databricksHW5_template

(https://databricks.com)

Assignment 5

setup environment (must run first)

4

```
%python
# Delete or comment this block out when running the notebook on the full dataset
# users_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_sample/Users.json")
# records_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_sample/Records.json")
# sessions_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_sample/Sessions.json")
# reviews_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_sample/Reviews.json")
# upvotes_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_sample/Upvotes.json")

# users_df.createOrReplaceTempView("users")
# records_df.createOrReplaceTempView("records")
# sessions_df.createOrReplaceTempView("sessions")
# reviews_df.createOrReplaceTempView("reviews")
# upvotes_df.createOrReplaceTempView("upvotes")
```

5

```
# %python
# # Uncomment this block when running the notebook on the full dataset (you must do this in the end before submitting the
solution!)
users_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_full/Users.json")
records_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_full/Records.json")
sessions_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_full/Sessions.json")
reviews_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_full/Reviews.json")
upvotes_df = spark.read.format("json").option("multiline", True).load("/FileStore/tables/zotmusic_full/Upvotes.json")

users_df.createOrReplaceTempView("users")
records_df.createOrReplaceTempView("records")
sessions_df.createOrReplaceTempView("sessions")
reviews_df.createOrReplaceTempView("reviews")
upvotes_df.createOrReplaceTempView("upvotes")
```

```
%sql
(select "users", count(*) from users)
union all
(select "records", count(*) from records)
union all
(select "sessions", count(*) from sessions)
union all
(select "reviews", count(*) from reviews)
union all
(select "upvotes", count(*) from upvotes)
```

Table

	users	count(1)
1	users	5000
2	records	100000
3	sessions	889979
4	reviews	962143
5	upvotes	9189735

5 rows

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Solution

8

```
#1.A
users_df.printSchema()

#1.B address data type is: struct. It has 4 string fields: city, state, street and zip.
```

```
root
 |-- address: struct (nullable = true)
 |    |-- city: string (nullable = true)
 |    |-- state: string (nullable = true)
 |    |-- street: string (nullable = true)
 |    |-- zip: string (nullable = true)
 |-- bio: string (nullable = true)
 |-- email: string (nullable = true)
 |-- genres: array (nullable = true)
 |    |-- element: string (containsNull = true)
 |-- is_artist: boolean (nullable = true)
 |-- is_listener: boolean (nullable = true)
 |-- joined_date: string (nullable = true)
```

```
|-- nickname: string (nullable = true)
|-- real_name: struct (nullable = true)
|    |-- first_name: string (nullable = true)
|    |-- last_name: string (nullable = true)
|-- stage_name: string (nullable = true)
|-- subscription: string (nullable = true)
|-- user_id: string (nullable = true)
```

9

```
%sql
--1.C
DESCRIBE records;
--1.D songs data type is: array<struct<bpm:bigint,length:bigint,mood:string,title:string,track_number:bigint>>
```

Table

	col_name	data_type	comment
1	description	string	null
2	genre	string	null
3	is_album	boolean	null
4	is_single	boolean	null
5	record_id	string	null
6	released_by	struct<artist_user_id:string,release_date:string>	null
7	songs	array<struct<bpm:bigint,length:bigint,mood:string,title:string,track_number:bigint>>	null
8	title	string	null
9	video_url	string	null

9 rows

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

10

```
#2.A DF
users_df.filter(users_df.user_id == 'user_W2xxLgbyRoqp').show()
```

```
+--------------------+--------------------+--------------------+--------------------+---------+-----------+-----------+-----
----------+---------+-----------+------------+-----------------+
|             address|                 bio|               email|              genres|is_artist|is_listener|joined_date|
nickname|real_name| stage_name|subscription|         user_id|
+--------------------+--------------------+--------------------+--------------------+---------+-----------+-----------+-----
----------+---------+-----------+------------+-----------------+
|{Joelmouth, NULL,...|Full computer tre...|katherinedelgado@...|[Techno, Rock, El...|     true|      false| 2019-05-04|kathe
rinedelgado|     NULL|stacyobrien|        NULL|user_W2xxLgbyRoqp|
+--------------------+--------------------+--------------------+--------------------+---------+-----------+-----------+-----
```

```
+----------+---------+----------+------------+-----------------+
```

