Analyzing Spotify Data To Improve Music Discovery

Sanjukta Baruah
School of Engg. and Applied Sciences
Data Science
Buffalo, United States
sbaruah@buffalo.edu

Priyadarshini Raghavendra
School of Engg. and Applied Sciences
Data Science
Buffalo, United States
praghave@buffalo.edu

Sujay Shrivastava
School of Engg. and Applied Sciences
Data Science
Buffalo, United States
sujayshr@buffalo.edu

***Abstract*—The initial dataset underwent a normalization process, evolving into a schema with diverse relations and relation types, unleashing powerful capabilities for insightful data analysis and a deeper understanding of the complex musical universe within Spotify's expansive database**

I. Problem Statement

The music industry stands at the cusp of transformation, with streaming platforms like Spotify fundamentally altering the way people engage with and consume music. In this dynamic landscape, understanding the nuanced intricacies of user behavior, preferences, and industry trends is pivotal. Spotify, as one of the most prominent music streaming services, accumulates vast datasets encompassing user interactions, playlists, music attributes, and more. The question at hand is how to unlock the full potential of this data goldmine. To effectively address this challenge, we propose the creation of a comprehensive and normalized database, underpinned by a well-structured Entity-Relationship (E/R) diagram. The database will serve as the bedrock for a holistic analysis, uncovering insights, trends, and correlations across a multitude of dimensions.

II. Data Generation and Expansion for Enhanced Query Interpretability and Optimization

In our project, we initially generated raw data from scratch using Python's random string generation functions. However, to transform this data into a more meaningful format and enhance query interpretability, we leveraged Faker.py. Additionally, in a separate effort, we expanded the size of the Users and tracks tables by tenfold. This expansion was undertaken to facilitate the optimization of queries. It's important to note that the ER diagram remains unchanged throughout these data enhancements.

III. Queries to solve the Business Problems

A. *Most popular genres in different regions*

<u>Before Optimization</u>

```
WITH UserTrackGenre AS (
    SELECT "user"."User_Region", "tracks"."Track_Genre",
           COUNT(*) AS play_count,
           ROW_NUMBER() OVER (
               PARTITION BY "user"."User_Region"
               ORDER BY COUNT(*) DESC
           ) AS genre_rank
    FROM "user_tracks"
    FULL OUTER JOIN "user"
        ON "user_tracks"."User_ID" = "user"."User_ID"
    FULL OUTER JOIN "tracks"
        ON "user_tracks"."Track_ID" = "tracks"."Track_ID"
    GROUP BY "user"."User_Region", "tracks"."Track_Genre"
)
SELECT
    "User_Region",
    "Track_Genre",
    play_count
FROM UserTrackGenre
WHERE genre_rank <= 3;
```

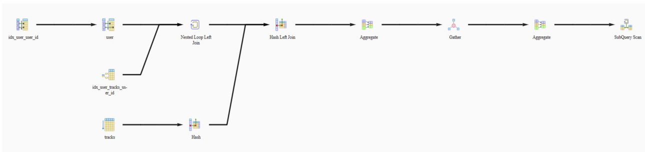
	User_Region text	Track_Genre text	play_count bigint
1	Afghanistan	Conjunto	935
2	Afghanistan	Swing	643
3	Afghanistan	Contemporary Gospel	621
4	Albania	Conjunto	1045
5	Albania	Contemporary Gospel	739
6	Albania	Swing	686
7	Algeria	Conjunto	1032
8	Algeria	Swing	714
9	Algeria	Boogie	669
10	American Samoa	Conjunto	1039
11	American Samoa	Swing	649
12	American Samoa	Boogie	639
13	Andorra	Conjunto	968
14	Andorra	Boogie	652
15	Andorra	Swing	633
16	Angola	Conjunto	943
17	Angola	Swing	667
18	Angola	Contemporary Gospel	661
19	Anguilla	Conjunto	986
20	Anguilla	Contemporary Gospel	698
21	Anguilla	Boogie	644
22	Antarctica (the territory South of 60 deg S)	Conjunto	956
23	Antarctica (the territory South of 60 deg S)	Boogie	702
24	Antarctica (the territory South of 60 deg S)	Contemporary Gospel	638

Total rows: 729 of 729 Query complete 00:00:07.219

Fig. 1. Table:1

The SQL query have designed serves to identify the three most popular music genres in each geographic region where our users are located. This is how it works: This SQL query identifies the top music genres in each region by joining user, track, and user track play data. It uses a Common Table Expression (CTE) named UserTrackGenre to combine and count track plays per genre for each region, employing a FULL OUTER JOIN to merge relevant data from user/_tracks, user, and tracks tables. The query then ranks these genres within each region using the ROW_NUMBER() window function, ordered by their play count in descending order. Finally, the main SELECT statement is intended to retrieve the regions, genres, and play counts, with a WHERE clause that filters to show only the top three genres per region, addressing the business need to understand popular music trends across different regions.

After Optimization



Performance Bottlenecks: Large tables like ”user,” ”user_tracks,” and ”tracks” involve extensive data processing, leading to performance bottlenecks during query execution.

Sorting and Window Function Overhead: The use of window functions, specifically ROW_NUMBER(), along with sorting based on count, can impose computational overhead, especially with a growing dataset.

Joining Large Tables: Joining tables with a substantial number of rows, such as the ”user_tracks” table with 3,999,992 rows, can result in slower query performance. To address these challenges, indexing concepts were adopted:

Indexing for Join Optimization: Indexes were created on the join columns, such as ”User_ID” in the ”user” and “user_tracks” table, to expedite the joining process.

