

SQL Data Analysis Project – Spotify Dataset

Dataset Description

The dataset consists of a single table named `spotify`, which contains audio features, engagement metrics, and platform-related information for tracks available on Spotify.

Table: `spotify`

Key columns include:

- `artist` – Name of the artist
- `track` – Track name
- `album / album_type` – Album details
- `danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo` – Audio features
- `duration_min` – Track duration in minutes
- `views, likes, comments, stream` – Engagement metrics
- `licensed, official_video` – Boolean indicators
- `energy_liveness` – Precomputed ratio
- `most_played_on` – Platform where the track is most played

This dataset enables analysis of both musical characteristics and user engagement behavior.

Project Objectives (Business Questions)

The main objectives of this SQL project are:

- Find the top 3 most-viewed tracks for each artist using window functions.
- Write a query to find tracks where the liveness score is above the average.
- Use a `WITH` clause to calculate the difference between the highest and lowest energy values for tracks in each album
- Find tracks where the energy-to-liveness ratio is greater than 1.2.
- Find the top 5 tracks with the highest energy values.
- List all tracks along with their views and likes where `official_video = TRUE`.
- For each album, calculate the total views of all associated tracks.
- Retrieve the track names that have been streamed on Spotify more than YouTube.

Exploratory Data Analysis (EDA)

```
select *  
from spotify;
```

--Count total rows SELECT COUNT(*)
FROM spotify; 7 20592

-- Count total columns
SELECT COUNT(*) AS Column_Count
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'spotify'; 7 24

--total artist
SELECT DISTINCT artist
FROM spotify;

```
SELECT COUNT(DISTINCT artist) 7 2074 FROM spotify;
```

-- Type of album
SELECT DISTINCT album_type 7 album , single , complication from spotify;

-- Max and Min Duration
SELECT MAX(Duration_min) as Max_Duration , MIN(Duration_min) as Min_Duration
FROM spotify;

The screenshot shows a SQL query results window with the following data:

	artist	track	album	album_type	danceability	energy	loudness	speechiness	acousticness	instrumentalness	liveness	valence	temp
1	Natash...	These Words	Unwrit...	album	0	0	0	0	0	0	0	0	0
2	White ...	Rain In the E...	Soothi...	album	0	0	0	0	0	0	0	0	0

Total rows: 2 Query complete 00:00:00.114 CRLF Ln 28, Col 1

-- here we have song with 0 duration , lets check it

```
SELECT *  
FROM spotify  
WHERE Duration_min = 0;
```

-- we got few records with 0 duration , remove them
DELETE FROM spotify
WHERE Duration_min = 0;

After removing , current max and min 7 Max : 77.9343 min Min : 0.516416667 min

SQL Analysis & Queries

1. Find the top 3 most-viewed tracks for each artist using window functions.

with tab as (

```
    SELECT artist,track, SUM(views) as total_views,
          DENSE_RANK() OVER (PARTITION BY artist ORDER BY SUM(views) DESC) as RN
     FROM spotify
    GROUP BY 1,2
)
```

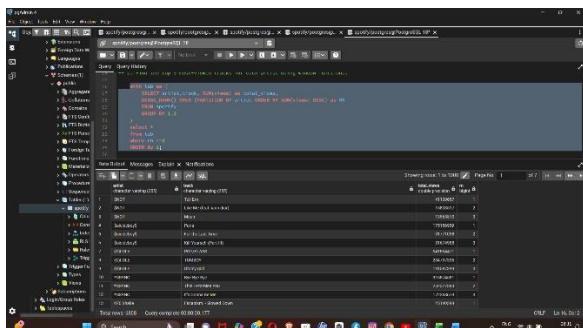
```
select * from tab where rn
```

```
<=3
```

```
        ORDER By 1;
```

