

By the end of this activity, you will be able to:

1. Access Postgres database tables with SparkSQL
2. Filter rows and columns of a Spark DataFrame
3. Group and perform aggregate functions on columns in a Spark DataFrame
4. Join two SparkDataframes on a single column

Step 1. **Open Jupyter Python Notebook for SparkSQL.** First Open the Terminal and enter the command "pyspark" to setup the server. Next, open a web browser by clicking on the web browser icon at the top of the toolbar:



Navigate to *localhost:8889/tree/Downloads/big-data-3/spark-sql*:

A screenshot of a web browser's address bar. It features a small globe icon on the left, followed by the text "localhost:8889/tree/Downloads/big-data-3/spark-sql". The address bar has a light gray border.

Open the SparkSQL Notebook by clicking on *SparkSQL.ipynb*:

A file explorer interface showing a single file. On the left is a small square icon. To its right is a green document icon with a white code symbol. Further right is the text "SparkSQL.ipynb" in a blue font.

Step 2. **Connect to Postgres Table.** This notebook already contains three lines of code so you do not have to enter them. Run these three lines. The first line imports the SQLContext module, which is needed access SQL databases in Spark:

```
In [1]: from pyspark.sql import SQLContext
```

The second line creates a new SQLContext from the SparkContext sc:

```
In [2]: sqlsc = SQLContext(sc)
```

The third line creates a new Spark DataFrame in the variable *df* for the Postgres table *gameclicks*:

```
In [3]: df = sqlsc.read.format("jdbc") \
    .option("url", "jdbc:postgresql://localhost/cloudera?user=cloudera") \
    .option("dbtable", "gameclicks") \
    .load()
```

The *format("jdbc")* says that the source of the DataFrame will be using a Java database connection, the *url* option is the URL connection string to access the Postgres database, and the *dbtable* option specifies the *gameclicks* table.

Step 3. **View Spark DataFrame schema and count rows.** We can call the *printSchema()* method to view the schema of the DataFrame:

```
In [4]: df.printSchema()

root
|-- timestamp: timestamp (nullable = false)
|-- clickid: integer (nullable = false)
|-- userid: integer (nullable = false)
|-- usersessionid: integer (nullable = false)
|-- ishit: integer (nullable = false)
|-- teamid: integer (nullable = false)
|-- teamlevel: integer (nullable = false)
```

The description lists the name and data type of each column.

We can also call the `count()` method to count the number of rows in the DataFrame:

```
In [5]: df.count()
```

```
Out[5]: 755806
```

Step 4. **View contents of DataFrame.** We can call the `show()` method to view the contents of the DataFrame. The argument specifies how many rows to display:

```
In [6]: df.show(5)
```

```
+-----+-----+-----+-----+-----+-----+-----+
|          timestamp|clickid|userid|usersessionid|ishit|teamid|teamlevel|
+-----+-----+-----+-----+-----+-----+-----+
|2016-05-26 15:06:...|    105|   1038|         5916|    0|    25|         1|
|2016-05-26 15:07:...|    154|   1099|         5898|    0|    44|         1|
|2016-05-26 15:07:...|    229|    899|         5757|    0|    71|         1|
|2016-05-26 15:07:...|    322|   2197|         5854|    0|    99|         1|
|2016-05-26 15:07:...|     22|   1362|         5739|    0|    13|         1|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Step 5. **Filter columns in DataFrame.** We can filter for one or more columns by calling the *select()* method:

```
In [7]: df.select("userid", "teamlevel").show(5)
```

userid	teamlevel
1038	1
1099	1
899	1
2197	1
1362	1

only showing top 5 rows

Step 6. **Filter rows based on criteria.** We can also filter for rows that match a specific criteria using *filter()*:

```
In [8]: df.filter(df["teamlevel"] > 1).select("userid", "teamlevel").show(5)
```

userid	teamlevel
1513	2
868	2
1453	2
1282	2
1473	2

only showing top 5 rows

The arguments to *filter()* are a Column, in this case specified as *df["teamlevel"]*, and the condition, which is greater than 1. The remainder of the commander selects only the *userid* and *teamlevel* columns and shows the first five rows.

Step 7. **Group by a column and count.** The `groupBy()` method groups the values of column(s). The `ishit` column only has values 0 and 1. We can calculate how many times each occurs by grouping the `ishit` column and counting the result:

