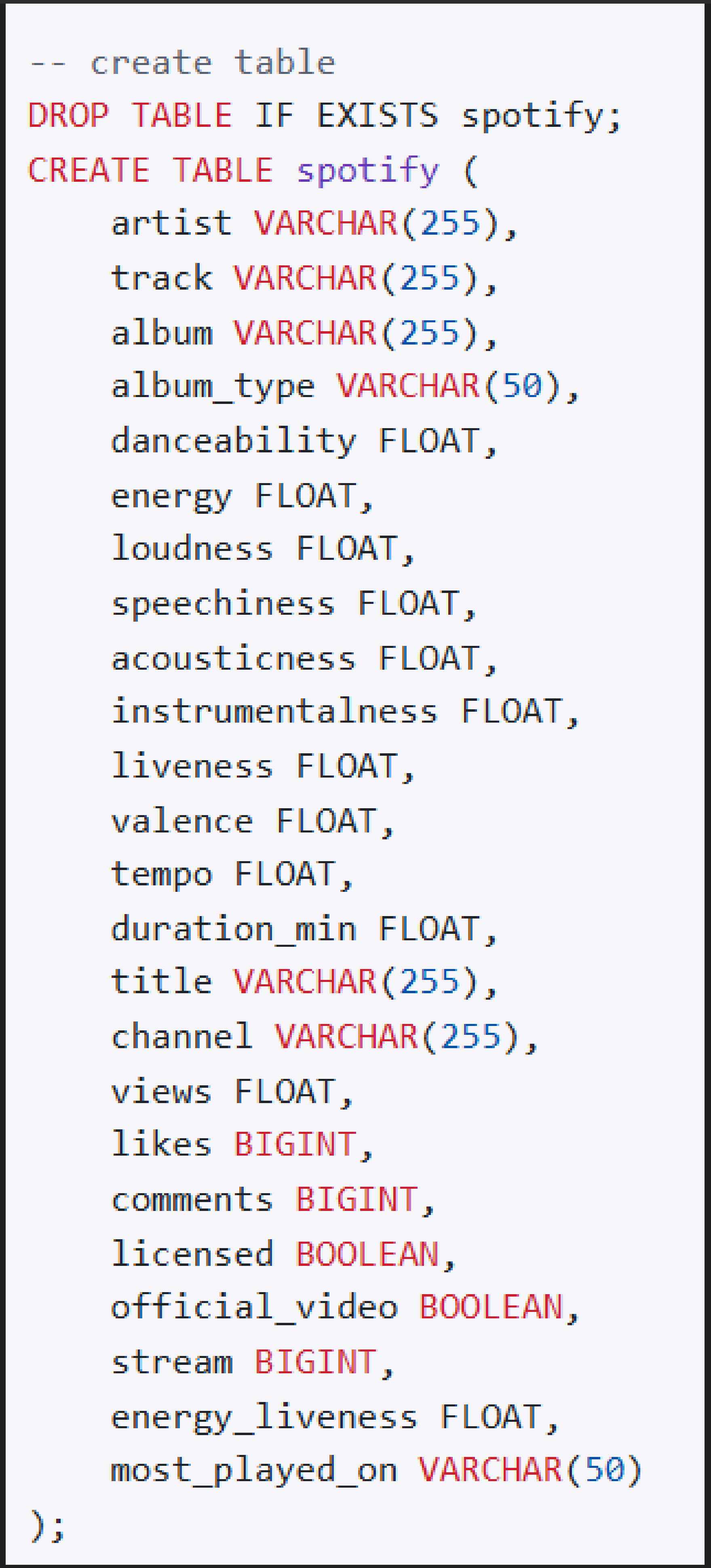
# SPOTIFY ADVANCED SQL PROJECT AND QUERY OPTIMIZATION

Using PostgreSQL

# OVERVIEW

This project involves analyzing a Spotify dataset with various attributes about tracks, albums, and artists using **SQL**. It covers an end-to-end process of normalizing a denormalized dataset, performing SQL queries of varying complexity (easy, medium, and advanced), and optimizing query performance. The primary goals of the project are to practice advanced SQL skills and generate valuable insights from the dataset.

## QUERY



```sql
-- create table
DROP TABLE IF EXISTS spotify;
CREATE TABLE spotify (
    artist VARCHAR(255),
    track VARCHAR(255),
    album VARCHAR(255),
    album_type VARCHAR(50),
    danceability FLOAT,
    energy FLOAT,
    loudness FLOAT,
    speechiness FLOAT,
    acousticness FLOAT,
    instrumentalness FLOAT,
    liveness FLOAT,
    valence FLOAT,
    tempo FLOAT,
    duration_min FLOAT,
    title VARCHAR(255),
    channel VARCHAR(255),
    views FLOAT,
    likes BIGINT,
    comments BIGINT,
    licensed BOOLEAN,
    official_video BOOLEAN,
    stream BIGINT,
    energy_liveness FLOAT,
    most_played_on VARCHAR(50)
);
```

# PROJECT STEPS

1. **Data Exploration**

Before diving into SQL, it’s important to understand the dataset thoroughly. The dataset contains attributes such as:

- Artist: The performer of the track.
- Track: The name of the song.
- Album: The album to which the track belongs.
- Album_type: The type of album (e.g., single or album).
- Various metrics such as danceability, energy, loudness, tempo, and more.

**2. Querying the Data**

After the data is inserted, various SQL queries can be written to explore and analyze the data. Queries are categorized into **easy**, **medium**, and **advanced** levels to help progressively develop SQL proficiency.