-- We dont use group by in top 3 salary in each department

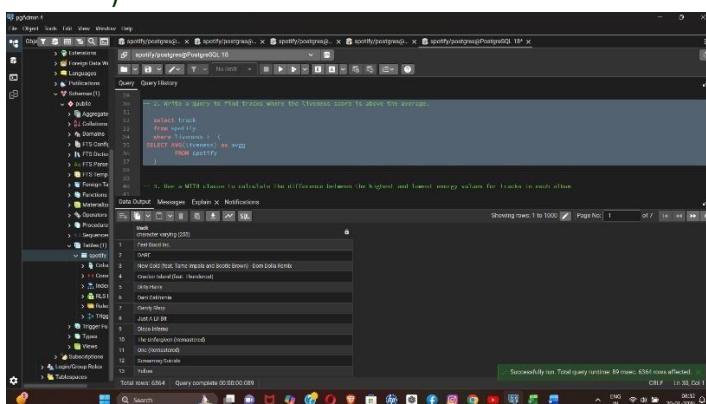
-- but here we have artist artist can have mutiple rows , we need to calculate total of each track thats why we are using group by



I grouped by artist and track to calculate total views, applied a window function to rank tracks per artist, and filtered the top three using DENSE_RANK().

2. Write a query to find tracks where the liveness score is above the average.

```
select track
  from spotify
 where liveness > (  SELECT
      AVG(liveness) as avgg
    FROM spotify
)
```



3. Use a WITH clause to calculate the difference between the highest and lowest energy values for tracks in each album

```
with sp_a as (
    SELECT album , MAX(energy) as max_energy , MIN(energy) as min_energy
    FROM spotify
    GROUP BY 1

)
select album , (max_energy - min_energy) as diff  from
sp_a
order by 2 desc
```

The screenshot shows the pgAdmin 4 interface with multiple tabs open. The main window displays a query in the SQL tab:

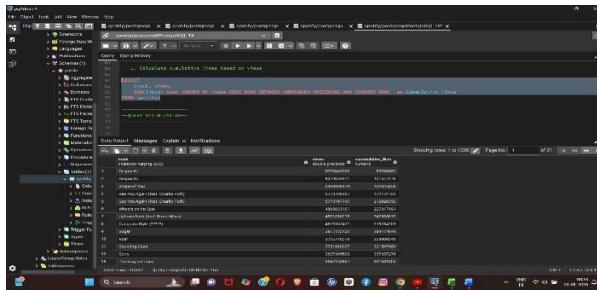
```
with sp_a as (
    SELECT album , MAX(energy) as max_energy , MIN(energy) as min_energy
    FROM spotify
    GROUP BY 1
)
select album , (max_energy - min_energy) as diff  from
sp_a
order by 2 desc
```

The results are shown in the Data Output tab, titled 'Result (1 row(s))'.

