Data Warehousing And Big Data Pipeline

Running Hive Queries in Spark

edureka!

© Brain4ce Education Solutions Pvt. Ltd.

Running Hive Queries in Spark

In this document, let us learn how to run Hive queries within Spark. Following are the steps to perform:

Step 1: Login to Spark shell
Command: spark2-shell

```
[edureka_396201@ip-20-0-41-190 ~]$ spark2-shell
Setting default log level to "ERROR".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context available as 'sc' (master = yarn, app id = application_1580701586944_2416).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0.cloudera2
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_144)
Type in expressions to have them evaluated.
Type :help for more information.
```

Step 2: Import the HiveContext class from org.apache.spark.sql.hive.HiveContext package
Command: import org.apache.spark.sql.hive.HiveContext

```
scala> import org.apache.spark.sql.hive.HiveContext
import org.apache.spark.sql.hive.HiveContext
```

Step 3: Create an instance of HiveContext class
Command: val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)

```
scala> val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)
warning: there was one deprecation warning; re-run with -deprecation for details
hiveContext: org.apache.spark.sql.hive.HiveContext = org.apache.spark.sql.hive.HiveContext@516b84d1
```

Step 4: Verify if you can access Hive from Spark shell
Command: hiveContext.sql("show tables").collect().foreach(println)

```
scala> hiveContext.sql("show tables").collect()
res1: Array[org.apache.spark.sql.Row] = Array([default,01studentsk,false], [default,04jun,f
alse], [default,181118_hive,false], [default,1codeshare,false], [default,201408_trip_data,f
alse], [default,20180104_hive_431591,false], [default,20181110_hive_431591,false], [default
,20181118_hive,false], [default,20181118_hive_431591,false], [default,20181118_hive_4315911
1,false], [default,20181121_sparkhive_431591,false], [default,21october_emp,false], [defaul
t,386334_bucketed_user,false], [default,386334_partitioned,false], [default,386334_partitio
ned_user,false], [default,386334bucket1,false], [default,386334bucket2,false], [default,395
030_emp_details,false], [default,56343_elp_10nov,false], [default,56343_elp_15oct,false], [
default,56343_elp_31jul,false], [default,808375_country,false], [defa...
```

Step 5: Create an external table in Hive using Spark

Command: hiveContext.sql("CREATE EXTERNAL TABLE if not exists amazon_review(marketplace string, customer_id int, review_id string, product_id string, product_parent string, product_title string, product_category string, star_rating int, helpful_votes int, total_votes int, vine string, verified_purchase string, review_headline string, review_body string, review_date date) ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/bigdatapgp/common_folder/hive_datasets/amazon' tblproperties('skip.header.line.count'='1')")

```
scala> hiveContext.sql("CREATE EXTERNAL TABLE if not exists amazon_review(  marketplace str
ing, customer_id int, review_id string, product_id string, product_parent string, product_t
itle string, product_category string, star_rating int, helpful_votes int, total_votes int,
vine string, verified_purchase string, review_headline string, review_body string, review_d
ate date)  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' LOCATION '/bigdatapgp/common_fold
er/hive_datasets/amazon' tblproperties('skip.header.line.count'='1' )")
res2: org.apache.spark.sql.DataFrame = []
```

Step 6: Login to Hive shell and verify if the table-amazon_review is created

Command: describe amazon_review;

```
[edureka_396201@ip-20-0-41-190 ~]$ hive
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512M; support was removed in 8.0
Java HotSpot(TM) 64-Bit Server VM warning: Using incremental CMS is deprecated and will likely be removed in a future release
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512M; support was removed in 8.0

Logging initialized using configuration in jar:file:/opt/cloudera/parcels/CDH-5.11.1-1.cdh5.11.1.p0.4/jars/hive-common-1.1.0-cdh5.11.1.jar!/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
hive> describe amazon_review;
OK
marketplace             string
customer_id             int
review_id               string
product_id              string
product_parent          string
product_title           string
product_category        string
star_rating             int
helpful_votes           int
total_votes             int
vine                    string
verified_purchase       string
review_headline         string
review_body             string
review_date             date
Time taken: 1.962 seconds, Fetched: 15 row(s)
```

Step 7: Run a select query on the above created table that groups the data by year and get the count of records per year

Command: hiveContext.sql("select year(review_date), count(*) from amazon_review group by year(review_date)") .collect().foreach(println)

```
scala> hiveContext.sql("select year(review_date), count(*) from amazon_review group by year
(review_date)") .collect().foreach(println)
[2003,100]
[2007,8886]
[2015,1304875]
[2006,2190]
[2013,609699]
[null,1]
[2014,1175123]
[2004,281]
[2012,219719]
[2009,23948]
[2001,48]
[2005,660]
[2000,31]
[2010,51055]
[2011,104415]
[2008,13853]
[1999,5]
[2002,54]
```

Step 8: Verify the output by running the same select query in Hive shell.

Command: select year(review_date), count(*) from amazon_review group by year(review_date);

```
hive> select year(review_date), count(*) from amazon_review group by year(review_date);
Query ID = edureka_396201_20200213095656_1269cf9d-687b-4e63-a84c-3b2a1f9d010c
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 21
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
```

.

.

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 6  Reduce: 21   Cumulative CPU: 136.21 sec   HDFS Read: 1353032916 HDFS Write: 173 SUCCESS
Total MapReduce CPU Time Spent: 2 minutes 16 seconds 210 msec
OK
1999    5
2000    31
2001    48
2002    54
2003    100
2004    281
2005    660
2006    2190
2007    8886
2008    13853
2009    23948
2010    51055
2011    104415
2012    219719
2013    609699
2014    1175123
2015    1304875
Time taken: 188.221 seconds, Fetched: 17 row(s)
```

Step 9: You can fetch the count of records per year using Spark as well by using following commands:
Command: val amazon_data= hiveContext.sql("select * from amazon_review")

```
scala> val amazon_data= hiveContext.sql("select * from amazon_review")
amazon_data: org.apache.spark.sql.DataFrame = [marketplace: string, customer_id: int ... 13
 more fields]
```

Command: amazon_data.groupBy(year($"review_date")).count().show

```
scala> amazon_data.groupBy(year($"review_date")).count().show
+-----------------+-------+
|year(review_date)|  count|
+-----------------+-------+
|             2003|    100|
|             2007|   8886|
|             2015|1304875|
|             2006|   2190|
|             2013| 609699|
|             null|      1|
|             2014|1175123|
|             2004|    281|
|             2012| 219719|
|             2009|  23948|
|             2001|     48|
|             2005|    660|
|             2000|     31|
|             2010|  51055|
|             2011| 104415|
|             2008|  13853|
|             1999|      5|
|             2002|     54|
+-----------------+-------+
```

You have successfully run Hive queries in Spark 😊