- Easy Queries
- Medium Queries
- Advanced Queries

**3. Query Optimization**

In advanced stages, the focus shifts to improving query performance. Some optimization strategies include:

- **Indexing**: Adding indexes on frequently queried columns.
- **Query Execution Plan**: Using EXPLAIN ANALYZE to review and refine query performance.

# 15 PRACTICE QUESTIONS

01 Easy Level

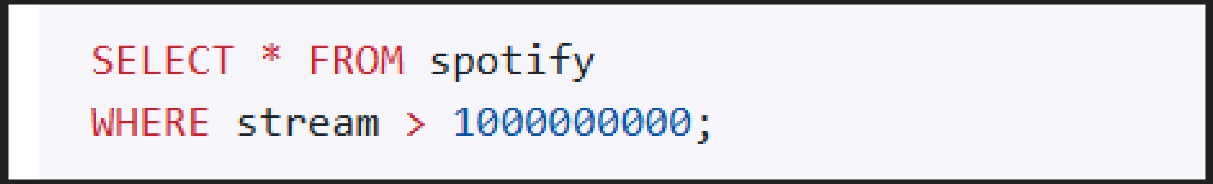
02 Medium Level

03 Advanced Level

# EASY LEVEL

**01**

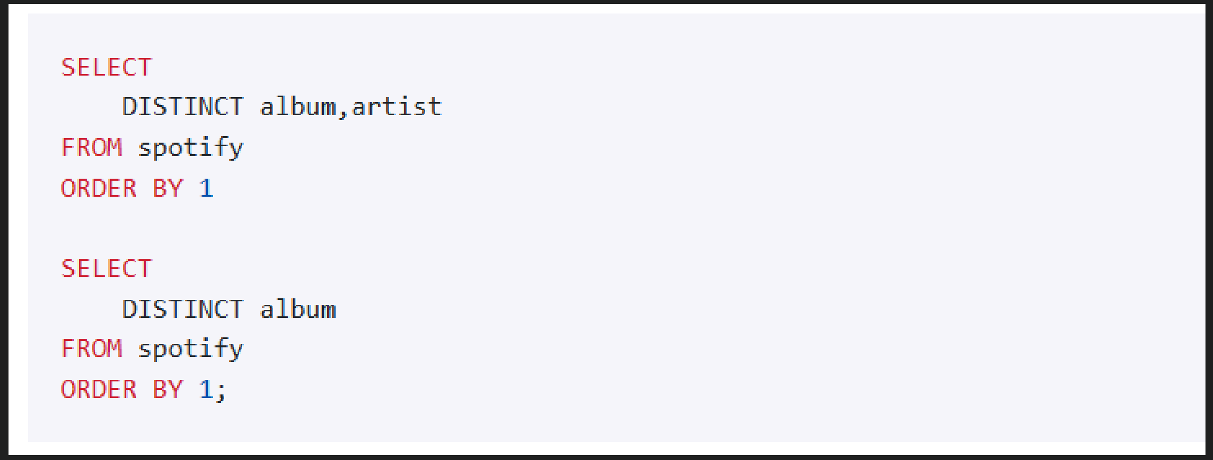
Retrieve the names of all tracks that have more than 1 billion streams.



```sql
SELECT * FROM spotify
WHERE stream > 1000000000;
```

**02**

List all albums along with their respective artists.

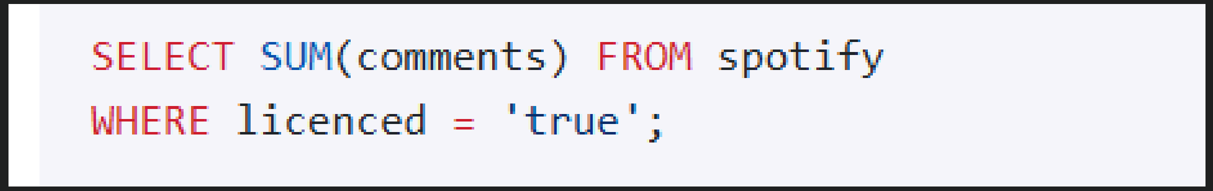


```sql
SELECT
    DISTINCT album,artist
FROM spotify
ORDER BY 1

SELECT
    DISTINCT album
FROM spotify
ORDER BY 1;
```

**03**

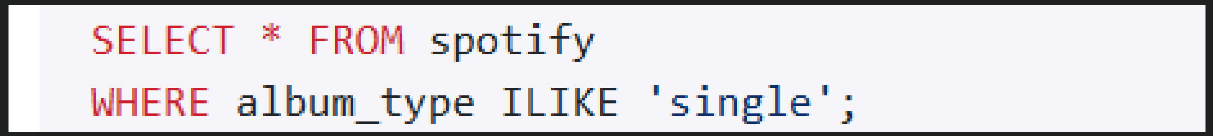
Get the total number of comments for tracks where `licensed = TRUE`.



```sql
SELECT SUM(comments) FROM spotify
WHERE licenced = 'true';
```

**04**

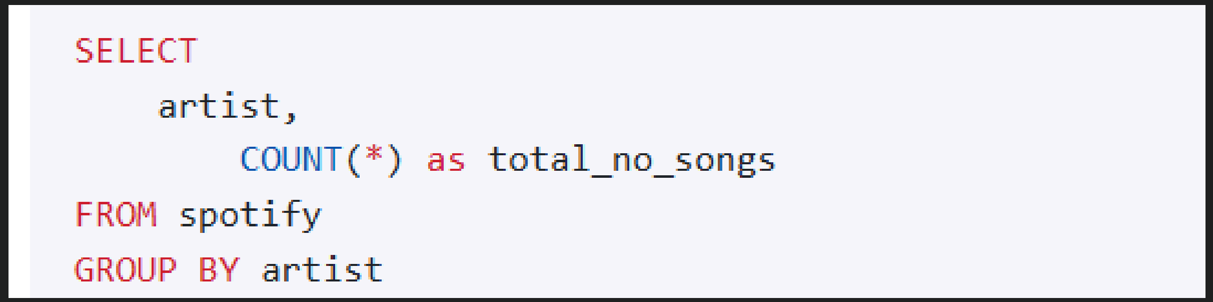
Find all tracks that belong to the album type `single`.



```sql
SELECT * FROM spotify
WHERE album_type ILIKE 'single';
```

**05**

Count the total number of tracks by each artist.