album	diff
Wise Tree	0.000100000000
Sentimental	0.000100000000
1	0.000100000000
2	0.000100000000
3	0.000100000000
4	0.000100000000
5	0.000100000000
6	0.000100000000
7	0.000100000000
8	0.000100000000
9	0.000100000000
10	0.000100000000
11	0.000100000000
12	0.000100000000
13	0.000100000000
14	0.000100000000
15	0.000100000000
16	0.000100000000
17	0.000100000000
18	0.000100000000
19	0.000100000000
20	0.000100000000
21	0.000100000000
22	0.000100000000
23	0.000100000000
24	0.000100000000
25	0.000100000000
26	0.000100000000
27	0.000100000000
28	0.000100000000
29	0.000100000000
30	0.000100000000
31	0.000100000000
32	0.000100000000
33	0.000100000000
34	0.000100000000
35	0.000100000000
36	0.000100000000
37	0.000100000000
38	0.000100000000
39	0.000100000000
40	0.000100000000
41	0.000100000000
42	0.000100000000
43	0.000100000000
44	0.000100000000
45	0.000100000000
46	0.000100000000
47	0.000100000000
48	0.000100000000
49	0.000100000000
50	0.000100000000
51	0.000100000000
52	0.000100000000
53	0.000100000000
54	0.000100000000
55	0.000100000000
56	0.000100000000
57	0.000100000000
58	0.000100000000
59	0.000100000000
60	0.000100000000
61	0.000100000000
62	0.000100000000
63	0.000100000000
64	0.000100000000
65	0.000100000000
66	0.000100000000
67	0.000100000000
68	0.000100000000
69	0.000100000000
70	0.000100000000
71	0.000100000000
72	0.000100000000
73	0.000100000000
74	0.000100000000
75	0.000100000000
76	0.000100000000
77	0.000100000000
78	0.000100000000
79	0.000100000000
80	0.000100000000
81	0.000100000000
82	0.000100000000
83	0.000100000000
84	0.000100000000
85	0.000100000000
86	0.000100000000
87	0.000100000000
88	0.000100000000
89	0.000100000000
90	0.000100000000
91	0.000100000000
92	0.000100000000
93	0.000100000000
94	0.000100000000
95	0.000100000000
96	0.000100000000
97	0.000100000000
98	0.000100000000
99	0.000100000000
100	0.000100000000
101	0.000100000000
102	0.000100000000
103	0.000100000000
104	0.000100000000
105	0.000100000000
106	0.000100000000
107	0.000100000000
108	0.000100000000
109	0.000100000000
110	0.000100000000
111	0.000100000000
112	0.000100000000
113	0.000100000000
114	0.000100000000
115	0.000100000000
116	0.000100000000
117	0.000100000000
118	0.000100000000
119	0.000100000000
120	0.000100000000
121	0.000100000000
122	0.000100000000
123	0.000100000000
124	0.000100000000
125	0.000100000000
126	0.000100000000
127	0.000100000000
128	0.000100000000
129	0.000100000000
130	0.000100000000
131	0.000100000000
132	0.000100000000
133	0.000100000000
134	0.000100000000
135	0.000100000000
136	0.000100000000
137	0.000100000000
138	0.000100000000
139	0.000100000000
140	0.000100000000
141	0.000100000000
142	0.000100000000
143	0.000100000000
144	0.000100000000
145	0.000100000000
146	0.000100000000
147	0.000100000000
148	0.000100000000
149	0.000100000000
150	0.000100000000
151	0.000100000000
152	0.000100000000
153	0.000100000000
154	0.000100000000
155	0.000100000000
156	0.000100000000
157	0.000100000000
158	0.000100000000
159	0.000100000000
160	0.000100000000
161	0.000100000000
162	0.000100000000
163	0.000100000000
164	0.000100000000
165	0.000100000000
166	0.000100000000
167	0.000100000000
168	0.000100000000
169	0.000100000000
170	0.000100000000
171	0.000100000000
172	0.000100000000
173	0.000100000000
174	0.000100000000
175	0.000100000000
176	0.000100000000
177	0.000100000000
178	0.000100000000
179	0.000100000000
180	0.000100000000
181	0.000100000000
182	0.000100000000
183	0.000100000000
184	0.000100000000
185	0.000100000000
186	0.000100000000
187	0.000100000000
188	0.000100000000
189	0.000100000000
190	0.000100000000
191	0.000100000000
192	0.000100000000
193	0.000100000000
194	0.000100000000
195	0.000100000000
196	0.000100000000
197	0.000100000000
198	0.000100000000
199	0.000100000000
200	0.000100000000
201	0.000100000000
202	0.000100000000
203	0.000100000000
204	0.000100000000
205	0.000100000000
206	0.000100000000
207	0.000100000000
208	0.000100000000
209	0.000100000000
210	0.000100000000
211	0.000100000000
212	0.000100000000
213	0.000100000000
214	0.000100000000
215	0.000100000000
216	0.000100000000
217	0.000100000000
218	0.000100000000
219	0.000100000000
220	0.000100000000
221	0.000100000000
222	0.000100000000
223	0.000100000000
224	0.000100000000
225	0.000100000000
226	0.000100000000
227	0.000100000000
228	0.000100000000
229	0.000100000000
230	0.000100000000
231	0.000100000000
232	0.000100000000
233	0.000100000000
234	0.000100000000
235	0.000100000000
236	0.000100000000
237	0.000100000000
238	0.000100000000
239	0.000100000000
240	0.000100000000
241	0.000100000000
242	0.000100000000
243	0.000100000000
244	0.000100000000
245	0.000100000000
246	0.000100000000
247	0.000100000000
248	0.000100000000
249	0.000100000000
250	0.000100000000
251	0.000100000000
252	0.000100000000
253	0.000100000000
254	0.000100000000
255	0.000100000000
256	0.000100000000
257	0.000100000000
258	0.000100000000
259	0.000100000000
260	0.000100000000
261	0.000100000000
262	0.000100000000
263	0.000100000000
264	0.000100000000
265	0.000100000000
266	0.000100000000
267	0.000100000000
268	0.000100000000
269	0.000100000000
270	0.000100000000
271	0.000100000000
272	0.000100000000
273	0.000100000000
274	0.000100000000
275	0.000100000000
276	0.000100000000
277	0.000100000000
278	0.000100000000
279	0.000100000000
280	0.000100000000
281	0.000100000000
282	0.000100000000
283	0.000100000000
284	0.000100000000
285	0.000100000000
286	0.000100000000
287	0.000100000000
288	0.000100000000
289	0.000100000000
290	0.000100000000
291	0.000100000000
292	0.000100000000
293	0.000100000000
294	0.000100000000
295	0.000100000000
296	0.000100000000
297	0.000100000000
298	0.000100000000
299	0.000100000000
300	0.000100000000
301	0.000100000000
302	0.000100000000
303	0.000100000000
304	0.000100000000
305	0.000100000000
306	0.000100000000
307	0.000100000000
308	0.000100000000
309	0.000100000000
310	0.000100000000
311	0.000100000000
312	0.000100000000
313	0.000100000000
314	0.000100000000
315	0.000100000000
316	0.000100000000
317	0.000100000000
318	0.000100000000
319	0.000100000000
320	0.000100000000
321	0.000100000000
322	0.000100000000
323	0.000100000000
324	0.000100000000
325	0.000100000000
326	0.000100000000
327	0.000100000000
328	0.000100000000
329	0.000100000000
330	0.000100000000
331	0.000100000000
332	0.000100000000
333	0.000100000000
334	0.000100000000
335	0.000100000000
336	0.000100000000
337	0.000100000000
338	0.000100000000
339	0.000100000000
340	0.000100000

