**SVKM’s NMIMS**

**Mukesh Patel School of Technology Management & Engineering (Mumbai Campus)**

**Computer Engineering Department (B Tech CSE/CSBS Sem IV/BTI Sem VIII/MBA.Tech-IV)**

**Database Management System**

**Project Report**

| Program | B Tech CSBS | |
| --- | --- | --- |
| Semester | Semester 4 | |
| Name of the Project: | Data analytics for artists using DBMS | |
|  | | |
| Details of Project Members |  |  |
| Batch | Roll No. | Name |
| B1 | E030 | Niket Korgaonkar |
| B1 | E035 | Manan Mehta |
| B2 | E071 | Aryan Vaish |
| Date of Submission:13/04/2025 | | |

**Contribution of each project Members:**

| Roll No. | Name: | Contribution |
| --- | --- | --- |
| E030 | Niket Korgaonkar | Worked on Storyline and ER diagram and Components |
| E035 | Manan Mehta | Worked on Components and MySQL and Relationship Model |
| E071 | Aryan Vaish | Worked on ER and Relationship Model and MySQL |

**Github link of your project:**

**Note:**

1. Create a readme file if you have multiple files
2. All files must be properly named (Example:R004\_DBMSProject)
3. Submit all relevant files of your work ( Report, all SQL files, Any other files)
4. **Plagiarism is highly discouraged (Your report will be checked for plagiarism)**

**Rubrics for the Project evaluation:**

| First phase of evaluation:  Innovative Ideas (5 Marks)  Design and Partial implementation (5 Marks) | 10 marks |
| --- | --- |
| Final phase of evaluation  Implementation, presentation and viva, Self-Learning and Learning Beyond classroom | 10 marks |

**Project Report**

**Selected Topic by:-Data analytics for artists using DBMS**

**Niket Korgaonkar, Roll number: E030**

**Manan Mehta, Roll number: E035**

**Aryan Vaish, Roll number: E071**

**Course: DBMS**

**AY: 2024-25**

**Table of Contents**

| **Sr no.** | **Topic** | **Page no.** |
| --- | --- | --- |
| **1** | Storyline | 5 |
| **2** | Components of Database Design | 9 |
| **3** | Entity Relationship Diagram | 10 |
| **4** | Relational Model | 11 |
| **5** | Normalization | 13 |
| **6** | SQL Queries | 20 |
| **7** | Project Demonstration | 46 |
| **9** | Self-learning beyond classroom | 48 |
| **10** | Learning from the project | 48 |
| **8** | Challenges faced | 50 |
| **9** | Conclusion | 50 |

**I. Storyline**

**Music Management, Made for Artists: Clarity, Control, and Creative Freedom**

Being an artist today means more than writing songs. You’re managing releases, checking streaming stats, tracking growth, and trying to figure out what’s working—all while staying inspired. With music scattered across platforms and data buried in dashboards, it’s easy to lose track of your own progress.

That's where intelligent music management software comes in. Built for artists, it enables you to keep track of everything you've put out, see how it's doing, and make informed, data-backed decisions about what to do next.

**The Problem: Creative Momentum, Disorganized Tracking**

Music adds up quickly—singles, features, demos, and albums. But there isn't usually a system for keeping track. Perhaps you're checking Spotify stats by hand. Perhaps you're making an educated guess about which songs are resonating. Or perhaps you've got dozens of tracks out there and no idea which ones are still picking up steam.

**Artists currently rely on:**

* Chaotic folders and stale spreadsheets
* Spread across multiple streaming platforms
* Intuition, not insight, to inform releases and promotions

Lacking a transparent view of their catalog and fan behavior, artists are missing out on important opportunities—and wasting time running down data rather than making music.

**The Solution: A Smarter Platform for Artists Who Want to Thrive**

This software functions as a command center for your music career. It consolidates all your releases and real-time information into one simple-to-use dashboard, so you can keep your catalog in check and learn from your fans with little effort.

