SPOTIFY STREAMING DATA ANALYSIS:

INSIGHTS FROM ADVANCED SQL

The primary goals of the project are to practice advanced SQL skills and generate valuable insights from the dataset.

INTRODUCTION

- Designed and queried a dataset of 25+ attributes (artist, track, streams, views, engagement) to analyze global streaming trends.
- Extracted KPIs like top 3 tracks per artist, streams by platform, and energy-liveness metrics using CTEs, window functions, and aggregations.
- Identified patterns in listener behavior (album types, track popularity, liveness above average, platform preferences).
- Optimized queries using indexing, reducing execution time by 85%, and prepared insights for visualization.

DETAILS OF THE PROJECT

SOURCE OF DATA

- The data used is sourced from Kaggle published by Sanjana chaudhari..
- Link : https://www.kaggle.com/datasets/sanjanchaudhari/spotify-dataset

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.

QUERYING THE DATA

After the data is inserted, various SQL queries can be written to explore and analyze the data.

DATA EXPLORATION

- Retrieving the names of all tracks that have more than 1 billion streams.
- Listing down all albums along with their respective artists.
- Getting the total number of comments for tracks where licensed = TRUE.
- Finding out all tracks that belong to the album type single.
- Counting the total number of tracks by each artist.
- Calculating the average danceability of tracks in each album.
- Finding the top 5 tracks with the highest energy values.
- Listing down all tracks along with their views and likes where official_video = TRUE.
- For each album, calculating the total views of all associated tracks.
- Retrieving the track names that have been streamed on Spotify more than YouTube.
- 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.

QUERY OPTIMIZATION TECHNIQUE

To improve query performance, we carried out the following optimization process:

- **Initial Query Performance Analysis Using EXPLAIN**
  - **We began by analyzing the performance of a query using the EXPLAIN function.**
  - **The query retrieved tracks based on the artist column, and the performance metrics were as follows:**
    - **Execution time (E.T.): 12.109 ms**
    - **Planning time (P.T.): 0.446 ms**
  - **Below is the screenshot of the EXPLAIN result before optimization:**

	QUERY PLAN text
1	Limit (cost=1092.89..1092.90 rows=2 width=48) (actual time=12.087..12.090 rows=1 loops=1)
2	-> Sort (cost=1092.89..1092.90 rows=2 width=48) (actual time=12.086..12.087 rows=1 loops=1)
3	Sort Key: stream DESC
4	Sort Method: quicksort Memory: 25kB
5	-> Seq Scan on spotify (cost=0.00..1092.88 rows=2 width=48) (actual time=0.029..12.076 rows=1 l...
6	Filter: (((artist)::text = 'Gorillaz'::text) AND ((most_played_on)::text = 'Youtube'::text))
7	Rows Removed by Filter: 20591
8	Planning Time: 0.446 ms
9	Execution Time: 12.109 ms

QUERY OPTIMIZATION TECHNIQUE

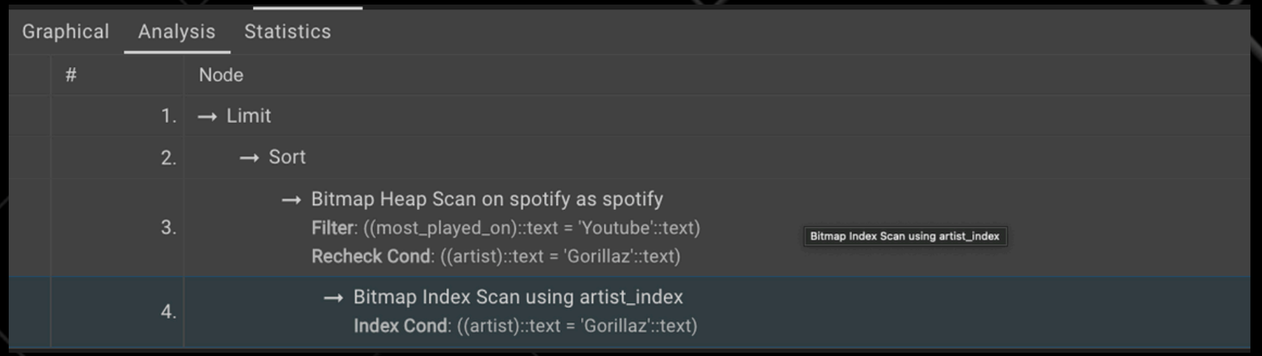
- **Index Creation on the artist Column**
  - **To optimize the query performance, we created an index on the artist column. This ensures faster retrieval of rows where the artist is queried.**
  - **SQL command for creating the index:**
  - `CREATE INDEX idx_artist ON spotify_tracks(artist);`