5. Calculate cumulative likes based on views

```
SELECT
    track, views,
        SUM(likes) over (ORDER BY views DESC ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) as Cumulative_likes
FROM spotify;
```

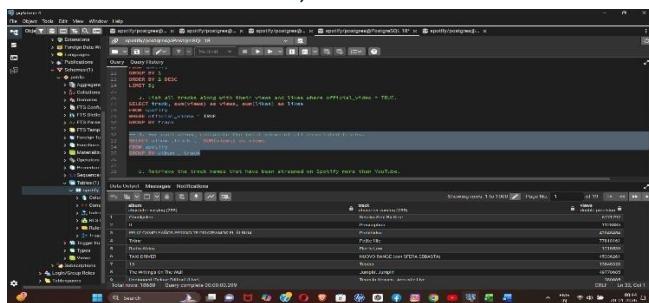


track	views	Cumulative_likes
... (list of tracks)	1000000	1000000
... (list of tracks)	900000	1900000
... (list of tracks)	800000	2700000
... (list of tracks)	700000	3400000
... (list of tracks)	600000	4000000
... (list of tracks)	500000	4500000
... (list of tracks)	400000	4900000
... (list of tracks)	300000	5200000
... (list of tracks)	200000	5400000
... (list of tracks)	100000	5500000
... (list of tracks)	50000	5550000
... (list of tracks)	25000	5575000
... (list of tracks)	10000	5585000
... (list of tracks)	5000	5590000
... (list of tracks)	2000	5592000
... (list of tracks)	1000	5593000
... (list of tracks)	500	5593500
... (list of tracks)	200	5593700
... (list of tracks)	100	5593800
... (list of tracks)	50	5593850
... (list of tracks)	25	5593875
... (list of tracks)	12	5593887
... (list of tracks)	6	5593893
... (list of tracks)	3	5593896
... (list of tracks)	1	5593897
... (list of tracks)	0	5593897

I used a window function with `SUM(likes)` ordered by views in descending order to calculate cumulative likes as a running total from the most viewed track to the current one.

6. For each album, calculate the total views of all associated tracks.

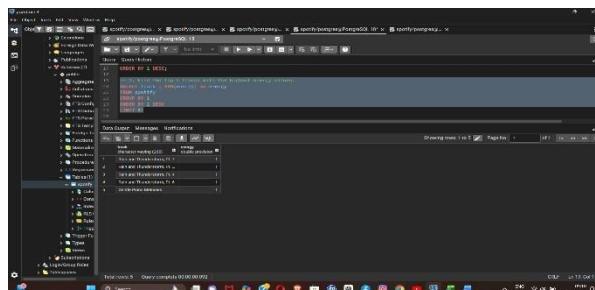
```
SELECT album ,track , SUM(views) as views
FROM spotify
GROUP BY album , track
```