11

```
%sql
--2.A SQL
SELECT *
FROM users
WHERE user_id = 'user_W2xxLgbyRoqp';
```

Table

	address	bio
1	{"city":"Joelmouth","state":null,"street":"78374 Michael Centers Suite 220","zip":"8041...	Full computer treat mention agreement. Also success or appear ind

1 row

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

12

```
#2.B DF
pop_records_df = records_df.filter(records_df.genre == "Pop")

pop_record_count_df = (
    pop_records_df.groupBy(pop_records_df.released_by.artist_user_id.alias("user_id"))
    .count()
    .orderBy("count", ascending=False)
)

pop_record_count_df.limit(5).show()
```

```
+-----------------+-----+
|          user_id|count|
+-----------------+-----+
|user_W5FmJadbR6OF|    8|
|user_giDyfRSuSRKk|    7|
|user_tylP-mPISHS1|    7|
|user_h1_M1rkrRs-F|    7|
|user_0lVVujY8RC6Y|    7|
```

```
+-----------------+-----+
```

13

```
%sql
--2.B SQL
SELECT released_by.artist_user_id AS user_id, COUNT(record_id) AS num_pop_records
FROM records
WHERE genre = 'Pop'
GROUP BY released_by.artist_user_id
ORDER BY num_pop_records DESC
LIMIT 5;
```

Table

	user_id	num_pop_records
1	user_W5FmJadbR6OF	8
2	user_giDyfRSuSRKk	7
3	user_tylP-mPlSHS1	7
4	user_h1_M1rkrRs-F	7
5	user_0lVVujY8RC6Y	7

5 rows

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

14

```
#2.C DF
from pyspark.sql.functions import year, month, col, count, desc
november_sessions_df = sessions_df.filter(
    (year(col("session_duration.initiate_at")) == 2023) &
    (month(col("session_duration.initiate_at")) == 11)
)

most_streamed_songs_df = (
    november_sessions_df.groupBy("song.record_id", "song.track_number")
    .agg(count("*").alias("session_count"))
    .orderBy(desc("session_count"))
)

most_streamed_songs_df.limit(3).show()
```

```
+------------------+------------+-------------+
|         record_id|track_number|session_count|
+------------------+------------+-------------+
|record_llStZtAYQfiA|          11|          887|
|record_rv-yMzjfQSC_|           1|          866|
|record_U-VuvQ46SD6p|           2|            3|
+------------------+------------+-------------+
```

15

```
%sql
--2.C SQL
SELECT song.record_id, song.track_number, COUNT(*) AS session_count
FROM sessions
WHERE YEAR(session_duration.initiate_at) = 2023 AND MONTH(session_duration.initiate_at) = 11
GROUP BY song.record_id, song.track_number
ORDER BY session_count DESC
LIMIT 3;
```

Table

	record_id	track_number	session_count
1	record_llStZtAYQfiA	11	887
2	record_rv-yMzjfQSC_	1	866
3	record_U-VuvQ46SD6p	2	3

3 rows

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

16

```
#2.D DF
from pyspark.sql.functions import explode, col, count, desc

exploded_genres_df = users_df.select(explode(col("genres")).alias("genre"))

genre_interest_df = (
    exploded_genres_df.groupBy("genre")
    .agg(count("*").alias("interested_users"))
    .orderBy(desc("interested_users"))
)

genre_interest_df.show(1)
```

```
+-----+----------------+
|genre|interested_users|
+-----+----------------+
| Funk|            1203|
+-----+----------------+
only showing top 1 row
```

17

```
%sql
--2.D SQL
SELECT genre, COUNT(*) AS interested_users
FROM (SELECT EXPLODE(genres) AS genre FROM users) exploded_genres
GROUP BY genre
ORDER BY interested_users DESC
LIMIT 1;
```