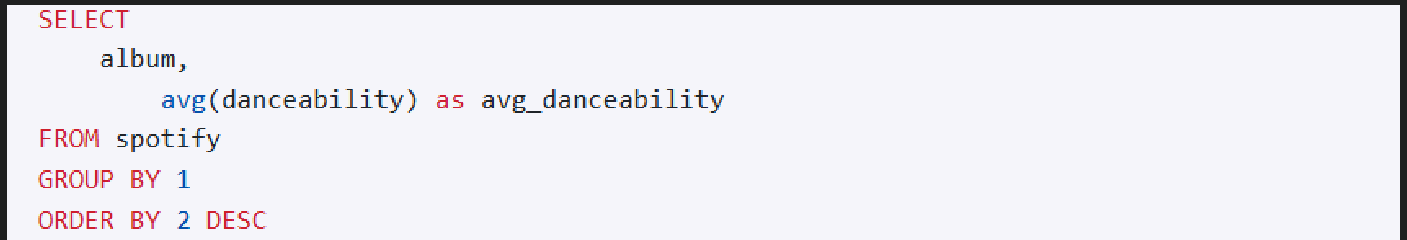


```sql
SELECT
    artist,
        COUNT(*) as total_no_songs
FROM spotify
GROUP BY artist
```

# MEDIUM LEVEL

**01**

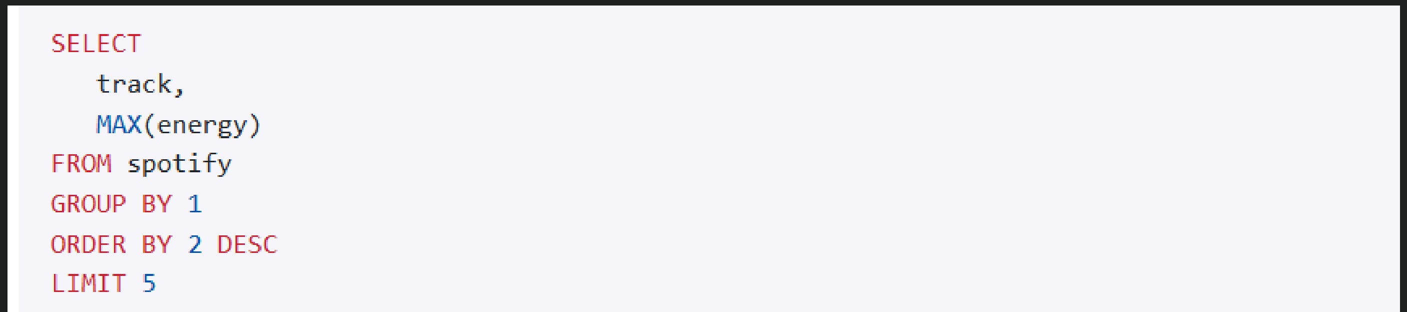
Calculate the average danceability of tracks in each album.



```
SELECT
    album,
        avg(danceability) as avg_danceability
FROM spotify
GROUP BY 1
ORDER BY 2 DESC
```

**02**

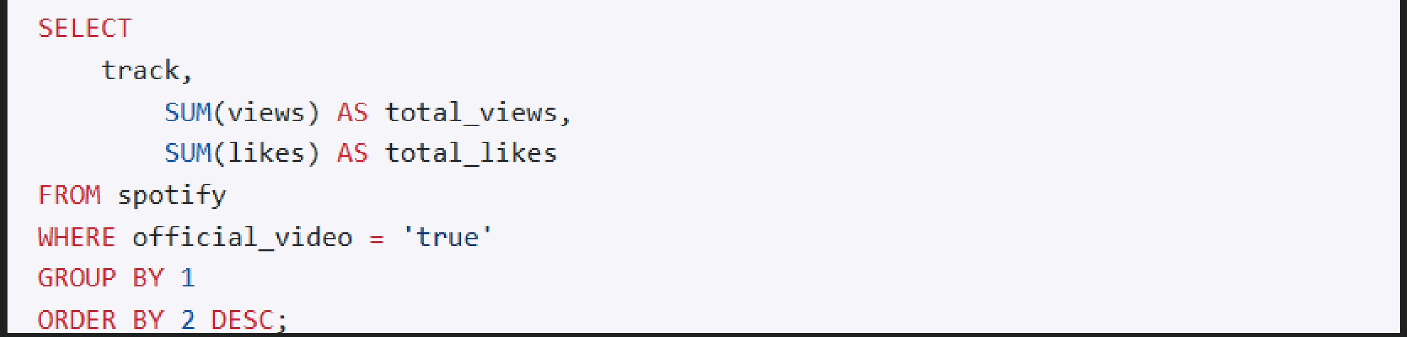
Find the top 5 tracks with the highest energy values.



