

Saransh Rawat (2022OG04027)

Ayush Bansal (2022OG04042)

Use case: Spotify Recommendation System Platform

Task 3: OLAP Queries

Spark Setup: For this we have used Data Bricks for Spark Infra setup below are the config details

The screenshot shows the configuration page for a cluster named "AB's Cluster".

- Policy:** Unrestricted (radio button selected).
- Access mode:** Multi node (radio button selected).
- Performance:**
 - Databricks runtime version: Runtime 13.3 LTS (Scala 2.12, Spark 3.4.1).
 - Use Photon Acceleration (checkbox checked).
 - Worker type:
 - Standard_DS3_v2 (selected), 14 GB Memory, 4 Cores.
 - Min workers: 2.
 - Max workers: 8.
 - Spot instances (checkbox unchecked).
- Driver type:** Same as worker (14 GB Memory, 4 Cores).
- Autoscaling:**
 - Enable autoscaling (checkbox checked).
 - Terminate after 20 minutes of inactivity (checkbox checked).
- Tags:** An empty key-value pair is added.

Summary:

2-8 Workers	28-112 GB Memory
1 Driver	8-32 Cores
Runtime	14 GB Memory, 4 Cores
	13.3.x-scala2.12
Photon	Standard_DS3_v2
	4-14 DBUs/h

Post completion of the Spark setup here are the few OLAP queries with respect to the Spotify Solution which helps to improve growth/sales

1. Identify Potential Collaborations for High-Growth Artists: Finds pairs of artists who are both experiencing high popularity. This helps identify potential collaborations between artists who are individually popular.

```
WITH HighGrowthArtists AS (
  SELECT artists, AVG(popularity) AS avg_popularity
  FROM spotify_collection
  GROUP BY artists
  HAVING AVG(popularity) > 75
)
SELECT ha1.artists AS high_growth_artist, ha2.artists AS potential_collaborator
FROM HighGrowthArtists ha1, HighGrowthArtists ha2
WHERE ha1.artists < ha2.artists;
```

AB Spotify Task 3 Python ▾ ★

File Edit View Run Help Last edit was now Provide feedback

Run all AB's Cluster Schedule Share

1. Identify Potential Collaborations for High-Growth Artists:

Finds pairs of artists who are both experiencing high popularity. This helps identify potential collaborations between artists who are individually popular.

```

1 Nas1
2
3 WITH HighGrowthArtists AS (
4   SELECT artist_id, AVG(popularity) AS avg_popularity
5   FROM spotify_collection
6   GROUP BY artist_id
7   HAVING AVG(popularity) > 75
8 )
9 SELECT hal.artist_id AS high_growth_artist, hal.artist_id AS potential_collector
10 FROM HighGrowthArtists hal1, HighGrowthArtists hal2
11 WHERE hal.artist_id < hal2.artist_id

```

(3) Spark Jobs

SQL pyspark.sql.DataFrame DataFrame = [high_growth_artist string, potential_collector string]

Table +

high_growth_artist	potential_collector
Calvin Harris John Newman	Jay Wheeler DJ Nelson Myke Towers
Madonna Slick	girl in red
Earth, Wind & Fire The Emotions	Ty Dolla Sign The Weeknd Wiz Khalifa Mustard
Camila Cabello DJ Baby	John Lennon
Calvin Harris Rihanna	Modjo
Alon Shmida Houston	KAMRAD
Kanye West	Uva Monroes

↓ 10,000 rows | Truncated data | 4.44 seconds runtime

This result is stored as PySpark data frame `_sqlDF` and in the IPython output cache as `Out[6]`. Learn more

Refreshed 7 minutes ago

2. Identify Potential Featured Tracks for Cross-Promotion: Finds tracks that could be featured for cross-promotion, considering similar genres and high popularity, which can lead to mutual promotional benefits.

```

WITH PotentialFeaturedTracks AS (
  SELECT st1.track_name, st2.track_name AS featured_track
  FROM spotify_collection st1, spotify_collection st2
  WHERE st1.track_genre = st2.track_genre
    AND st1.popularity > 75
    AND st2.popularity > 70
    AND st1.track_name != st2.track_name
)
SELECT pft.track_name, COUNT(pft.featured_track) AS num_featured_tracks
FROM PotentialFeaturedTracks pft
GROUP BY pft.track_name
ORDER BY num_featured_tracks DESC
LIMIT 5;