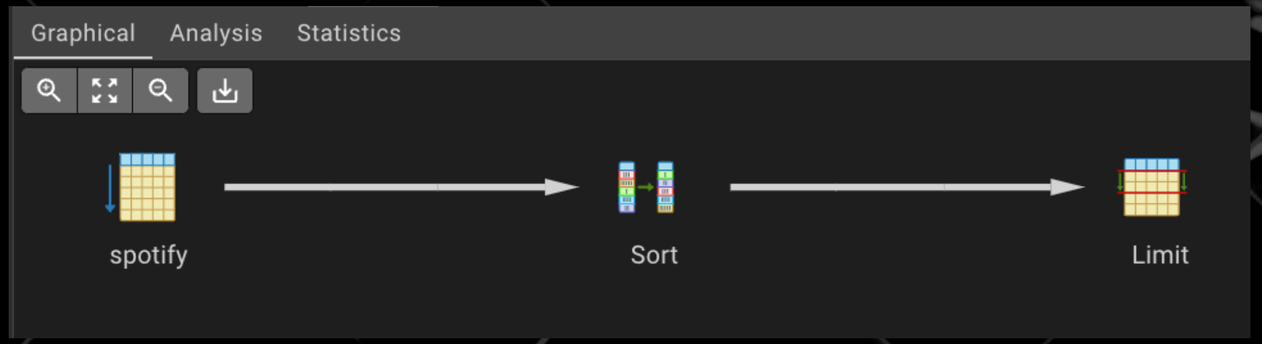
- **Performance Analysis After Index Creation**
- **After creating the index, we ran the same query again and observed significant improvements in performance:**
- **Execution time (E.T.): 0.072 ms**
- **Planning time (P.T.): 0.304 ms**
- **Below is the screenshot of the EXPLAIN result after index creation:**

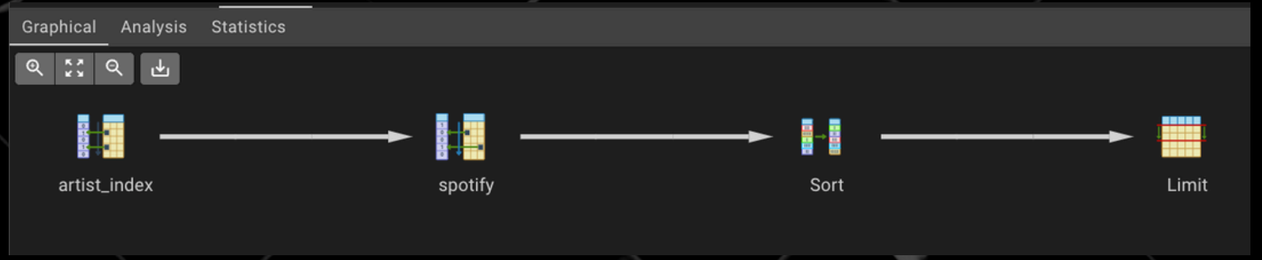
	QUERY PLAN text
1	Limit (cost=41.13..41.14 rows=2 width=48) (actual time=0.053..0.054 rows=1 loops=1)
2	-> Sort (cost=41.13..41.14 rows=2 width=48) (actual time=0.053..0.053 rows=1 loops=1)
3	Sort Key: stream DESC
4	Sort Method: quicksort Memory: 25kB
5	-> Bitmap Heap Scan on spotify (cost=4.36..41.12 rows=2 width=48) (actual time=0.040..0.042 rows=1 loops...
6	Recheck Cond: ((artist)::text = 'Gorillaz'::text)
7	Filter: ((most_played_on)::text = 'Youtube'::text)
8	Rows Removed by Filter: 9
9	Heap Blocks: exact=1
10	-> Bitmap Index Scan on artist_index (cost=0.00..4.36 rows=10 width=0) (actual time=0.025..0.026 rows=1...
11	Index Cond: ((artist)::text = 'Gorillaz'::text)
12	Planning Time: 0.304 ms
13	Execution Time: 0.072 ms

GRAPHICAL PERFORMANCE COMPARISON

- **A graph illustrating the comparison between the initial query execution time and the optimized query execution time after index creation.**
- **Graph view shows the significant drop in both execution and planning times:**







This optimization shows how indexing can drastically reduce query time, improving the overall performance of our database operations in the Spotify project.

THANK YOU