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

**03**

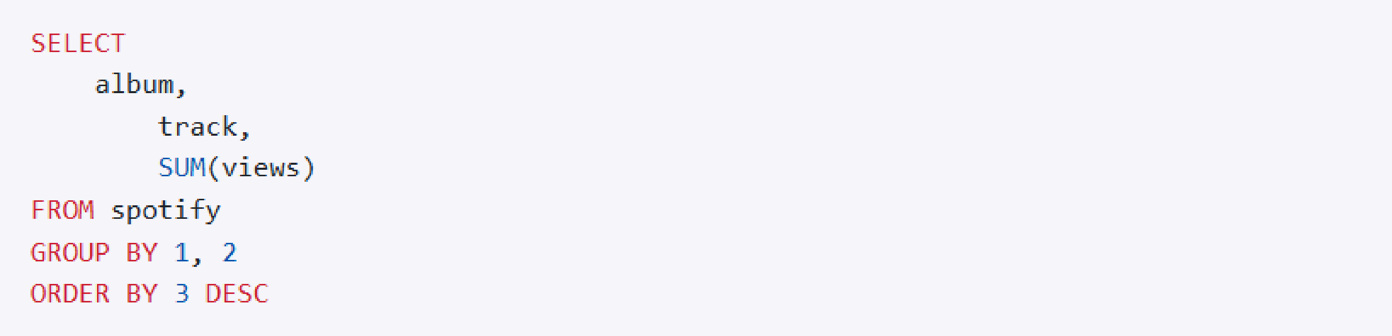
List all tracks along with their views and likes where `official_video = TRUE`.



```
SELECT
    track,
        SUM(views) AS total_views,
        SUM(likes) AS total_likes
FROM spotify
WHERE official_video = 'true'
GROUP BY 1
ORDER BY 2 DESC;
```

**04**

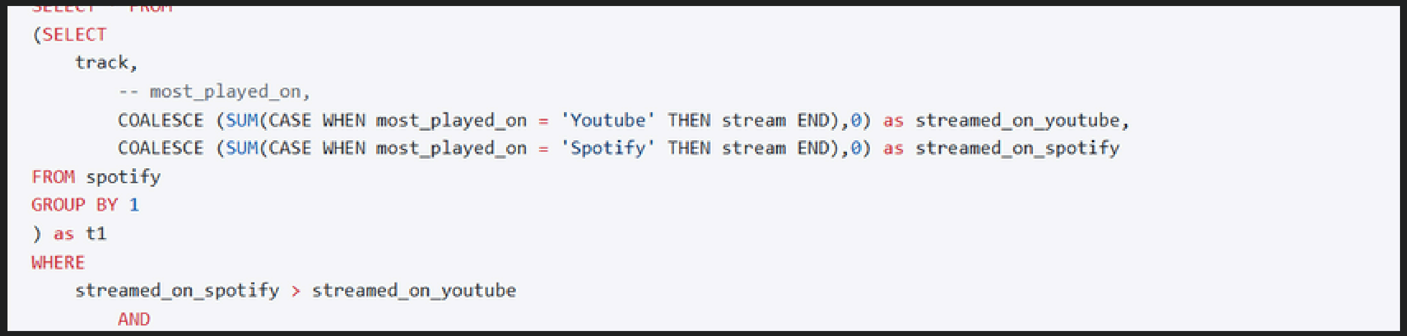
For each album, calculate the total views of all associated tracks.



```
SELECT
    album,
        track,
        SUM(views)
FROM spotify
GROUP BY 1, 2
ORDER BY 3 DESC
```

**05**

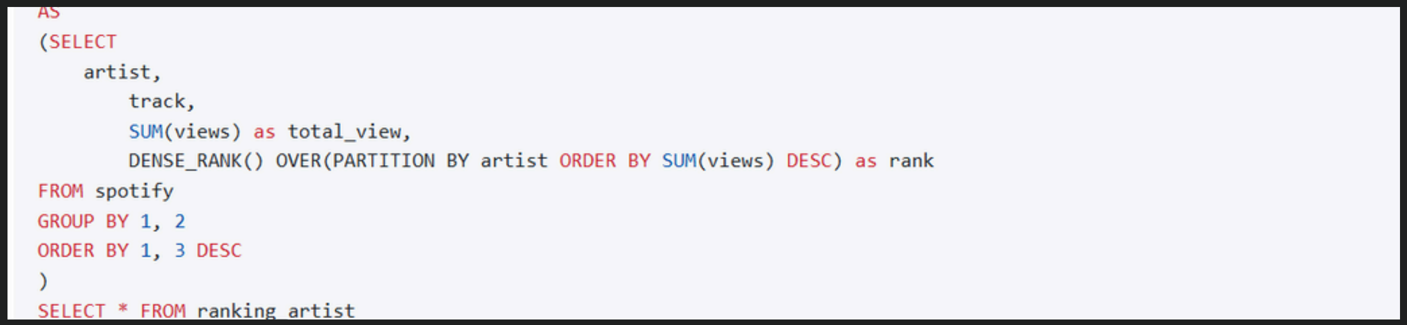
Retrieve the track names that have been streamed on Spotify more than YouTube.



```
SELECT * FROM
(SELECT
    track,
        -- most_played_on,
        COALESCE (SUM(CASE WHEN most_played_on = 'Youtube' THEN stream END),0) as streamed_on_youtube,
        COALESCE (SUM(CASE WHEN most_played_on = 'Spotify' THEN stream END),0) as streamed_on_spotify
FROM spotify
GROUP BY 1
) as t1
WHERE
    streamed_on_spotify > streamed_on_youtube
        AND
```