```

AB Spotify Task 3 Python ▾ ★

File Edit View Run Help Last edit was 8 minutes ago Provide feedback

Run all AB's Cluster Schedule Share

2. Identify Potential Featured Tracks for Cross-Promotion:

Finds tracks that could be featured for cross-promotion, considering similar genres and high popularity, which can lead to mutual promotional benefits.

```

1 Nas1
2
3 WITH PotentialFeaturedTracks AS (
4   SELECT st1.track_name, st2.track_name AS featured_track
5   FROM spotify_collection st1, spotify_collection st2
6   WHERE st1.track_genre = st2.track_genre
7   AND st1.popularity > 75
8   AND st2.popularity > 70
9   AND st1.track_name != st2.track_name
10 )
11 SELECT pft.track_name, COUNT(pft.featured_track) AS num_featured_tracks
12 FROM PotentialFeaturedTracks pft
13 GROUP BY pft.track_name
14 ORDER BY num_featured_tracks DESC
15 LIMIT 5;
16

```

(3) Spark Jobs

SQL pyspark.sql.DataFrame DataFrame = [track_name string, num_featured_tracks long]

Table +

track_name	num_featured_tracks
Without Me	2251
Heat Waves	1529
Numb	1286
Love Me Like You Do	1235
The Middle	1221

↓ 5 rows | 2.98 seconds runtime

This result is stored as PySpark data frame `_sqlDF` and in the IPython output cache as `Out[10]`. Learn more

Refreshed 5 minutes ago

3. Playlist Recommendations Based on Audio Features: Provides playlist recommendations based on similar audio features (danceability, valence, energy) of tracks.

```
WITH PlaylistRecommendations AS (
    SELECT DISTINCT st1.track_name, st2.track_name AS recommended_track
    FROM spotify_collection st1, spotify_collection st2
    WHERE st1.track_genre = st2.track_genre
        AND st1.track_name != st2.track_name
        AND ABS(st1.danceability - st2.danceability) < 0.1
        AND ABS(st1.valence - st2.valence) < 0.1
        AND ABS(st1.energy - st2.energy) < 0.1
)
SELECT pr.track_name, COUNT(pr.recommended_track) AS num_recommendations
FROM PlaylistRecommendations pr
GROUP BY pr.track_name
ORDER BY num_recommendations DESC
LIMIT 5;
```

The screenshot shows the AB Spotify Task 3 interface. The top navigation bar includes 'File', 'Edit', 'View', 'Run', 'Help', and 'Provide feedback'. On the right, there are buttons for 'Run all', 'AB's Cluster', 'Schedule', and 'Share'. Below the navigation is a text area with two command history entries: 'Cmd 10' and 'Cmd 11'. 'Cmd 10' contains the problem statement: '3. Playlist Recommendations Based on Audio Features: Provides playlist recommendations based on similar audio features (danceability, valence, energy) of tracks.' 'Cmd 11' contains the provided SQL code. To the right of the code is a table titled 'Table' with columns 'track_name' and 'num_recommendations'. The table data is as follows:

track_name	num_recommendations
Home	1213
Without You	1181
Alone	938
Runaway	930
You	914

Below the table, it says '5 rows | 12.92 seconds runtime'. At the bottom, a note states: 'This result is stored as PySpark data frame _sqlIdF and in the IPython output cache as Out[11]. Learn more'.

4. Identify Tracks with Gradual Loudness Decrease: Finds tracks where the loudness gradually decreases, potentially suitable for creating playlists with a calming effect.

```
WITH GradualLoudnessDecrease AS (
    SELECT track_name, loudness,
        LEAD(loudness) OVER (ORDER BY track_name) AS next_loudness
    FROM spotify_table
)
SELECT gld.track_name
FROM GradualLoudnessDecrease gld
WHERE next_loudness < loudness
LIMIT 10;
```

AB Spotify Task 3 Python 

File Edit View Run Help Last edit was 2 minutes ago Provide feedback

Run all AB's Cluster Schedule Share

Cell 12

4. Identify Tracks with Gradual Loudness Decrease:
Finds tracks where the loudness gradually decreases, potentially suitable for creating playlists with a calming effect.

Cell 13

```
1 Nas
2 WITH gradualLoudnessDecrease AS (
3   SELECT track_name, loudness,
4   LAG(loudness) OVER(ORDER BY track_name) AS next_loudness
5   FROM spotify.collection
6 )
7 SELECT gld.track_name
8 WHERE gradualLoudnessDecrease.gld
9 WHERE next_loudness < loudness
10 LIMIT 10;
```

(D) Speak Jobs

* JobID: pyparqsql_dataframe Dataframe <track_name: string>

Table  +

track_name
1 Snuff Crew
2 Vol2***
3 Vol2***
4 Vol2***
5 Vol2***
6 Vol2***
7 Furin Barber***

10 rows | 2.02 seconds runtime

Refreshed 2 minutes ago

This result is stored as PySpark data frame `_sqlDF` and in the Python output cache as `Out[12]`. Learn more