Table

	genre	interested_users
1	Funk	1203

1 row

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

18

```
#2.E DF
from pyspark.sql.functions import col, to_date

filtered_reviews_df = reviews_df.filter(
    (to_date(col("posted_by.posted_time")) >= "2024-08-03") &
    (to_date(col("posted_by.posted_time")) < "2024-09-08")
)

joined_df = filtered_reviews_df.join(users_df, col("posted_by.user_id") == col("user_id"))

result_df = (
    joined_df.select("review_id", "email", col("posted_by.posted_time").alias("posted_time"))
    .orderBy(col("posted_time").asc())
    .limit(5)
)

result_df.show()
```

```
+------------------+--------------------+-------------------+
|         review_id|               email|        posted_time|
+------------------+--------------------+-------------------+
|review_aArS-aRvRyG5|robertsaunders@gm...|2024-08-03 00:00:31|
|review_O-V9LubWTBKR|biancadavis@iclou...|2024-08-03 00:01:00|
|review_rNdhyxyxQxud|andrewwatkins@fox...|2024-08-03 00:02:01|
|review_xAEdPYgfSGiS|walleraudrey@gmai...|2024-08-03 00:02:07|
|review_2e4fbC-zSWO8|riverakelsey@yaho...|2024-08-03 00:02:39|
+------------------+--------------------+-------------------+
```

19

```
%sql
--2.E SQL
SELECT reviews.review_id, users.email, reviews.posted_by.posted_time AS posted_time
FROM reviews
JOIN users
ON reviews.posted_by.user_id = users.user_id
WHERE DATE(reviews.posted_by.posted_time) >= '2024-08-03'
AND DATE(reviews.posted_by.posted_time) < '2024-09-08'
ORDER BY reviews.posted_by.posted_time ASC
LIMIT 5;
```

Table

	review_id	email	posted_time

1	review_aArS-aRvRyG5	robertsaunders@gmail.com	2024-08-03 00:00:31
2	review_O-V9LubWTBKR	biancadavis@icloud.com	2024-08-03 00:01:00
3	review_rNdhyxyxQxud	andrewwatkins@foxmail.com	2024-08-03 00:02:01
4	review_xAEdPYgfSGiS	walleraudrey@gmail.com	2024-08-03 00:02:07
5	review_2e4fbC-zSWO8	riverakelsey@yahoo.com	2024-08-03 00:02:39

5 rows

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

20

```
#2.F DF
from pyspark.sql.functions import col, count, desc

reviews_df = reviews_df.withColumn("posted_by_user_id", col("posted_by.user_id"))

upvotes_reviews_df = upvotes_df.join(reviews_df, upvotes_df["review_id"] == reviews_df["review_id"])

upvotes_reviews_users_df = upvotes_reviews_df.join(
    users_df, upvotes_reviews_df["posted_by_user_id"] == users_df["user_id"]
)

result_df = (
    upvotes_reviews_users_df.groupBy("email")
    .agg(count("*").alias("total_upvotes"))
    .orderBy(desc("total_upvotes"))
    .limit(3)
)

result_df.show()
```

```
+--------------------+-------------+
|               email|total_upvotes|
+--------------------+-------------+
|leejames@protonma...|         4682|
|orodriguez@gmail.com|         4626|
|    qbrown@yahoo.com|         4622|
+--------------------+-------------+
```

21

```
%sql
--2.F SQL
SELECT users.email, COUNT(upvotes.review_id) AS total_upvotes
FROM upvotes
JOIN reviews
ON upvotes.review_id = reviews.review_id
JOIN users
ON reviews.posted_by.user_id = users.user_id
GROUP BY users.email
ORDER BY total_upvotes DESC
LIMIT 3;
```

Table

	email	total_upvotes
1	leejames@protonmail.com	4682
2	orodriguez@gmail.com	4626
3	qbrown@yahoo.com	4622