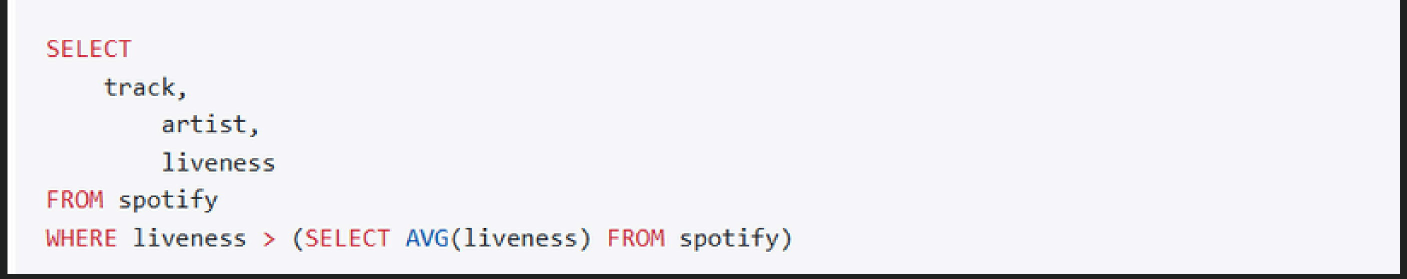
# ADVANCED LEVEL

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



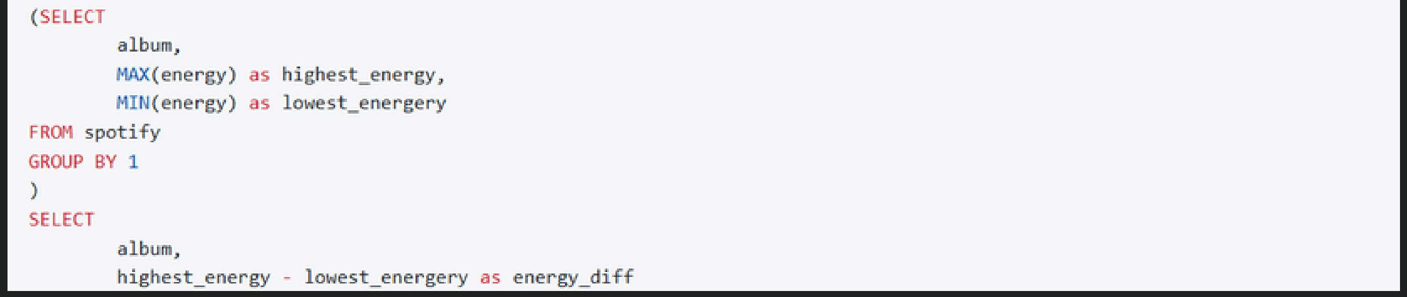
```
AS
(SELECT
    artist,
        track,
        SUM(views) as total_view,
        DENSE_RANK() OVER(PARTITION BY artist ORDER BY SUM(views) DESC) as rank
FROM spotify
GROUP BY 1, 2
ORDER BY 1, 3 DESC
)
SELECT * FROM ranking_artist
```

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



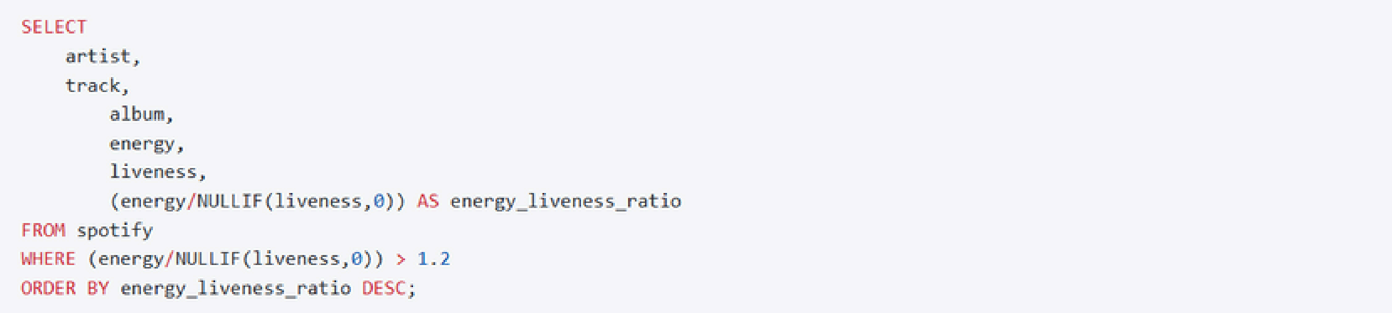
```
SELECT
    track,
        artist,
        liveness
FROM spotify
WHERE liveness > (SELECT AVG(liveness) FROM spotify)
```

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



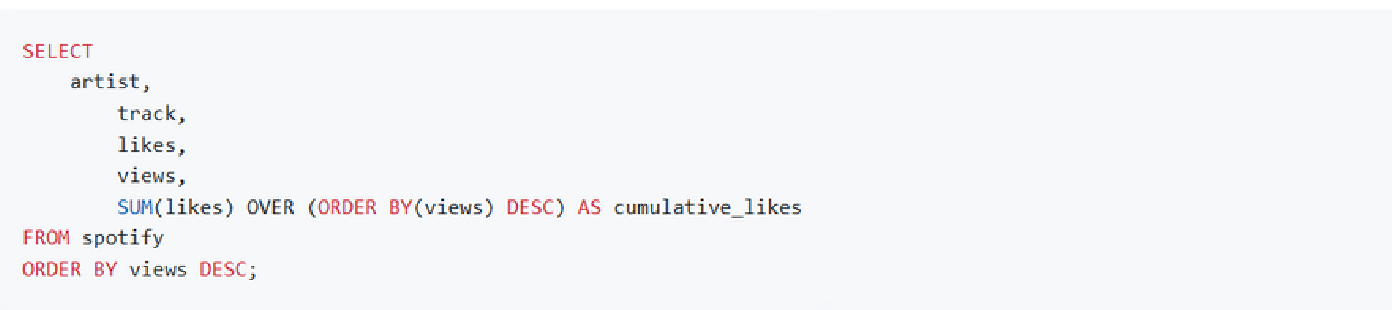
```
(SELECT
        album,
        MAX(energy) as highest_energy,
        MIN(energy) as lowest_energery
FROM spotify
GROUP BY 1
)
SELECT
        album,
        highest_energy - lowest_energery as energy_diff
```