Window Function Optimization: Efforts were made to ensure that the index on ”user”(”User_ID”) is effectively utilized by the query planner to optimize the window function, reducing the computational load.

B. Album with most subscribed users

Before Optimisation

```
select "album"."Album_Title",
count(distinct "user"."User_Email") as count from album
join "album_artists" on
"album_artists"."Album_ID" = "album"."Album_ID"
join "user_artists" on
"user_artists"."Artist_ID" = "album_artists"."Artist_ID"
join "user" on "user"."User_ID" = "user_artists"."User_ID"
where "user"."Subscription_ID" IS NOT NULL
group by "album"."Album_Title"
order by count
```

The SQL query provided in Fig 3 aims to identify albums

	Album_Title text	count bigint
1	approach born	11891
2	face power	11891
3	last	11895
4	happen sign	11895
5	with	11918
6	design	11918
7	focus	11946
8	drive paper	11946
9	plant now	11946
10	institution director	11946
11	so rise	11946
12	may	11947
13	prevent	11947
14	allow	11952
15	teach	11952
16	reflect statement	11954
17	coach space	11954
18	speak	11954
19	nor other	11958
20	fall today	11959
21	plan	11959
22	return camera	11962
23	center	11962
24	reflect	11968

Total rows: 475 of 475 Query complete 00:01:07.572

Fig. 3. Table:2

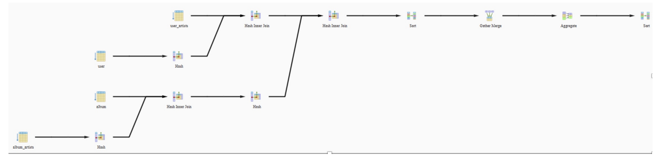


Fig. 4. Execution plan before optimization

with the highest number of subscribed users, which is a key indicator of album-wise revenue potential. This is how it works: The SQL query calculates the number of unique subscribers per album to gauge album-wise revenue potential. It joins the album, album_artists, user_artists, and user tables to establish a relationship between albums and their listeners who have active subscriptions (ensured by the Subscription_ID IS NOT NULL condition). By counting distinct User_Email entries grouped by Album_Title, the query ranks albums based on the number of subscribed users associated with them, which serves as a proxy for their revenue-generating popularity. The order by count statement then arranges the albums in descending order of their subscribed user count, putting the most popular albums at the top of the list.

After Optimization

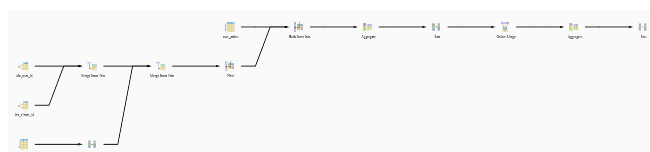


Fig. 5. Execution plan after optimization

```
SELECT
    "album"."Album_Title",
    SUM(CASE WHEN "user"."Subscription_ID"
    IS NOT NULL THEN 1 ELSE 0 END) AS count
FROM    "user"
JOIN
    "user_artists" ON "user_artists"."Artist_ID" =
    "user"."User_ID"
JOIN
    "album_artists" ON "album_artists"."Album_ID" =
    "user_artists"."Artist_ID"
JOIN
    "album" ON "album"."Album_ID" =
    "album_artists"."Album_ID"
GROUP BY
    "album"."Album_Title"
ORDER BY count DESC;
```

Initially, we optimized the database performance by creating an index on the User ID column, named ‘idx_user_id‘, given that the User table contained a substantial number of records, totaling 500,000 entries. Subsequently, we addressed the performance bottleneck caused by first joining all tables and then filtering the data. To improve efficiency, we restructured the code to implement filtering at the initial stage. Specifically, we fetched only the subscribed users from the User table and used this subset for subsequent joins with the user_artists table, album_artists table, and album table to retrieve the album titles. The result is a streamlined process that calculates the count of subscribed users (indicative of revenue contributors) associated with each album. This approach optimizes the computation of revenue contributors for each album in a more efficient manner.

C. Each artist's follower count is determined by the number of unique subscribed users associated with them

Before Optimization

```
SELECT
    "artist"."Artist_Name",
    (SELECT COUNT(DISTINCT "user"."Subscription_ID")
    --changed
     FROM "user_artists"
     JOIN "user" ON "user"."User_ID" =
     "user_artists"."User_ID"
     WHERE "user_artists"."Artist_ID" = "artist"."Artist_ID")
     AS user_count
FROM
    artist
order by "artist"."Artist_Name"
```

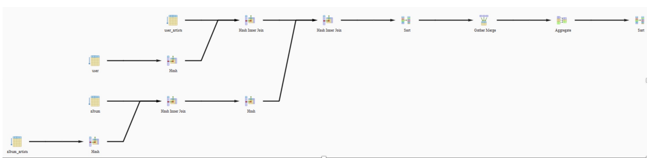


Fig. 6. Execution plan before optimization

	Artist_Name text	user_count bigint
1	Aaron Craig	12081
2	Aaron Marquez	12219
3	Adam Serrano	12071
4	Alexis Murphy	11993
5	Alicia Hood	12270
6	Allison Stephens	12229
7	Alyssa Cunningham	12179
8	Alyssa Lynch	12219
9	Amanda Dalton DDS	12250
10	Amanda Martin	12321
11	Amanda Smith	12174
12	Amber Cunningham	12239
13	Amber Davis	12292
14	Amy Burns	12083
15	Andrea Henderson	12050
16	Andrew Burton	12250
17	Andrew Sanchez	12244
18	Andrew Smith	12202
19	Angel Gray	12325
20	Anthony Olson	12101
21	Anthony Scott	12118
22	Anthony Townsend	12167
23	Ashley Roberson	12075
24	Ashley Rowe	12177

Total rows: 200 of 200   Query complete 00:05:01.055

Fig. 7. Table:3

The SQL queries provided are designed to measure the number of subscribers each artist has, which can indicate the potential revenue that each artist could generate. This is how it works: The SQL performs an inner join between the artist and user_artists tables on the Artist_ID, and another inner join with the user table on the User_ID. This setup links artists to users who are subscribers (ensured by the Subscription_ID IS NOT NULL condition). The query then counts the distinct subscription IDs for each artist and groups the results by Artist_Name, giving us a list of artists ordered by the number of unique subscribers they have. This count serves as a proxy for the artist's popularity and potential revenue impact since more subscribers could translate to more streams and revenue.