**With this tool, you can:**

* Monitor every song you've released, on every platform
* Watch streams, saves, playlist adds, and monthly listeners in real time
* Segment your audience by geography, age, and gender
* View performance trends over time, from initial release to the present
* Organize your catalog by release date, genre, or project
* Receive alerts when older tracks begin to gain new traction

It's not merely about collecting data—it's about assisting you in making sense of it.

**Use Cases: Real Artists, Real Impact**

Here's how artists at various stages are leveraging the platform to remain focused and make better decisions:

**1. The Independent Artist Releasing Regularly**

Alex releases a new song every few months but never actually knows which ones are sticking. Now, they can see stream growth, top-performing tracks, and listener retention in real time—so they know precisely where to double down.

**2. The Touring Artist Planning Smarter**

Jordan is gearing up for a tour of the region. They use the location information to see precisely which cities are streaming their music the most—and leverage that to inform their setlists and ad campaigns.

**3. The Catalog-Heavy Artist Getting Their Act Together**

Taylor has over 50 songs released and no system. Now, their entire catalog is organized, searchable, and connected to live stats—making it simple to promote older songs that still have life in them.

**4. The Collaborator Managing Features**

Sam appears on a great many other artists' tracks. This platform allows them to monitor all their features, not only their own drops—so they understand which collabs are increasing their fanbase.

**5. The Data-Driven Artist Scaling Up**

Rae aims to create more intelligent campaigns and develop a long-term strategy. With growth charts, listener trends, and platform comparisons, they're making informed decisions based on what's actually working—not guesswork.

**The Big Picture**

This is not another dashboard. It's artists having agency over their careers, being connected with their fans, and translating streaming information into actual creative guidance. Whether you're new or growing quickly, this tool ensures that you never lose the connection to your music—or what's happening to you.

Because knowing your numbers isn't about chasing statistics—it's about creating great music, connecting with more listeners, and crafting a sustainable career.

**Functional Requirements for Musical Introduction Analysis Database System**

**Artist Table**  
 Stores information about musical artists.

* ArtistID (Primary Key) – Unique numeric identifier for each artist.
* Name – Full name of the artist or band.
* Country – The artist's country of origin.

**Genre Table**  
 Defines different music categories.

* GenreID (Primary Key) – Unique numeric identifier for each genre.
* Name – The name of the genre (e.g., Pop, Rock, Hip-Hop, Jazz).

**Album Table**  
 Contains details about music albums and links them to artists.

* AlbumID (Primary Key) – Unique numeric identifier for each album.
* Title – The name of the album.
* ReleaseDate – The date when the album was released.
* ArtistID (Foreign Key) – References the ArtistID to associate the album with an artist.

**Song Table**  
 Stores individual song details and connects them to albums and genres.

* SongID (Primary Key) – Unique numeric identifier for each song.
* Title – The name of the song.
* Duration – Length of the song in seconds.
* ReleaseDate – The date when the song was released.
* AlbumID (Foreign Key) – References the AlbumID to link the song to an album.
* GenreID (Foreign Key) – References the GenreID to classify the song by genre.

**Stream Table**  
 Tracks streaming statistics for songs.

* StreamID (Primary Key) – Unique numeric identifier for each streaming record.
* SongID (Foreign Key) – References the SongID to identify which song was streamed.
* StreamDate – The date when the streaming activity occurred.
* StreamCount – The number of times the song was streamed on that date.

**II. Components of Database Design**

**Artist**

* ArtistID (PK) – Unique identifier for each artist
* Name – Name of the artist
* Country – Country of origin of the artist

**Album**

* AlbumID (PK) – Unique identifier for each album
* Title – Title of the album
* ReleaseDate – Date the album was released
* ArtistID (FK) – Links to the artist who created the album