04 Find tracks where the energy-to-liveness ratio is greater than 1.2.



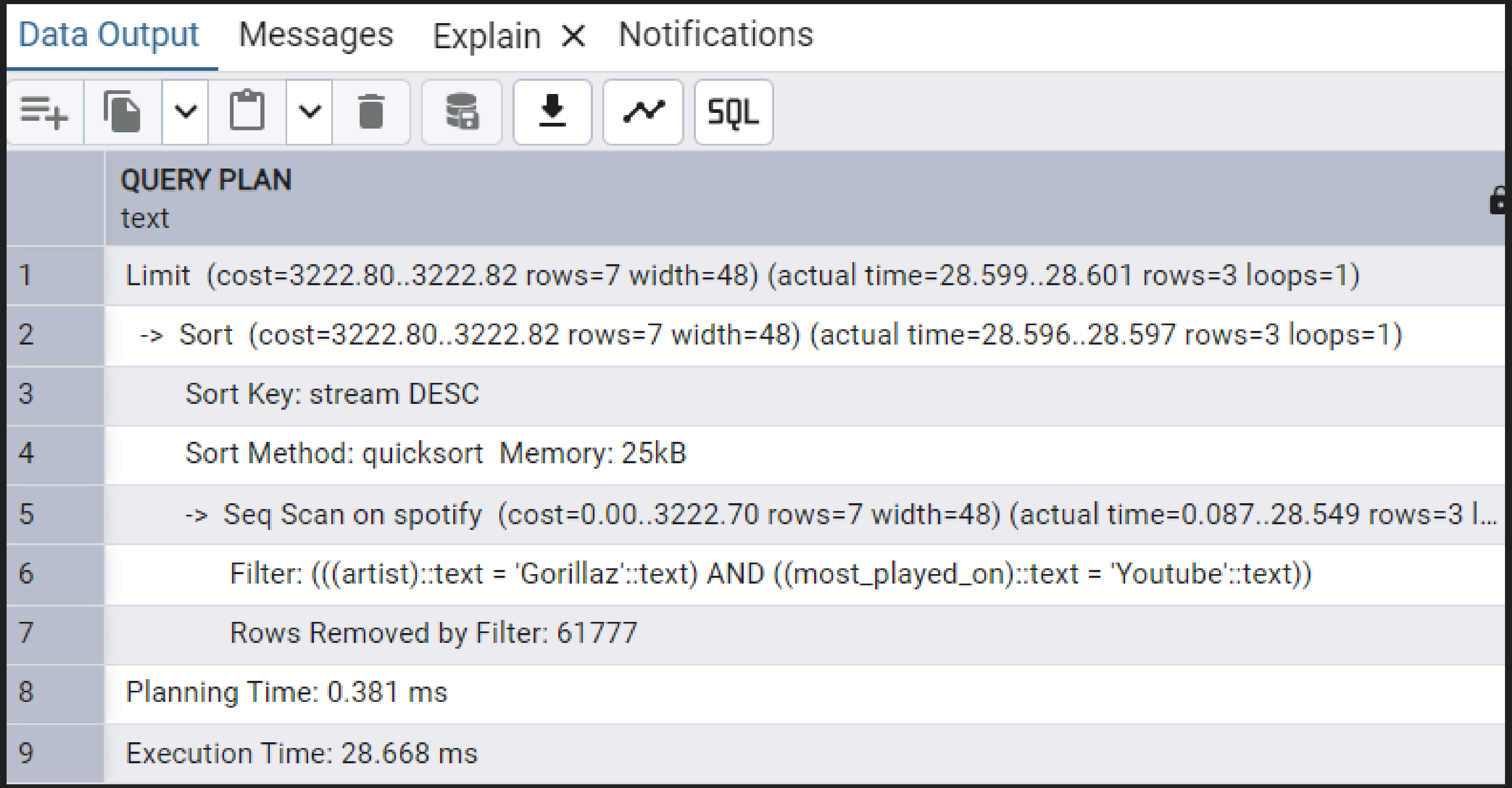
```
SELECT
    artist,
    track,
        album,
        energy,
        liveness,
        (energy/NULLIF(liveness,0)) AS energy_liveness_ratio
FROM spotify
WHERE (energy/NULLIF(liveness,0)) > 1.2
ORDER BY energy_liveness_ratio DESC;
```