After Optimization

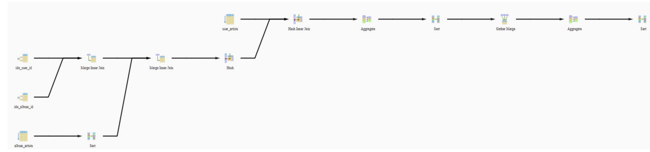


Fig. 8. Execution plan after optimization

```
select "artist"."Artist_Name", count("user"."Subscription_ID")
from "user"
inner join "user_artists" on "user"."User_ID" =
"user_artists"."User_ID"
inner join "artist" on "artist"."Artist_ID" =
"user_artists"."Artist_ID"
where "user"."Subscription_ID" IS NOT NULL
group by "artist"."Artist_Name"
```

The unoptimized query initially involves a subquery within the SELECT clause, which dynamically calculates the count of distinct "Subscription_ID" values for each artist. However, this subquery performs a join operation between the "user_artists" and "user" tables, applies a WHERE condition based on the current artist, and then sorts the result set by the artist name. The outer query, which retrieves artist names and the corresponding user counts, additionally orders the final result set by artist name. In contrast, the optimized query eliminates the need for a subquery and employs straightforward JOIN operations between the "user," "user_artists," and "artist" tables. The WHERE clause filters out rows where "Subscription_ID" is not null, focusing only on subscribed users. The GROUP BY clause efficiently groups the results by artist name. By consolidating these improvements, the optimized query avoids unnecessary subqueries, reduces redundant sorting operations, and enhances overall performance, resulting in a more streamlined and readable SQL statement.

	Artist_Name text	count bigint
1	Aaron Craig	12081
2	Aaron Marquez	12219
3	Adam Serrano	12071
4	Alexis Murphy	11993
5	Alicia Hood	12270
6	Allison Stephens	12229
7	Alyssa Cunningham	12179
8	Alyssa Lynch	12219
9	Amanda Dalton DDS	12250
10	Amanda Martin	12321
11	Amanda Smith	12174
12	Amber Cunningham	12239
13	Amber Davis	12292
14	Amy Burns	12083
15	Andrea Henderson	12050
16	Andrew Burton	12250
17	Andrew Sanchez	12244
18	Andrew Smith	12202
19	Angel Gray	12325
20	Anthony Olson	12101
21	Anthony Scott	12118
22	Anthony Townsend	12167
23	Ashley Roberson	12075
24	Ashley Rowe	12177

Total rows: 200 of 200   Query complete 00:00:14.884

D. Display podcast by region

```
select "user"."User_Region", "podcasts"."Podcast_Title"
from "playlist_and_podcast"
join "podcasts" on "podcasts"."Podcast_ID" =
"playlist_and_podcast"."Podcast_ID"
join "user_playlist" on
"user_playlist"."User_Playlist_ID" =
"playlist_and_podcast"."User_Playlist_ID"
join "user_and_playlist" on
"user_and_playlist"."User_Playlist_ID" =
"user_playlist"."User_Playlist_ID"
join "user" on
"user"."User_ID" = "user_and_playlist"."User_ID"
order by "user"."User_Region"
```

The SQL query provided retrieves a list of podcast titles

	User_Region text	Podcast_Title text
1	Afghanistan	whose least foot
2	Afghanistan	a fire commercial
3	Afghanistan	himself effect skin
4	Afghanistan	various up letter
5	Afghanistan	low analysis exist
6	Afghanistan	finish prevent truth
7	Afghanistan	American car rate
8	Afghanistan	class any week
9	Afghanistan	under affect fall
10	Afghanistan	another book pretty
11	Afghanistan	cell try surface
12	Afghanistan	as rise continue
13	Afghanistan	everything agree letter
14	Afghanistan	executive reflect process
15	Afghanistan	rich blue tend
16	Afghanistan	employee stage need
17	Afghanistan	pass recent long
18	Afghanistan	memory middle light
19	Afghanistan	accept floor phone
20	Afghanistan	seek same doctor
21	Afghanistan	possible to away
22	Afghanistan	example tax lose
23	Afghanistan	music skin them
24	Afghanistan	personal record sometimes

Total rows: 1000 of 248550 Query complete 00:00:01.640

Fig. 10. Table:4

along with their corresponding user regions by joining several tables: podcasts, user_playlist, playlist_and_podcast, and user. This allows the query to establish a relationship between the podcasts and the users' geographic locations. The query's result is ordered by the User_Region, which could help in understanding regional preferences for podcasts. For businesses, this information can be vital for targeted marketing and content localization strategies, as it reveals the popularity of specific podcasts in different regions. Such data can inform decisions on which podcasts to promote more heavily in each region to cater to local tastes and increase listener engagement.