album	track	views
... (list of albums)	... (list of tracks)	1000000
... (list of albums)	... (list of tracks)	900000
... (list of albums)	... (list of tracks)	800000
... (list of albums)	... (list of tracks)	700000
... (list of albums)	... (list of tracks)	600000
... (list of albums)	... (list of tracks)	500000
... (list of albums)	... (list of tracks)	400000
... (list of albums)	... (list of tracks)	300000
... (list of albums)	... (list of tracks)	200000
... (list of albums)	... (list of tracks)	100000
... (list of albums)	... (list of tracks)	50000
... (list of albums)	... (list of tracks)	25000
... (list of albums)	... (list of tracks)	10000
... (list of albums)	... (list of tracks)	5000
... (list of albums)	... (list of tracks)	2000
... (list of albums)	... (list of tracks)	1000
... (list of albums)	... (list of tracks)	500
... (list of albums)	... (list of tracks)	250
... (list of albums)	... (list of tracks)	125
... (list of albums)	... (list of tracks)	62
... (list of albums)	... (list of tracks)	31
... (list of albums)	... (list of tracks)	15.5
... (list of albums)	... (list of tracks)	7.75
... (list of albums)	... (list of tracks)	3.875
... (list of albums)	... (list of tracks)	1.9375
... (list of albums)	... (list of tracks)	0.96875

7. Find the top 5 tracks with the highest energy values.

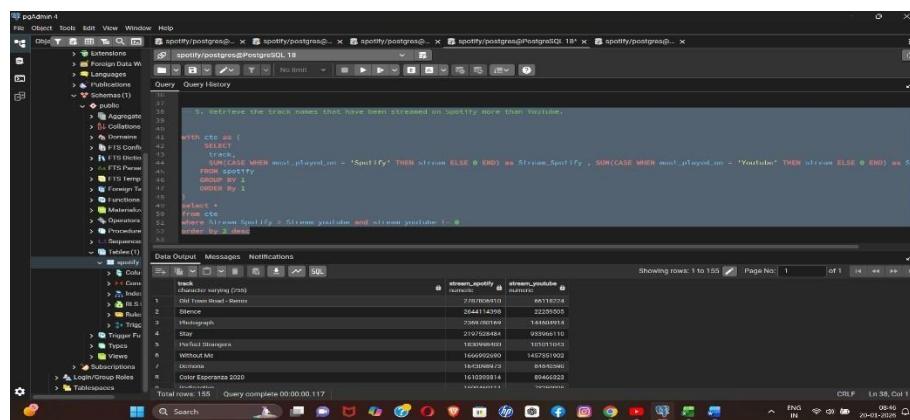
```
SELECT track , AVG(energy) as energy
FROM spotify
GROUP BY 1
ORDER BY 2 DESC
LIMIT 5;
```



track	energy
... (list of tracks)	1.0
... (list of tracks)	0.9999999999999999

8. Retrieve the track names that have been streamed on Spotify more than YouTube.

```
with cte as (
    SELECT      track,
    SUM(CASE WHEN most_played_on = 'Spotify' THEN stream ELSE 0 END) as Stream_Spotify , SUM(CASE WHEN most_played_on = 'Youtube' THEN stream ELSE 0 END) as Stream_Youtube
    FROM spotify
    GROUP BY 1
    ORDER By 1
)
select * from cte
where Stream_Spotify > Stream_youtube and stream_youtube != 0 order
by 2 desc
```



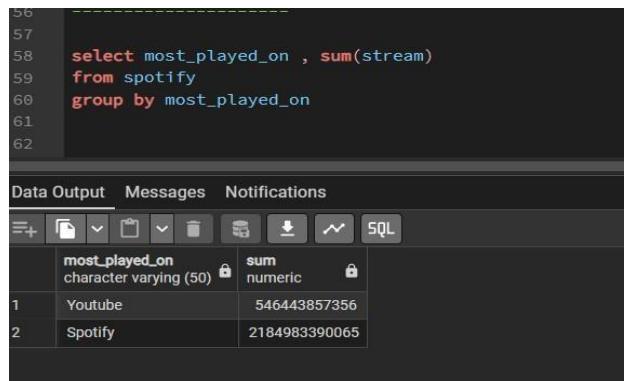
The screenshot shows the pgAdmin 4 interface with multiple tabs open. The main tab displays the SQL query for retrieving tracks with more Spotify streams than YouTube streams. Below the query, the Data Output tab shows the results, which include 155 rows. The columns are track, stream_spotify, and stream_youtube. The results show various tracks like 'Call Me Irresponsible', 'Silence', 'Photograph', 'Stay', 'I'm Gonna Be (500 Miles)', 'Without Me', 'Decimus', and 'Color Esperanza 2020'. The Spotify streams column has values ranging from 2184983390065 to 270580510, while the YouTube streams column has values ranging from 0 to 22239505.