**Genre**

* GenreID (PK) – Unique identifier for each genre
* Name – Name of the genre

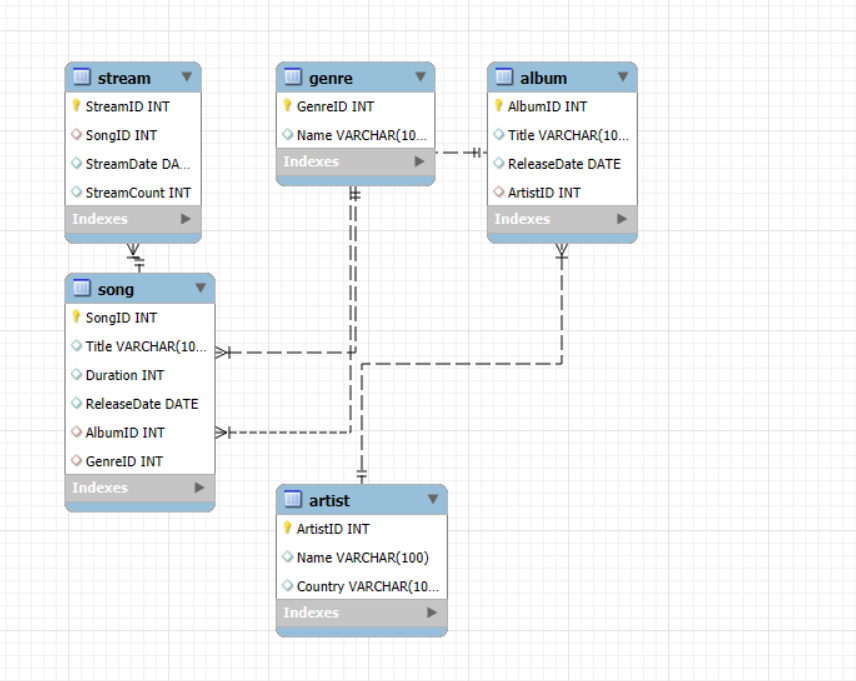
**Song**

* SongID (PK) – Unique identifier for each song
* Title – Title of the song
* Duration – Duration of the song in seconds
* ReleaseDate – Date the song was released
* AlbumID (FK) – Links to the album the song belongs to
* GenreID (FK) – Links to the genre of the song

**Stream**

* StreamID (PK) – Unique identifier for each stream record
* SongID (FK) – Links to the song that was streamed
* StreamDate – Date when the stream occurred
* StreamCount – Number of times the song was streamed on that date

**III. Entity Relationship Diagram**



**IV. Relational Model**

### 

### **Songs Table**

| **Attribute** | **Data Type** | **Description** |
| --- | --- | --- |
| song\_id | INT (PK, AUTO\_INCREMENT) | Unique identifier for the song |
| title | VARCHAR(100) NOT NULL | Title of the song |
| duration | INT | Duration of the song in seconds |
| release\_date | DATE | Date the song was released |
| album\_id | INT (FK) | Links to the album it belongs to |
| genre\_id | INT (FK) | Links to the genre of the song |

### **Albums Table**

| **Attribute** | **Data Type** | **Description** |
| --- | --- | --- |
| album\_id | INT (PK, AUTO\_INCREMENT) | Unique identifier for the album |
| title | VARCHAR(100) NOT NULL | Title of the album |
| release\_date | DATE | Release date of the album |
| artist\_id | INT (FK) | Links to the artist who created the album |

### **Genres Table**

| **Attribute** | **Data Type** | **Description** |
| --- | --- | --- |
| genre\_id | INT (PK, AUTO\_INCREMENT) | Unique identifier for the genre |
| name | VARCHAR(100) NOT NULL | Name of the genre |

### **Streams Table**

| **Attribute** | **Data Type** | **Description** |
| --- | --- | --- |
| stream\_id | INT (PK, AUTO\_INCREMENT) | Unique identifier for each stream record |
| song\_id | INT (FK) | Links to the song that was streamed |
| stream\_date | DATE | Date when the stream occurred |
| stream\_count | INT | Number of times the song was streamed |

### **Artists Table**

| **Attribute** | **Data Type** | **Description** |
| --- | --- | --- |
| artist\_id | INT (PK, AUTO\_INCREMENT) | Unique identifier for the artist |
| name | VARCHAR(100) NOT NULL | Name of the artist |
| country | VARCHAR(100) | Country of origin of the artist |