E. Display number of podcasts by region

```
select "user"."User_Region",
count( distinct "podcasts"."Podcast_Title")
as CNT from "playlist_and_podcast" --changed
join "podcasts" on
"podcasts"."Podcast_ID" =
"playlist_and_podcast"."Podcast_ID"
join "user_playlist" on
"user_playlist"."User_Playlist_ID" =
"playlist_and_podcast"."User_Playlist_ID"
join "user_and_playlist" on
"user_and_playlist"."User_Playlist_ID" =
"user_playlist"."User_Playlist_ID"
join "user" on
"user"."User_ID" = "user_and_playlist"."User_ID"
group by "user"."User_Region"
order by CNT desc
```

The SQL query solves the business problem of displaying the number of unique podcasts available by region, which can inform regional market analysis and content distribution strategies. It does so by joining the user, user_playlist, playlist_and_podcast, and podcasts tables to map user regions to podcasts. The query then counts the distinct podcast titles for each region, groups the results by User_Region, and orders them in descending order of podcast count. This allows businesses to identify which regions have a higher variety of podcasts, potentially indicating a greater listener engagement or a more diverse listener base, thus aiding in targeted marketing and resource allocation for content creation and promotion.

	User_Region text	cnt bigint
1	Congo	2068
2	Korea	2043
3	Fiji	1120
4	Trinidad and Tobago	1096
5	Denmark	1087
6	Kyrgyz Republic	1080
7	Cape Verde	1078
8	Nicaragua	1075
9	Singapore	1072
10	Albania	1070
11	Bangladesh	1070
12	Estonia	1068
13	Bahrain	1068
14	Dominican Republic	1067
15	Palau	1065
16	Mayotte	1065
17	Gambia	1065
18	Chile	1065
19	Bouvet Island (Bouvetoya)	1064
20	Greece	1062
21	Grenada	1062
22	Monaco	1061
23	Morocco	1061
24	Ireland	1060

Total rows: 243 of 243 Query complete 00:00:00.956

Fig. 11. Table:5

F. Display most played host of podcasts by region (Ref: Fig 13, Fig 14)

```
WITH RankedPodcastHosts AS (
  SELECT
    "user"."User_Region",
    "podcasts"."Host_Name",
    ROW_NUMBER() OVER
    (PARTITION BY "user"."User_Region"
    ORDER BY COUNT(*) DESC) AS rank
  FROM
    "playlist_and_podcast"
    JOIN "podcasts" ON "podcasts"."Podcast_ID" =
    "playlist_and_podcast"."Podcast_ID"
    JOIN "user_playlist"
    ON "user_playlist"."User_Playlist_ID" =
    "playlist_and_podcast"."User_Playlist_ID"
    JOIN "user_and_playlist" ON
    "user_and_playlist"."User_Playlist_ID" =
    "user_playlist"."User_Playlist_ID"
    JOIN "user" ON "user"."User_ID" =
    "user_and_playlist"."User_ID"
  GROUP BY
    "user"."User_Region", "podcasts"."Host_Name"
)
SELECT
  "User_Region",
  "Host_Name"
FROM
  RankedPodcastHosts
WHERE
  rank = 1;
```

The SQL query provided addresses the business problem of identifying the most played podcast host in each region. It does this through a Common Table Expression (CTE) that joins relevant tables to correlate user regions with podcast hosts, counting the occurrences to determine popularity. The ROW_NUMBER() function is then used within each region to rank hosts based on their play count, ordered in descending order. By filtering where rank = 1, the query selects the top-ranked podcast host per region, effectively displaying the most played host across different geographical areas. This data can guide targeted marketing campaigns and content creator partnerships by highlighting which hosts resonate most with listeners in each region.

	User_Region text	Host_Name text
1	Afghanistan	Hunter Brown
2	Albania	Candace Sharp
3	Algeria	Amanda Mccullough
4	American Samoa	Jacob Juarez
5	Andorra	Gregory Rhodes
6	Angola	Anthony Brown
7	Anguilla	Travis Gordon
8	Antarctica (the territory South of 60 deg S)	Nichole Nguyen
9	Antigua and Barbuda	Jennifer Walsh MD
10	Argentina	Laura Hansen
11	Armenia	Chad Graham
12	Aruba	William Cook
13	Australia	Cynthia Martinez
14	Austria	Laura Harper
15	Azerbaijan	Christopher Johnson
16	Bahamas	Stephanie Williams
17	Bahrain	Rebecca Smith
18	Bangladesh	Marcus Ellis
19	Barbados	Susan Joseph
20	Belarus	Zachary Moore
21	Belgium	Sara Smith
22	Belize	Nicole Dominguez
23	Benin	Elizabeth Brown
24	Bermuda	Justin Nguyen

Total rows: 243 of 243 Query complete 00:00:03.637

Fig. 12. Table:6

G. Comparative Analysis of Payment Method Usage: Credit Card Versus Gift and Debit Card Transactions(Ref:Fig 16)

```
SELECT
    COUNT(CASE WHEN "Payment Method" =
    'Credit Card' THEN "Subscriber ID" END)
    AS CREDIT_CARD_COUNT,
    COUNT(CASE WHEN "Payment Method"
    IN ('Gift Card', 'Debit Card')
    THEN "Subscriber ID" END)
    AS GIFT_DEBIT_COUNT
FROM
    "transactions";
```

The SQL query addresses the business problem of comparing the frequency of usage between credit cards and gift/debit cards as payment methods. By using the COUNT function combined with a CASE statement, the query tallies the number of transactions made with credit cards and those made with either gift cards or debit cards. This differentiation helps a business understand customer payment preferences, which can influence payment processing decisions, fee structures, and promotional offers. The output showing separate counts for each payment method provides clear data to inform strategies for enhancing customer experience and optimizing transaction processing efficiency.

