Spark SQL - Hive Tables

Hive comes bundled with the Spark library as **HiveContext**, which inherits from **SQLContext**. Using HiveContext, you can create and find tables in the HiveMetaStore and write queries on it using HiveQL. Users who do not have an existing Hive deployment can still create a HiveContext. When not configured by the **hive-site.xml**, the context automatically creates a metastore called **metastore_db** and a folder called **warehouse** in the current directory.

Consider the following example of **employee** record using Hive tables. All the recorded data is in the text file named **employee.txt.** Here, we will first initialize the HiveContext object. Using that, we will create a table, load the employee record data into it using HiveQL language, and apply some queries on it.

employee.txt − Place it in the current directory where the spark-shell is running.

```
1201, satish, 25
1202, krishna, 28
1203, amith, 39
1204, javed, 23
1205, prudvi, 23
```

Start the Spark Shell

First, we have to start the Spark Shell. Working with HiveTables means we are working on Hive MetaStore. Hence, the system will automatically create a warehouse for storing table data. Therefore, it is better to run Spark Shell on super user. Consider the following command.

```
$ su
password:
#spark-shell
scala>
```

Create SQLContext Object

Use the following command for initializing the HiveContext into the Spark Shell

```
scala> val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)
```

Create Table using HiveQL

Use the following command for creating a table named **employee** with the fields **id**, **name**, and **age**. Here, we are using the **Create** statement of **HiveQL** syntax.

```
scala> sqlContext.sql("CREATE TABLE IF NOT EXISTS employee(id INT, name STRING, age INT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n'")
```

Load Data into Table using HiveQL

Use the following command for loading the employee record data into the employee table. If it is executed successfully, the given employee records are stored into the **employee** table as per the schema.

```
scala> sqlContext.sql("LOAD DATA LOCAL INPATH 'employee.txt' INTO TABLE employee")
```

Select Fields from the Table

We can execute any kind of SQL queries into the table. Use the following command for fetching all records using HiveQL select query.

```
scala> val result = sqlContext.sql("FROM employee SELECT id, name, age")
```

To display the record data, call the **show()** method on the result DataFrame.

```
scala> result.show()
```

Output

```
<console>:26, took 0.157137 s
+------+---------+----+
| id | name |age |
+------+---------+----+
| 1201 | Satish | 25 |
| 1202 | Krishna | 28 |
| 1203 | amith | 39 |
| 1204 | javed | 23 |
| 1205 | prudvi | 23 |
+------+---------+----+
```