05 Calculate the cumulative sum of likes for tracks ordered by the number of views, using window functions.



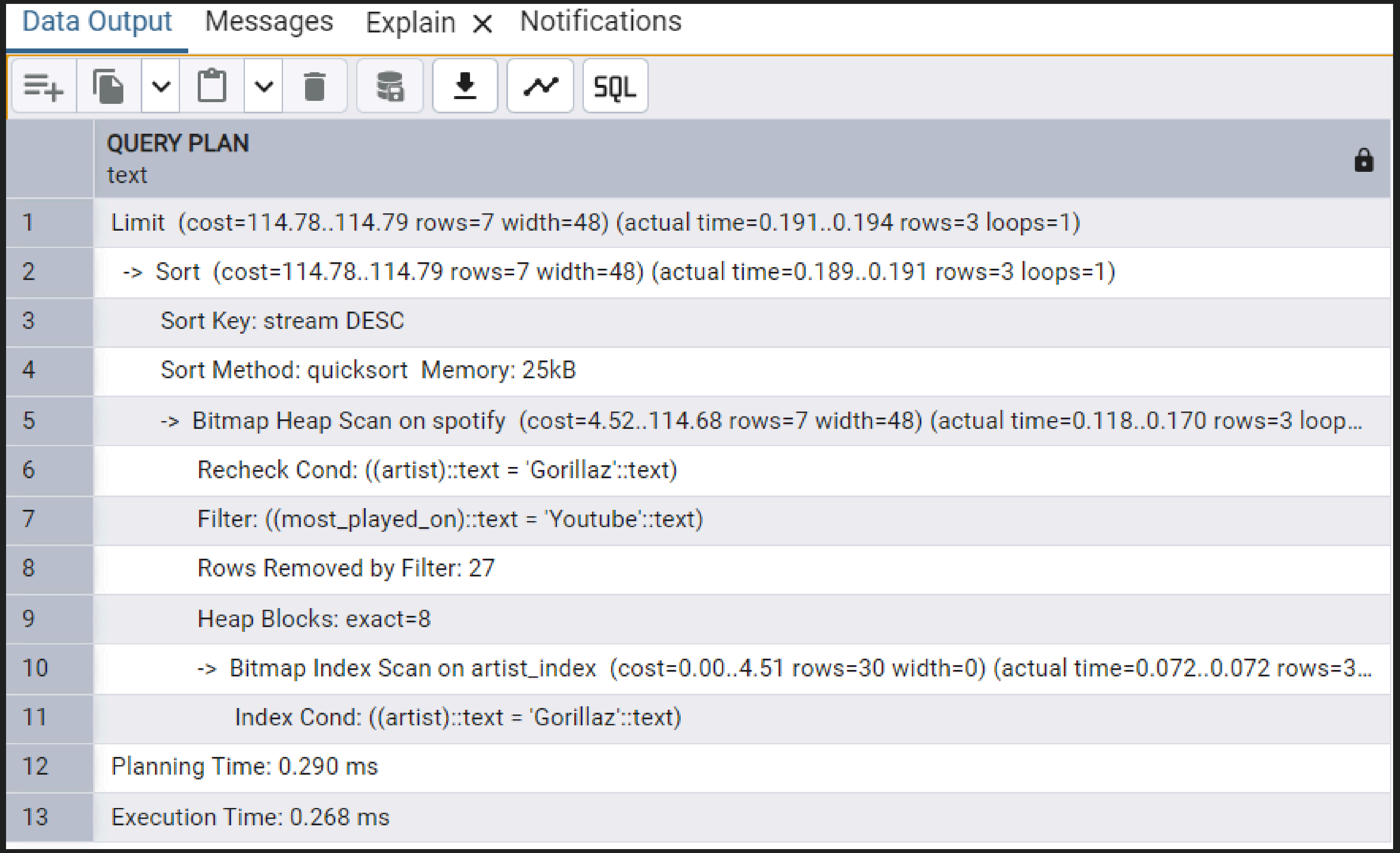
```
SELECT
    artist,
        track,
        likes,
        views,
        SUM(likes) OVER (ORDER BY(views) DESC) AS cumulative_likes
FROM spotify
ORDER BY views DESC;
```

## Initial Query Performance Analysis Using EXPLAIN



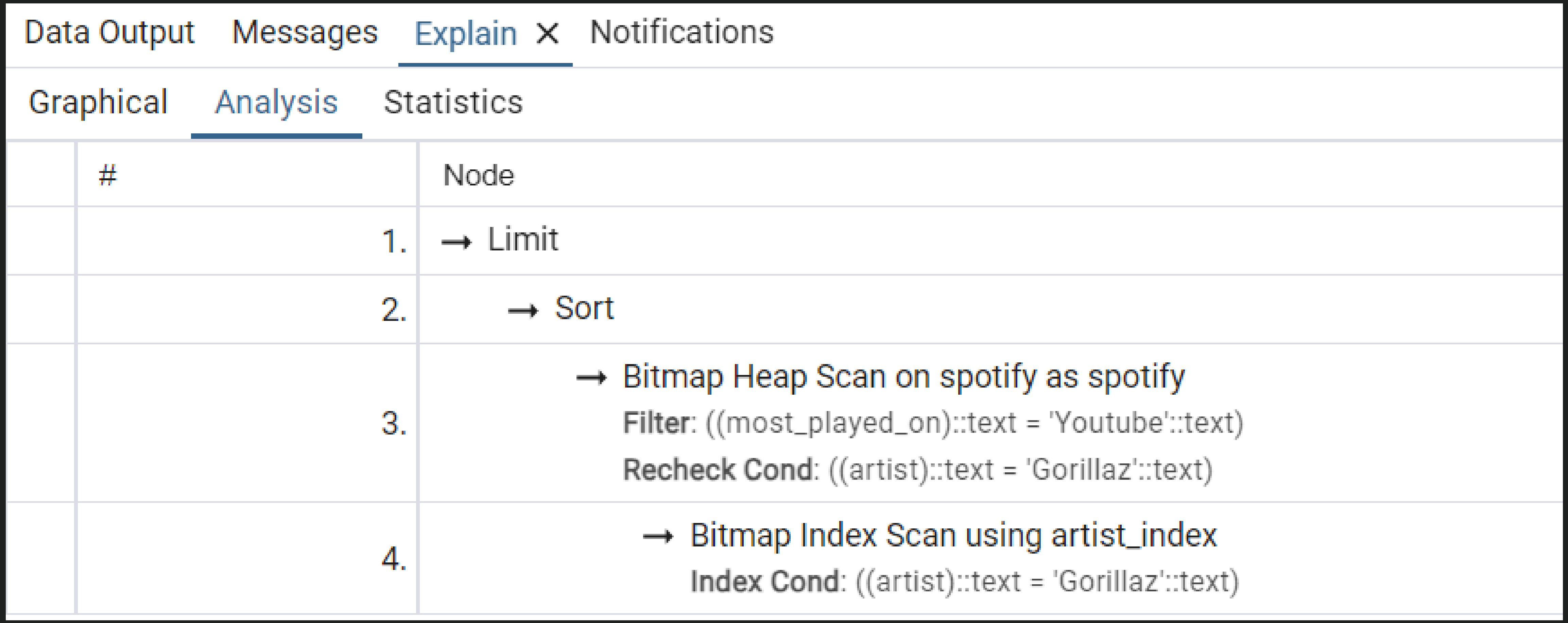
```
     QUERY PLAN
     text
1    Limit  (cost=3222.80..3222.82 rows=7 width=48) (actual time=28.599..28.601 rows=3 loops=1)
2      ->  Sort  (cost=3222.80..3222.82 rows=7 width=48) (actual time=28.596..28.597 rows=3 loops=1)
3            Sort Key: stream DESC
4            Sort Method: quicksort  Memory: 25kB
5            ->  Seq Scan on spotify  (cost=0.00..3222.70 rows=7 width=48) (actual time=0.087..28.549 rows=3 l...
6                  Filter: (((artist)::text = 'Gorillaz'::text) AND ((most_played_on)::text = 'Youtube'::text))
7                  Rows Removed by Filter: 61777
8    Planning Time: 0.381 ms
9    Execution Time: 28.668 ms
```