	credit_card_count bigint	gift_debit_count bigint
1	176124	184343

Total rows: 1 of 1 Query complete 00:00:00.377

Fig. 13. Table:7

H. Total revenue from all users on Spotify(Ref: Fig 17, Fig 18)

The SQL query provided counts the number of users who have an active subscription in the 'user' table, indicated by the Subscription_ID being not null. While this query effectively determines the number of subscribers, it does not directly calculate the total revenue. To estimate revenue from subscribers, you would need to multiply the count of subscribers by the subscription fee. If the subscription fee is consistent, the query's result can be used to calculate total revenue by multiplying the number of subscribers by that fee. If there are multiple subscription tiers or pricing plans, further details would be needed to accurately calculate total revenue.

```
select COUNT("User_ID") from "user"
where "user"."Subscription_ID"
IS NOT NULL
```

	count bigint
1	249750

Fig. 14. Table:8

I. Region wise revenue from users on Spotify

```
select COUNT("User_ID") as CNT, "user".
"User_Region" from "user"
where "user"."Subscription_ID" IS NOT NULL
group by "user"."User_Region"
order by CNT desc
```

The SQL query presented is a valuable tool for Spotify to gain insights into its revenue distribution among different regions. It accomplishes this by counting the number of users with active subscriptions in each region, identified by the 'Subscription_ID' not being null. The query then groups these counts by the 'User_Region,' creating a breakdown of the number of subscribers in each geographic area. By ordering the results in descending order based on the count ('CNT'), the query prioritizes regions with the highest number of subscribers. While this query directly counts subscribers and not revenue, it lays a crucial foundation for estimating revenue by multiplying the count of subscribers in each region by the respective subscription fee for that region. This information equips Spotify with region-specific revenue insights, enabling targeted marketing, content localization, and resource allocation to maximize revenue potential across different regions.

	cnt bigint	User_Region text
1	2112	Korea
2	2097	Congo
3	1099	Fiji
4	1098	Pitcairn Islands
5	1095	Switzerland
6	1093	Hong Kong
7	1085	Singapore
8	1080	Guyana
9	1077	Bangladesh
10	1076	Trinidad and Tobago
11	1074	Denmark
12	1070	Bahrain
13	1070	Tanzania
14	1069	Liberia
15	1068	Guatemala
16	1067	Argentina
17	1065	Oman
18	1063	Colombia
19	1063	Sao Tome and Principe
20	1063	Mayotte
21	1062	Kiribati
22	1061	Pakistan
23	1061	Ireland
24	1060	Puerto Rico

Total rows: 243 of 243 Query complete 00:00:00.426

Fig. 15. Table:9

J. Analysis of User Engagement: Number of Users with Descending Track Duration

Before Optimization

```
SELECT distinct(track_title),
"Total_Duration_in_seconds", count(users)
over (Partition by track_title,
"Total_Duration_in_seconds"
order by "Total_Duration_in_seconds")
from (
SELECT
  "tracks"."Track_Title" as track_title,
  "tracks"."Track_Duration_Minutes" *
   60 + "tracks"."Track_Duration_Seconds"
   AS "Total_Duration_in_seconds",
  "user"."User_ID" as users
FROM
  "tracks"
JOIN
  "user_tracks" ON "user_tracks"."Track_ID" =
   "tracks"."Track_ID"
JOIN
  "user" ON "user"."User_ID" =
   "user_tracks"."User_ID"
--  GROUP BY
--    "tracks"."Track_Title",
"Total_Duration_in_seconds"
-- ORDER BY
--   "Total_Duration_in_seconds" DESC;
) as new_one order by
 "Total_Duration_in_seconds"
```

	Track_Title text	Total_Duration_in_seconds bigint	Number_of_Users bigint
1	ball	359	8
2	determine economy	359	8
3	main	359	8
4	reveal	359	8
5	step	359	24
6	home	359	8
7	indicate	359	16
8	sing	359	16
9	soldier	359	16
10	fact message	359	8
11	including	359	8
12	play	359	16
13	month	359	8
14	among	359	24
15	once	359	8
16	last	359	8
17	building	359	24
18	discover	359	8
19	long	359	8
20	note such	359	8
21	pretty only	359	8
22	PM	359	8
23	deal	359	8
24	project	359	8

Total rows: 1000 of 256342   Query complete 00:00:11.173

Fig. 16. Table:10

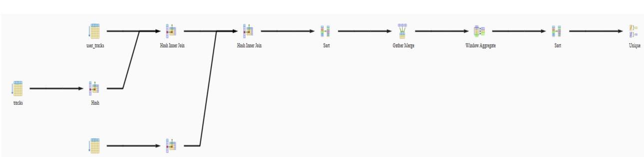


Fig. 17. Execution plan before optimization

The SQL query provided is a powerful tool for Spotify to analyze user engagement with tracks based on their duration. It achieves this by joining the 'tracks,' 'user_tracks,' and 'user' tables to establish a relationship between users and the tracks they've played. The query calculates the total duration of each track in seconds by converting the minutes and seconds into a unified format. It then counts the number of users who have played each track and groups the results by track title and total duration. By ordering the results in descending order of total duration, the query highlights tracks with the longest playtimes and the corresponding number of users who have engaged with them. This information can guide Spotify in identifying user preferences for longer tracks, aiding in content curation and personalized recommendations for users based on their track duration preferences.

