9_spark_sql_quiz_solution

July 11, 2021

1 Answer Key to the Data Wrangling with Spark SQL Quiz

This quiz uses the same dataset and most of the same questions from the earlier "Quiz - Data Wrangling with Data Frames Jupyter Notebook." For this quiz, however, use Spark SQL instead of Spark Data Frames.

Helpful resources: http://spark.apache.org/docs/latest/api/python/pyspark.sql.html

```
In [4]: from pyspark.sql import SparkSession
        # from pyspark.sql.functions import isnan, count, when, col, desc, udf, col, sort_array,
        # from pyspark.sql.functions import sum as Fsum
        # from pyspark.sql.window import Window
        # from pyspark.sql.types import IntegerType
```

```
In [5]: # 1) import any other libraries you might need
        # 2) instantiate a Spark session
        # 3) read in the data set located at the path "data/sparkify_log_small.json"
        # 4) create a view to use with your SQL queries
        # 5) write code to answer the quiz questions

        spark = SparkSession \
            .builder \
            .appName("Spark SQL Quiz") \
            .getOrCreate()

        user_log = spark.read.json("data/sparkify_log_small.json")

        user_log.createOrReplaceTempView("log_table")
```

2 Question 1

Which page did user id "" (empty string) NOT visit?

```
In [6]: user_log.printSchema()
```

```
root
 |-- artist: string (nullable = true)
 |-- auth: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- itemInSession: long (nullable = true)
 |-- lastName: string (nullable = true)
 |-- length: double (nullable = true)
 |-- level: string (nullable = true)
 |-- location: string (nullable = true)
 |-- method: string (nullable = true)
 |-- page: string (nullable = true)
 |-- registration: long (nullable = true)
 |-- sessionId: long (nullable = true)
 |-- song: string (nullable = true)
 |-- status: long (nullable = true)
 |-- ts: long (nullable = true)
 |-- userAgent: string (nullable = true)
 |-- userId: string (nullable = true)
```

```
In [7]: # SELECT distinct pages for the blank user and distinc pages for all users
        # Right join the results to find pages that blank visitor did not visit
        spark.sql("SELECT * \
                    FROM ( \
                        SELECT DISTINCT page \
                        FROM log_table \
                        WHERE userID='') AS user_pages \
                    RIGHT JOIN ( \
                        SELECT DISTINCT page \
                        FROM log_table) AS all_pages \
                    ON user_pages.page = all_pages.page \
                    WHERE user_pages.page IS NULL").show()
```

```
+----+----------------+
|page|            page|
+----+----------------+
|null|Submit Downgrade|
|null|       Downgrade|
|null|          Logout|
|null|   Save Settings|
|null|        Settings|
|null|        NextSong|
|null|         Upgrade|
|null|           Error|
|null|  Submit Upgrade|
+----+----------------+
```

3 Question 2 - Reflect

Why might you prefer to use SQL over data frames? Why might you prefer data frames over SQL?

Both Spark SQL and Spark Data Frames are part of the Spark SQL library. Hence, they both use the Spark SQL Catalyst Optimizer to optimize queries.

You might prefer SQL over data frames because the syntax is clearer especially for teams already experienced in SQL.

Spark data frames give you more control. You can break down your queries into smaller steps, which can make debugging easier. You can also cache intermediate results or repartition intermediate results.

4 Question 3

How many female users do we have in the data set?

```
In [8]: spark.sql("SELECT COUNT(DISTINCT userID) \
                    FROM log_table \
                    WHERE gender = 'F'").show()
```

```
+----------------------+
|count(DISTINCT userID)|
+----------------------+
|                   462|
+----------------------+
```

5 Question 4

How many songs were played from the most played artist?

```
In [9]: # Here is one solution
        spark.sql("SELECT Artist, COUNT(Artist) AS plays \
                FROM log_table \
                GROUP BY Artist \
                ORDER BY plays DESC \
                LIMIT 1").show()

        # Here is an alternative solution
        # Get the artist play counts
        play_counts = spark.sql("SELECT Artist, COUNT(Artist) AS plays \
                FROM log_table \
                GROUP BY Artist")

        # save the results in a new view
        play_counts.createOrReplaceTempView("artist_counts")
        # use a self join to find where the max play equals the count value
        spark.sql("SELECT a2.Artist, a2.plays FROM \
                  (SELECT max(plays) AS max_plays FROM artist_counts) AS a1 \
                  JOIN artist_counts AS a2 \
                  ON a1.max_plays = a2.plays \
                  ").show()
```

```
+--------+-----+
|  Artist|plays|
+--------+-----+
|Coldplay|   83|
+--------+-----+
```

```
+--------+-----+
|  Artist|plays|
+--------+-----+
|Coldplay|   83|
+--------+-----+
```

6 Question 5 (challenge)

How many songs do users listen to on average between visiting our home page? Please round your answer to the closest integer.

```
In [31]: # SELECT CASE WHEN 1 > 0 THEN 1 WHEN 2 > 0 THEN 2.0 ELSE 1.2 END;
         is_home = spark.sql("SELECT userID, page, ts, CASE WHEN page = 'Home' THEN 1 ELSE 0 END
                     WHERE (page = 'NextSong') or (page = 'Home') \
                     ")

         # keep the results in a new view
         is_home.createOrReplaceTempView("is_home_table")

         # find the cumulative sum over the is_home column
         cumulative_sum = spark.sql("SELECT *, SUM(is_home) OVER \
             (PARTITION BY userID ORDER BY ts DESC ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT
             FROM is_home_table")

         # keep the results in a view
         cumulative_sum.createOrReplaceTempView("period_table")

         # find the average count for NextSong
         spark.sql("SELECT AVG(count_results) FROM \
                   (SELECT COUNT(*) AS count_results FROM period_table \
         GROUP BY userID, period, page HAVING page = 'NextSong') AS counts").show()
```

```
+------------------+
|avg(count(period))|
+------------------+
| 6.898347107438017|
+------------------+
```