track	stream_spotify	stream_youtube
Call Me Irresponsible	270580510	48718724
Silence	264411498	22239505
Photograph	276610779	14446016
Stay	2191526484	93596110
I'm Gonna Be (500 Miles)	1155389390	10339930
Without Me	1665959590	142751902
Decimus	161030989473	84840260
Color Esperanza 2020	1610309814	89465929

I used conditional aggregation to calculate Spotify and YouTube streams per track, grouped by track, and then filtered tracks where Spotify streams exceed YouTube streams.

Check Which platform have most streams :

```
select most_played_on , sum(stream) as stream
from spotify
group by most_played_on
```



The screenshot shows the pgAdmin 4 interface with multiple tabs open. The main tab displays the SQL query for calculating the total streams per platform. Below the query, the Data Output tab shows the results, which include 2 rows. The columns are most_played_on and sum. The results show 'Youtube' with a sum of 546443857356 and 'Spotify' with a sum of 2184983390065.

most_played_on	sum
Youtube	546443857356
Spotify	2184983390065

KEY INSIGHTS GENERATED

1. Top 3 Most-Viewed Tracks per Artist

Insight:

- Artists usually have a small number of tracks driving most of their popularity.
- Popularity is not evenly distributed across all tracks.

Business Value:

- Useful for tour setlists and featured content.

2. Tracks with Liveness Above Average

Insight:

- Only a limited subset of tracks have high liveness values.
- High-liveness tracks often resemble live recordings or concert-style performances.

Business Value:

- Useful for creating live-performance playlists.
- Can guide decisions on live session releases and concert promotions.

3. Energy Variation Within Albums

Insight:

- Some albums show high energy variation, meaning they mix calm and intense tracks.
- Other albums are energy-consistent, maintaining a similar mood throughout.

Business Value:

- Helps identify albums best suited for:
 - Workout playlists (high, consistent energy)
 - Story-telling or concept albums

4. Tracks with Energy-to-Liveness Ratio > 1.2

Insight:

- These tracks are high-energy but studio-produced, not live.

Business Value:

- Ideal candidates for:
 - Commercial promotions and ads **5.**

Top 5 Highest-Energy Tracks

Insight:

- High-energy tracks are usually shorter, faster, and intense.
- They often overlap with dance and party-oriented genres.

Business Value:

- Useful for fitness playlists, festival sets, and high-engagement ads.

6. Cumulative likes based on views

Insight:

- Identify the Most Influential Tracks **Business Value:**
 - Popularity Distribution

7. Total Views per Album

Insight:

- Album popularity is often driven by one or two blockbuster tracks.
- Some albums perform well overall, while others rely on a single hit.

Business Value:

- Supports album-level performance analysis.
- Helps labels decide between single-driven vs album-driven strategies.

8. Spotify vs YouTube Streaming Comparison

Insight:

- Certain tracks perform significantly better on Spotify than YouTube.
- Indicates platform-specific audience behavior.

Business Value:

- Helps optimize platform-specific marketing strategies.
- Useful for deciding where to prioritize exclusive releases

This project demonstrates how SQL analytics can uncover patterns in music popularity, engagement, and platform performance, enabling data-driven decisions for marketing, playlist curation, and content strategy.