After Optimization

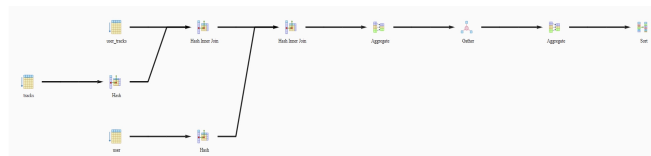


Fig. 18. Execution plan after optimization

```
SELECT
  "tracks"."Track_Title",
  "tracks"."Track_Duration_Minutes" * 60 +
  "tracks"."Track_Duration_Seconds"
  AS "Total_Duration_in_seconds",
  COUNT("user"."User_ID") AS "Number_of_Users"
FROM
  "tracks"
JOIN
  "user_tracks" ON "user_tracks"."Track_ID" =
  "tracks"."Track_ID"
JOIN
  "user" ON "user"."User_ID" =
  "user_tracks"."User_ID"
GROUP BY
  "tracks"."Track_Title",
  "Total_Duration_in_seconds"
ORDER BY
  "Total_Duration_in_seconds" DESC,
  "tracks"."Track_Title";
```

In the unoptimized query, a subquery is used to create a derived table (new_one) that includes the calculated total duration in seconds for each track and the associated user IDs. The outer query then applies the DISTINCT keyword to the track title, and a window function (COUNT over PARTITION) is used to count the number of users for each unique combination of track title and total duration.

The optimized query simplifies the approach by directly performing the necessary aggregations in the main query. It joins the "tracks," "user_tracks," and "user" tables and groups the results by track title and total duration. The COUNT function is used to calculate the number of users for each group.

K. Analyzing Churn Rate: Understanding Subscription Time and User Loss

```
WITH ChurnData AS (
  SELECT
    "Subscriber ID",
    EXTRACT(day FROM AGE("End Date", "Start Date"))
    AS No_of_days,
    EXTRACT(month FROM "End Date")
    AS Churn_Month,
    EXTRACT(year FROM "End Date")
    AS Churn_Year
  FROM
    "premium_subscription"
  WHERE
    "End Date" != '2099-01-01 00:00:00'
    -- Only consider completed subscriptions
)

SELECT
  Churn_Year,
  Churn_Month,
  COUNT(*) AS lost_customers,
  COUNT(*) * 100.0 / (SELECT COUNT(*)
  FROM "user") AS churn_rate
FROM
  ChurnData
GROUP BY
  Churn_Year, Churn_Month
ORDER BY
  Churn_Year DESC, Churn_Month DESC;
```

The SQL query presented offers valuable insights into user

	churn_year numeric	churn_month numeric	lost_customers bigint	churn_rate numeric
1	2024	1	5	0.001
2	2023	12	9	0.002
3	2023	11	9	0.002
4	2023	10	13	0.003
5	2023	9	11	0.002
6	2023	8	16	0.003
7	2023	7	22	0.004
8	2023	6	22	0.004
9	2023	5	16	0.003
10	2023	4	31	0.006
11	2023	3	25	0.005
12	2023	2	22	0.004
13	2023	1	38	0.008
14	2022	12	39	0.008
15	2022	11	36	0.007
16	2022	10	25	0.005
17	2022	9	36	0.007
18	2022	8	46	0.009
19	2022	7	37	0.007
20	2022	6	42	0.008
21	2022	5	40	0.008
22	2022	4	42	0.008
23	2022	3	43	0.009
24	2022	2	40	0.008

Total rows: 72 of 72 Query complete 00:00:00.347

Fig. 19. Table:11

churn rate for Spotify's premium subscription service. It begins by creating a temporary dataset ('ChurnData') that extracts essential information such as the subscriber's ID, the number of days they subscribed for, and the churn month and year. The query focuses on completed subscriptions ('End Date' not equal to '2099-01-01 00:00:00') to accurately calculate churn rates. It then proceeds to count the number of lost customers and calculates the churn rate as a percentage of the total user count. The results are grouped by churn month and year, providing a historical view of user churn. This data empowers Spotify to assess the effectiveness of retention strategies, understand when churn is highest, and make informed decisions to reduce churn rates and enhance customer loyalty.

L. Tracking Customer Dynamics: Monthly Analysis of Incoming, Outgoing, and Current Premium Subscribers

```
WITH MonthlyUserSubscriptions AS (
  SELECT
    DISTINCT
    EXTRACT(MONTH FROM "Start Date") AS month,
    EXTRACT(YEAR FROM "Start Date") AS year,
    "Subscriber ID",
    CASE WHEN "End Date" < '2099-01-01 00:00:00'
    THEN "Subscriber ID" END AS lost_customers
  FROM
    "premium_subscription"
  WHERE
    EXTRACT(YEAR FROM "Start Date") BETWEEN 2018 AND 2023
)

SELECT
  month,
  year,
  COUNT("Subscriber ID") AS subscription_count,
  COUNT(DISTINCT lost_customers) AS lost_customers_count,
  COUNT("Subscriber ID") - COUNT(DISTINCT lost_customers)
  AS difference
FROM
  MonthlyUserSubscriptions
GROUP BY
  month, year
ORDER BY
  year, month;
```

	month numeric	year numeric	subscription_count bigint	lost_customers_count bigint	difference bigint
1	1	2018	223	63	160
2	2	2018	234	58	176
3	3	2018	250	71	179
4	4	2018	239	57	182
5	5	2018	272	64	208
6	6	2018	235	61	174
7	7	2018	259	56	203
8	8	2018	260	65	195
9	9	2018	236	59	177
10	10	2018	283	68	215
11	11	2018	270	76	194
12	12	2018	259	67	192
13	1	2019	262	62	200
14	2	2019	263	72	191
15	3	2019	247	68	179
16	4	2019	221	65	156
17	5	2019	279	62	217
18	6	2019	240	69	171
19	7	2019	258	70	188
20	8	2019	254	55	199
21	9	2019	237	67	170
22	10	2019	261	60	201
23	11	2019	233	56	177
24	12	2019	238	73	165