```
In [9]: df.groupBy("ishit").count().show()
```

```
+-----+-----+
|ishit| count|
+-----+-----+
|    0|672423|
|    1| 83383|
+-----+-----+
```

Step 8. **Calculate average and sum.** Aggregate operations can be performed on columns of DataFrames. First, let's import the Python libraries for the aggregate operations. Next, we can calculate the average and total values by calling the `mean()` and `sum()` methods, respectively:

```
In [10]: from pyspark.sql.functions import *
df.select(mean('ishit'), sum('ishit')).show()
```

```
+-----+-----+
|      avg(ishit)|sum(ishit)|
+-----+-----+
|0.1103232840173272|      83383|
+-----+-----+
```

Step 9. **Join two DataFrames.** We can merge or join two Dataframes on a single column. First, let's create a DataFrame for the `adclicks` table in the Postgres database by copying the third cell in this notebook and changing `gameclicks` to `adclicks` and storing the result in a new variable `df2`:



```
In [11]: df2 = sqlsc.read.format("jdbc") \
        .option("url", "jdbc:postgresql://localhost/cloudera?user=cloudera") \
        .option("dbtable", "adclicks") \
        .load()
```

Let's view the columns in *df2* by calling *printSchema()*:

```
In [12]: df2.printSchema()

root
 |-- timestamp: timestamp (nullable = false)
 |-- txid: integer (nullable = false)
 |-- usersessionid: integer (nullable = false)
 |-- teamid: integer (nullable = false)
 |-- userid: integer (nullable = false)
 |-- adid: integer (nullable = false)
 |-- adcategory: string (nullable = false)
```

We can see that the *adclicks* *df2* DataFrame also has a column called *userid*. Next, we will combine the *gameclicks* and *adclicks* DataFrames by calling the *join()* method and saving the resulting DataFrame in a variable called *merge*:

```
In [13]: merge = df.join(df2, 'userid')
```

We are calling the *join()* method on the *gameclicks* DataFrame; the first argument is the DataFrame to join with, i.e., the *adclicks* DataFrame, and the second argument is the column name in both DataFrames to join on.

Let's view the schema of *merge*:

```
In [14]: merge.printSchema()

root
|-- userid: integer (nullable = false)
|-- timestamp: timestamp (nullable = false)
|-- clickid: integer (nullable = false)
|-- usersessionid: integer (nullable = false)
|-- ishit: integer (nullable = false)
|-- teamid: integer (nullable = false)
|-- teamlevel: integer (nullable = false)
|-- timestamp: timestamp (nullable = false)
|-- txid: integer (nullable = false)
|-- usersessionid: integer (nullable = false)
|-- teamid: integer (nullable = false)
|-- adid: integer (nullable = false)
|-- adcategory: string (nullable = false)
```

We can see that the merged DataFrame has all the columns of both *gameclicks* and *adclicks*.

Finally, let's look at the contents of *merge*:

```
In [15]: merge.show(5)
```

	userid	timestamp	clickid	usersessionid	ishit	teamid	teamlevel	timestamp	txid	usersessionid	teamid	adid	adcategory
231	2016-06-08 00:45:...	376796	23626	0	142	4	2016-06-08 01:40:...	23669	23626	142	27	games	
231	2016-06-08 00:45:...	376796	23626	0	142	4	2016-06-08 09:24:...	24122	23626	142	4	games	
231	2016-06-08 00:45:...	376796	23626	0	142	4	2016-06-08 17:21:...	24659	23626	142	22	computers	
231	2016-06-08 00:45:...	376796	23626	0	142	4	2016-06-08 23:34:...	25076	23626	142	21	movies	
231	2016-06-08 00:45:...	376796	23626	0	142	4	2016-06-09 16:32:...	26220	23626	142	16	clothing	

only showing top 5 rows