','mail','malr','cis man', |
|  | 'cis male') then 'Male' |
|  | when lower(trim(gender)) in ('cis female','f','female','woman','femake','female ', |
|  | 'cis-female/femme','female (cis)','femail') then 'Female' |
|  | else 'Transgender' |
|  | end as gender, |
|  | case when treatment == 'Yes' then 1 else 0 end as yes, |
|  | case when treatment == 'No' then 1 else 0 end as no |
|  | from survey\_tbl) |
|  | where gender != 'Transgender' |
|  | group by gender""" |
|  | ) |

You already understand this code. Right? We used it in the earlier videos. If you look at the final data frame, the schema of my data frame complies with the schema of the target Cassandra table. That's a critical requirement. You might face a lot of issues if both the schemas are not same.  
You can use the below code to save the data frame to Cassandra. The code is similar to what we have been using in this tutorial. Specify the format as Cassandra. You can use the overwrite mode, but you also have to confirm the truncation. You must specify your Cassandra host, port, *Keyspace*, and the target table name. That's it. Execute the same method and your data moves to Cassandra.

|  |  |
| --- | --- |
|  | dfout.write |
|  | .format("org.apache.spark.sql.cassandra") |
|  | .mode("overwrite") |
|  | .option("confirm.truncate", "true") |
|  | .option("spark.cassandra.connection.host", "10.142.0.3") |
|  | .option("spark.cassandra.connection.port", "9042") |
|  | .option("keyspace", "sparkdb") |
|  | .option("table", "survey\_results") |
|  | .save() |

You can go back to your CQL client and verify the data load.

How to configure Spark Cassandra connection parameters

If you think that specifying Cassandra host and port for every table write operation is not a good thing to do. You can specify these settings at the Spark session level. Use the below code.

|  |  |
| --- | --- |
|  | import org.apache.spark.sql.cassandra.\_ |
|  | spark.setCassandraConf( |
|  | Map( |
|  | "spark.cassandra.connection.host" -> "10.142.0.3", |
|  | "spark.cassandra.connection.port" -> "9042" |
|  | ) |
|  | ) |

Once you set these settings at the session level, you can save the data frame without these settings. You can test it using the following code.

|  |  |
| --- | --- |
|  | dfout.write |
|  | .format("org.apache.spark.sql.cassandra") |
|  | .mode("overwrite") |
|  | .option("confirm.truncate", "true") |
|  | .option("keyspace", "sparkdb") |
|  | .option("table", "survey\_results") |
|  | .save() |

In fact, you can pass these configurations from the command line as well using *--conf* option. There is no need to hardcode the connection parameters in your program.

How to read Cassandra table in Spark

Reading from Cassandra table is again as simple as reading from any other data source. You can use the following code to test the read operation.

|  |  |
| --- | --- |
|  | val df\_read = spark.read |
|  | .format("org.apache.spark.sql.cassandra") |
|  | .option("spark.cassandra.connection.host", "10.142.0.3") |
|  | .option("spark.cassandra.connection.port", "9042") |
|  | .option("keyspace", "sparkdb") |
|  | .option("table", "survey\_results") |
|  | .load() |
|  | df\_read.show |