Total rows: 40 of 40 Query complete 00:00:00.288

Fig. 20. Table:12

The SQL query provides a comprehensive view of Spotify's premium subscription customer dynamics on a month-by-month basis. It begins by creating a temporary dataset ('MonthlyUserSubscriptions') that extracts essential information such as the subscription start month, subscription end month for lost customers, and subscriber IDs. The query filters data from 2018 to 2023, focusing on this time frame. It then proceeds to count the number of new incoming customers, lost customers (those whose subscriptions ended), and current customers for each month and year. The results include the month, year, the total subscription count, the count of lost customers, and the net difference between incoming and outgoing customers. This data enables Spotify to monitor its customer retention and acquisition efforts, aiding in the evaluation of subscription strategies and identifying periods of significant customer activity.

M. Customer Analysis: Cumulative sum of users month-wise

```
SELECT month, year, users, sum(users)
over (order by year, month, users)
as cumulative_sum from
(SELECT *, lag(users,1)
 over(order by (year, month))
 as second_month from(
SELECT
EXTRACT(MONTH FROM created_at)
   AS month,
EXTRACT(YEAR FROM created_at)
   AS year,
COUNT(user_id) AS users

FROM
(
SELECT
"Created_At"
     AS created_at,
"User_ID"
     AS user_id
FROM
"user" AS users
) AS created_at_and_user
GROUP BY month, year
) as temp
) as new_output
```

	month numeric	year numeric	users bigint	cumulative_sum numeric
1	1	2018	7071	7071
2	2	2018	6438	13509
3	3	2018	7206	20715
4	4	2018	6960	27675
5	5	2018	7185	34860
6	6	2018	6892	41752
7	7	2018	7242	48994
8	8	2018	7083	56077
9	9	2018	6831	62908
10	10	2018	7204	70112
11	11	2018	6984	77096
12	12	2018	7265	84361
13	1	2019	7355	91716
14	2	2019	6660	98376
15	3	2019	7198	105574
16	4	2019	6929	112503
17	5	2019	7014	119517
18	6	2019	6897	126414
19	7	2019	6996	133410
20	8	2019	7072	140482
21	9	2019	6884	147366
22	10	2019	7069	154435
23	11	2019	6896	161331
24	12	2019	7158	168489

Total rows: 72 of 72 Query complete 00:00:00.971

Fig. 21. Table:5

The query calculates the monthly counts of new users, determining the number of users who joined the platform in each month. It extracts the month and year from the ”created_at” timestamp and counts the distinct user IDs. The query uses the window function LAG to compare the current month’s user count with the count from the previous month. The result, named ”second_month,” represents the user count in the month immediately preceding each month in the dataset. For each month, the query calculates the cumulative sum of users up to that point. The window function SUM is applied over the ordered sequence of months, providing a running total of users as the months progress. By examining the monthly user counts and their cumulative sums, the query provides insights into the growth patterns of user acquisition over time. It enables stakeholders to identify trends, seasonality, or periods of significant growth in user registrations.

IV. Queries with INSERT, UPDATE, DELETE

Query 1 - Insert into ”user” table:

```
insert into "user" ("User_ID", "User_Email",
"User_Encrypted_Pass", "User_Region", "Created_At",
"Is_Subscribed", "Subscription_ID")
Values
(500001,'dmql@buffalo.edu', 'd1s65a1as5v2dfv52dsvdsfv',
'United States', CURRENT_TIMESTAMP,'no', NULL)
```

	User_ID bigint	User_Email text	User_Encrypted_Pass text	User_Region text	Created_At timestamp without time zone	Is_Subscribed text	Subscription_ID double precision
1	500001	dmql@buffalo.edu	d1s65a1as5v2dfv52dsvdsfv	United States	2023-12-10 13:58:34.162358	no	[null]
2	500000	sierra16@yahoo.com	bcf1cbb6b8d5cb2b8c8c62b49c1bb43a	Egypt	2023-12-10 03:27:32	yes	249750
3	499999	kkelly@hotmail.com	ed45007a03bf0248da03ed00554ba39d	Guinea	2023-12-10 02:52:33	yes	249749

Fig. 22. Insert into ”user” table

-Purpose: Describe the purpose of this query. In this case, it’s adding a new user with specific details such as email, region, and subscription status.

-Impact: Explain the impact on the database. Mention that it adds a new user with a unique ID, their registration timestamp, and subscription information.

Query 2 - Update ”user” table:

```
UPDATE "user"
SET "Is_Subscribed" = 'yes',
"Subscription_ID" = 10000
WHERE "User_ID" = 500001;
```

	User_ID bigint	User_Email text	User_Encrypted_Pass text	User_Region text	Created_At timestamp without time zone	Is_Subscribed text	Subscription_ID double precision
1	500001	dmql@buffalo.edu	d1s65a1as5v2dfv52dsvdsfv	United States	2023-12-10 13:58:34.162358	yes	10000
2	500000	sierra16@yahoo.com	bcf1cbb6b8d5cb2b8c8c62b49c1bb43a	Egypt	2023-12-10 03:27:32	yes	249750
3	499999	kkelly@hotmail.com	ed45007a03bf0248da03ed00554ba39d	Guinea	2023-12-10 02:52:33	yes	249749
4	499998	ghoward@ray-leonard.com	5c4c34957beb0bdde669facb2c501cb2	Nigeria	2023-12-10 02:47:35	yes	249748

Fig. 23. Update ”user” table

-Purpose: Describe the purpose, which is to change the subscription status of a specific user to ’yes’ and assign a subscription ID.