## Performance Analysis After Index Creation



```
     QUERY PLAN
     text
1    Limit  (cost=114.78..114.79 rows=7 width=48) (actual time=0.191..0.194 rows=3 loops=1)
2      ->  Sort  (cost=114.78..114.79 rows=7 width=48) (actual time=0.189..0.191 rows=3 loops=1)
3            Sort Key: stream DESC
4            Sort Method: quicksort  Memory: 25kB
5            ->  Bitmap Heap Scan on spotify  (cost=4.52..114.68 rows=7 width=48) (actual time=0.118..0.170 rows=3 loop...
6                  Recheck Cond: ((artist)::text = 'Gorillaz'::text)
7                  Filter: ((most_played_on)::text = 'Youtube'::text)
8                  Rows Removed by Filter: 27
9                  Heap Blocks: exact=8
10                 ->  Bitmap Index Scan on artist_index  (cost=0.00..4.51 rows=30 width=0) (actual time=0.072..0.072 rows=3...
11                       Index Cond: ((artist)::text = 'Gorillaz'::text)
12   Planning Time: 0.290 ms
13   Execution Time: 0.268 ms
```

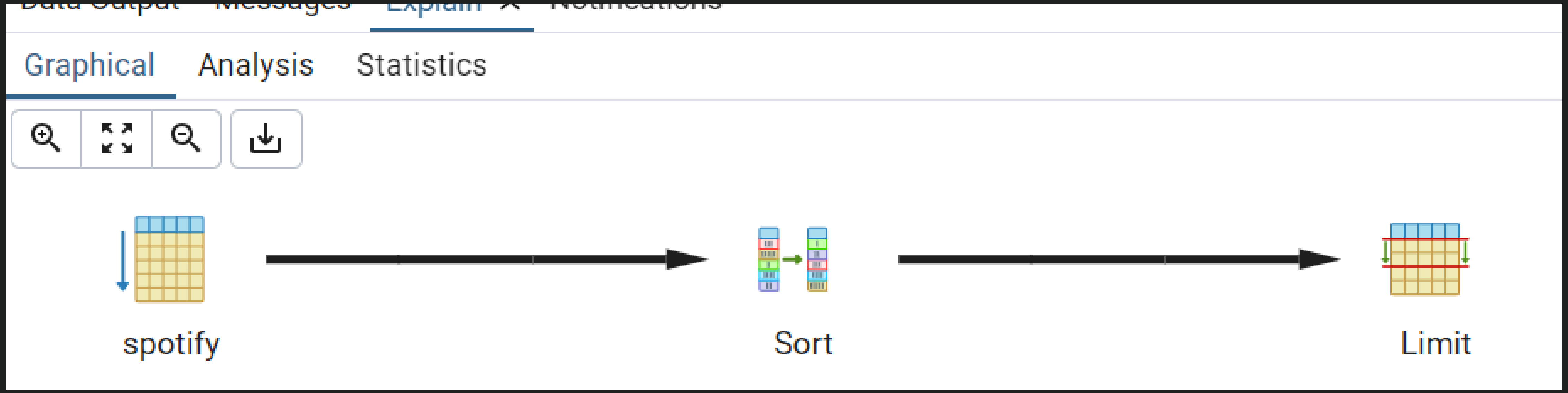
# Graphical Performance Comparison



Data Output Messages Explain × Notifications

Graphical Analysis Statistics

| # | Node |
|---|---|
| 1. | → Limit |
| 2. | → Sort |
| 3. | → Bitmap Heap Scan on spotify as spotify<br>**Filter**: ((most_played_on)::text = 'Youtube'::text)<br>**Recheck Cond**: ((artist)::text = 'Gorillaz'::text) |
| 4. | → Bitmap Index Scan using artist_index<br>**Index Cond**: ((artist)::text = 'Gorillaz'::text) |



Data Output Messages Explain × Notifications

Graphical Analysis Statistics

# THANK YOU