3 rows

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

22

```
#2.G DF
from pyspark.sql.functions import col, avg, count, desc

active_listeners_df = sessions_df.groupBy("user_id").count().filter(col("count") >= 275)

certified_reviews_df = (
    upvotes_df.groupBy("review_id").count().filter(col("count") >= 5)
    .join(reviews_df, "review_id")
    .join(active_listeners_df, reviews_df.posted_by.user_id == active_listeners_df.user_id)
)

artist_certified_ratings_df = certified_reviews_df.join(
    records_df, "record_id"
).select(
    records_df.released_by.artist_user_id.alias("artist_user_id"),
    certified_reviews_df.rating.alias("certified_rating")
)

artist_acrr_df = artist_certified_ratings_df.groupBy("artist_user_id").agg(
    avg("certified_rating").alias("acrr"),
    count("certified_rating").alias("total_certified_ratings")
)

users_df = users_df.withColumn("first_name", col("real_name.first_name"))
users_df = users_df.withColumn("last_name", col("real_name.last_name"))

artist_details_df = artist_acrr_df.join(
    users_df, artist_acrr_df.artist_user_id == users_df.user_id
).select(
    "artist_user_id",
    "acrr",
    "total_certified_ratings",
    "first_name",
    "last_name"
)

top_artists_df = artist_details_df.orderBy(desc("acrr")).limit(5)

top_artists_df.show()
```

```
+----------------+-----------------+----------------------+----------+----------+
|   artist_user_id|             acrr|total_certified_ratings|first_name| last_name|
+----------------+-----------------+----------------------+----------+----------+
|user_BNTubG2nQCaO|3.4583333333333335|                    96|   Kristen|     Lopez|
|user_sEETlirSSTC5|3.4285714285714284|                    98|      NULL|      NULL|
|user_158lawlPQSSz| 3.382716049382716|                    81|      NULL|      NULL|
```

```
|user_DYywyDQ0TPKC|3.3707865168539324|                    89|     Kelly|     Baker|
|user_id0U530vTYOJ| 3.369047619047619|                    84|  Jennifer|Richardson|
+-----------------+------------------+----------------------+----------+----------+
```

```
23
```

```
%sql
--2.G SQL

WITH ActiveListeners AS (
    SELECT user_id
    FROM sessions
    GROUP BY user_id
    HAVING COUNT(*) >= 275
),
CertifiedReviews AS (
    SELECT
        reviews.review_id,
        reviews.rating AS certified_rating,
        reviews.record_id,
        reviews.posted_by.user_id AS listener_id
    FROM reviews
    JOIN (
        SELECT review_id, COUNT(*) AS upvote_count
        FROM upvotes
        GROUP BY review_id
        HAVING COUNT(*) >= 5
    ) AS upvote_counts
    ON reviews.review_id = upvote_counts.review_id
    WHERE reviews.posted_by.user_id IN (SELECT user_id FROM ActiveListeners)
),
ArtistCertifiedRatings AS (
    SELECT
        records.released_by.artist_user_id AS artist_user_id,
        CertifiedReviews.certified_rating
    FROM CertifiedReviews
    JOIN records
    ON CertifiedReviews.record_id = records.record_id
),
ArtistACRR AS (
    SELECT
        artist_user_id,
        AVG(certified_rating) AS acrr,
        COUNT(certified_rating) AS total_certified_ratings
    FROM ArtistCertifiedRatings
    GROUP BY artist_user_id
)
SELECT
    ArtistACRR.artist_user_id,
    ArtistACRR.acrr,
    ArtistACRR.total_certified_ratings,
    users.real_name.first_name AS first_name,
    users.real_name.last_name AS last_name
FROM ArtistACRR
JOIN users
ON ArtistACRR.artist_user_id = users.user_id
```

```
ORDER BY acrr DESC
LIMIT 5;
```

Table

	artist_user_id	acrr	total_certified_ratings	first_name	last_name
1	user_BNTubG2nQCaO	3.4583333333333335	96	Kristen	Lopez
2	user_sEETlirSSTC5	3.4285714285714284	98	null	null
3	user_158lawlPQSSz	3.382716049382716	81	null	null
4	user_DYywyDQ0TPKC	3.3707865168539324	89	Kelly	Baker
5	user_id0U530vTYOJ	3.369047619047619	84	Jennifer	Richardson

5 rows

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

24

```
# 3
# Go BACK to the beginning and run the whole notebook on the full dataset, both in Databricks Community Version and
Databricks Premium Version.
# Write down your observations about the differences of using single-node community version vs. using multi-node premium
version of Databricks.

# The single-node version of Spark is limited by the hardware resources of a single machine, making it slower for
# large-scale data processing. In contrast, the multi-node version distributes tasks across a cluster,
# leveraging parallel computation to significantly improve speed, especially for big data and complex workloads.
# As a result, the multi-node version is much faster for large-scale processing.
```

25