Dataset Quality Notes

- Null checks for likes, views, streams
- Handling duplicates if any
- Removed removed tracks with 0 duration

BASED ON STREAM

Artist with most stream :

```
Select artist, sum(stream) as total_stream
from spotify
group by artist
order by 2 desc;
```

The screenshot shows a PostgreSQL terminal window with two panes. The left pane displays the SQL query. The right pane shows the results, which are a table with columns 'artist' and 'total_stream'. The results list various artists and their total stream counts, ordered by total streams in descending order.

artist	total_stream
Post Malone	15251263853
Ed Sheeran	14394881557
Dua Lipa	1340876274
XXXTENTACION	13224351699
The Weeknd	1303197376
Justin Bieber	12097767422
Imagine Dragons	11858310928
Coldplay	11778478236
Khalid	11386839195
Bruno Mars	10897862950
Ariana Grande	10857411888
Billie Eilish	10747172641
Megan Thee Stallion	10722507616

Album with most stream :

```
Select album, sum(stream) as total_stream from
spotify
group by album
order by 2 desc;
```

Track with most stream :

```
Select track, sum(stream) as
total_stream from spotify group by track
order by 2 desc;
```

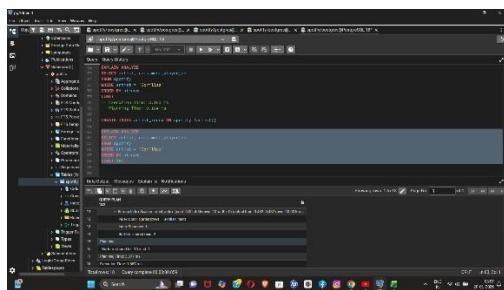
The screenshot shows a PostgreSQL terminal window with two panes. The left pane displays the SQL query. The right pane shows the results, which are a table with columns 'track' and 'total_stream'. The results list various tracks and their total stream counts, ordered by total streams in descending order.

track	total_stream
Closer	5465364999
Can't Hold Us (feat. Ray Dalton)	5225629104
Sunflower - Spider-Man: Into the Spider-Verse	5076659598
Happier	4737615430
STAY (with Justin Bieber)	4731555010
Serfotia	4672439700
The Middle	4566682832
Eastside (with Halsey & Khalid)	4274887533
lovely (with Khalid)	4221147558
Enemy (with JID) - from the series Arcane League of Legends	4182327816
El Ultimo Adiós - Various Artists Version	4074479016
Cold Heart - PNAU Remix	4072975457

QUERY OPTIMIZATION

Before having index :

```
EXPLAIN ANALYZE
SELECT artist,track,most_played_on
FROM spotify
WHERE artist = 'Gorillaz'
ORDER BY stream
LIMIT 10;
```

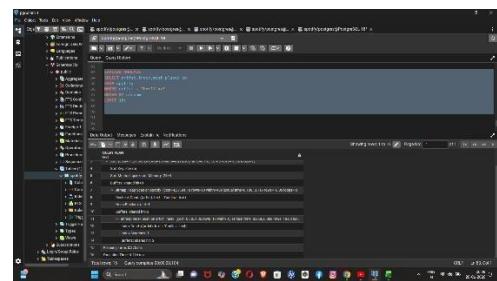


```
-- Execution Time: 3.163 ms
-- Planning Time: 0.164 ms
```

LETS ADD INDEX

```
CREATE INDEX artist_index ON spotify (artist);
```

```
EXPLAIN ANALYZE
SELECT artist,track,most_played_on
FROM spotify
WHERE artist = 'Gorillaz'
ORDER BY stream
LIMIT 10;
```



```
-- Execution Time: 0.375 ms
-- Planning Time: 0.216 ms
```

Insight :

By creating an index on the artist column ,queries filtering by artist became significantly faster.

I optimized queries using indexing, which significantly improved execution speed, reduced resource usage on large music datasets.

SQL Performance Metrics

After query optimization, a small table showing “Before vs After Index”:

Query	Execution Time	Planning Time	Improvement
Original	3.163 ms	0.164 ms	-
After Index	0.375 ms	0.216 ms	88% faster

“Key Takeaways / Recommendations”

After analysing the generated insights , these are some key Takeways:

- **Marketing / Promotions:** Focus campaigns on top tracks or artists with highly engaging tracks.
- **Playlist Curation:** Use high-energy or high-liveness tracks for specific moods.
- **Platform Strategy:** Release exclusive content on Spotify if the track performs better there.
- **Album Strategy:** Identify whether albums are single-driven or evenly popular.

Conclusion

Using SQL, this project analyzed the Spotify dataset to uncover track popularity, audio features, user engagement, and platform performance. Queries were optimized using indexing, and insights generated can guide playlist curation, marketing strategy, and platform-specific decisions.