-Impact: Explain the impact on the database. Mention that it modifies an existing user’s subscription details.

Query 3 - Insert into ”premium$_s$ubscription”$table$:

```
Insert into "premium_subscription"
("Subscriber ID", "Start Date",
"End Date", "Canceled or Not")
values
(10001, current_timestamp,
TO_TIMESTAMP('2099-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS'),
'renew')
```

	Subscriber ID bigint	Start Date timestamp without time zone	End Date timestamp without time zone	Canceled or Not text
1	10001	2023-12-10 14:01:31.744195	2099-01-01 00:00:00	renew
2	10000	2020-10-13 00:00:00	2099-01-01 00:00:00	renew

-Purpose: Describe the purpose of this query, which is to record a new premium subscription with a start date, end date, and cancellation status.

-Impact: Explain the impact on the database. Mention that it adds a new premium subscription record.

Query 4 - Delete from ”user” table:

```
Delete from "user"
where "User_ID" = 500001
```

-Purpose: Describe the purpose, which is to remove a user with a specific ID from the database.

-Impact: Explain the impact on the database. Mention that it removes a user’s record from the ”user” table.

	User_ID bigint	User_Email text	User_Encrypted_Pass text	User_Region text	Created_At timestamp without time zone	Is_Subscribed text	Subscription_ID double precision
1	500000	sierra16@yahoo.com	bcf1cbb6b8d5cb2b8c8c62b49c1bb43a	Egypt	2023-12-10 03:27:32	yes	249750
2	499999	kkelly@hotmail.com	ed45007a03bf0248da03ed00554ba39d	Guinea	2023-12-10 02:52:33	yes	249749
3	499998	ghoward@ray-leonard.com	5c4c34957beb0bdde669facb2c501cb2	Nigeria	2023-12-10 02:47:35	yes	249748

Fig. 25. Delete from "user" table

Query 5 - Update "premium_subscription" table:

```
UPDATE "premium_subscription"
SET "Canceled or Not" = 'cancel',
"End Date" = CURRENT_DATE + INTERVAL '1 month'
WHERE "premium_subscription"."Subscriber ID" = 10001;
```

	Subscriber ID bigint	Start Date timestamp without time zone	End Date timestamp without time zone	Canceled or Not text
1	10001	2023-12-10 14:01:31.744195	2024-01-10 00:00:00	cancel
2	10000	2020-10-13 00:00:00	2099-01-01 00:00:00	renew

Fig. 26. Update "premium_subscription" table

-Purpose: Describe the purpose of this query, which is to mark a premium subscription as canceled and update the end date.

-Impact: Explain the impact on the database. Mention that it updates an existing premium subscription record to reflect the cancellation status.

V. Website Description: "Spotify Data Management Hub"

This website serves as a comprehensive platform designed to seamlessly manage and explore the vast Spotify dataset. With integrated PostgreSQL functionality, it empowers users with the ability to perform essential database operations effortlessly. Here's a glimpse into the key features and functionalities of this platform:

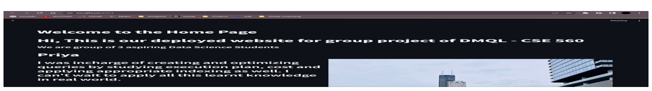


Fig. 27. Front Page

1. Data Integration:

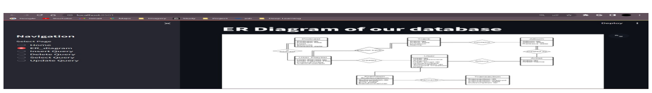


Fig. 28. ER

Our platform is built upon a robust PostgreSQL backend that integrates seamlessly with the Spotify dataset. This integration ensures that you have direct access to the rich and diverse data stored in the database, allowing for efficient data management.

2. CRUD Operations:

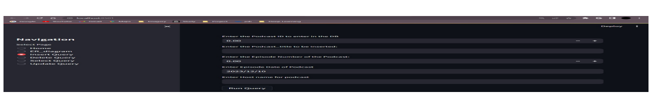


Fig. 29. Insert

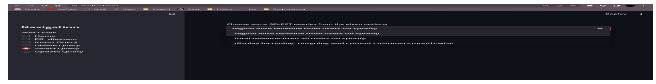


Fig. 30. Select

The heart of our platform lies in its support for CRUD (Create, Read, Update, Delete) operations. Users can effortlessly perform the following actions:

Insert: Add new data entries to the database, enabling the incorporation of fresh insights and trends into the Spotify dataset.

Update: Modify existing data records to reflect the latest changes or corrections, ensuring data accuracy and relevance.

Delete: Remove obsolete or erroneous data entries, maintaining the cleanliness and integrity of the dataset.

Select: Retrieve data from the database based on specific criteria, enabling in-depth analysis and exploration.

References

[1] https://medium.com/towards-data-engineering/design-the-database-for-a-system-like-spotify-95ffd1fb5927
[2] https://www.geeksforgeeks.org/python-pandas-working-with-dates-and-times/.
[3] https://www.geeksforgeeks.org/python-faker-library/
[4] https://www.geeksforgeeks.org/types-of-keys-in-relational-model-candidate-super-primary-alternate-and-foreign/
[5] https://www.holistics.io/blog/top-5-free-database-diagram-design-tools/
[6] https://app.diagrams.net/
[7] https://docs.streamlit.io/
[8] https://www.thatjeffsmith.com/archive/2023/01/learning-sql-with-your-own-spotify-streaming-history-data/