**V. Normalization**

**1. Unnormalized Form (UNF) – Initial Structure**

CREATE TABLE MusicData (

StreamID INT PRIMARY KEY,

SongID INT,

SongTitle VARCHAR(100),

Duration INT,

SongReleaseDate DATE,

AlbumID INT,

AlbumTitle VARCHAR(100),

AlbumReleaseDate DATE,

ArtistID INT,

ArtistName VARCHAR(100),

ArtistCountry VARCHAR(100),

GenreID INT,

GenreName VARCHAR(100),

StreamDate DATE,

StreamCount INT

);

**Key Problems in UNF:**

* All data stored in a single table with mixed entities
* Extreme duplication of artist/album/genre information
* Update anomalies: Changing artist details requires modifying multiple records
* Insert anomalies: Cannot add an artist without adding a song/stream
* Delete anomalies: Removing streams might accidentally delete artist data

**2. First Normal Form (1NF) – Basic Normalization**

CREATE TABLE Streams (

StreamID INT PRIMARY KEY,

SongID INT,

StreamDate DATE,

StreamCount INT

);

CREATE TABLE Songs (

SongID INT PRIMARY KEY,

Title VARCHAR(100),

Duration INT,

ReleaseDate DATE,

AlbumID INT,

GenreID INT

);

CREATE TABLE Albums (

AlbumID INT PRIMARY KEY,

Title VARCHAR(100),

ReleaseDate DATE,

ArtistID INT

);

CREATE TABLE Artists (

ArtistID INT PRIMARY KEY,

Name VARCHAR(100),

Country VARCHAR(100)

);

CREATE TABLE Genres (

GenreID INT PRIMARY KEY,

Name VARCHAR(100)

);

**1NF Improvements:**

* Separated data into distinct entity tables
* Eliminated repeating groups of information
* Ensured all fields contain single values only
* Established primary keys for each entity

**Remaining Issues After 1NF:**

* Missing relationships between tables
* Partial dependencies still exist
* Potential for update anomalies remains

**3. Second Normal Form (2NF) – Relationship Establishment**

sql

CREATE TABLE Streams (

StreamID INT PRIMARY KEY,

SongID INT,

StreamDate DATE,

StreamCount INT

);

CREATE TABLE Songs (

SongID INT PRIMARY KEY,

Title VARCHAR(100),

Duration INT,

ReleaseDate DATE,

AlbumID INT,

GenreID INT,

FOREIGN KEY (AlbumID) REFERENCES Albums(AlbumID)

);

CREATE TABLE Albums (

AlbumID INT PRIMARY KEY,

Title VARCHAR(100),

ReleaseDate DATE,

ArtistID INT,

FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID)

);

**2NF Improvements:**

* Added foreign key relationships between tables
* Eliminated partial dependencies
* Improved data integrity through references

**4. Third Normal Form (3NF) – Final Optimized Structure**

sql

CREATE TABLE Artists (

ArtistID INT PRIMARY KEY,

Name VARCHAR(100),

Country VARCHAR(100)

);

CREATE TABLE Genres (

GenreID INT PRIMARY KEY,

Name VARCHAR(100)

);

CREATE TABLE Albums (

AlbumID INT PRIMARY KEY,

Title VARCHAR(100),

ReleaseDate DATE,

ArtistID INT,

FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID)

);

CREATE TABLE Songs (

SongID INT PRIMARY KEY,

Title VARCHAR(100),

Duration INT,

ReleaseDate DATE,

AlbumID INT,

GenreID INT,

FOREIGN KEY (AlbumID) REFERENCES Albums(AlbumID),

FOREIGN KEY (GenreID) REFERENCES Genres(GenreID)

);

CREATE TABLE Streams (

StreamID INT PRIMARY KEY,

SongID INT,

StreamDate DATE,

StreamCount INT,

FOREIGN KEY (SongID) REFERENCES Songs(SongID)

);

**3NF Achievements:**

* Eliminated all transitive dependencies
* Established complete referential integrity
* Removed all data redundancy
* Prevented update/insert/delete anomalies
* Created optimal structure for querying

Normalization Progression Summary Table:

| Stage | Key Changes | Benefits |
| --- | --- | --- |
| UNF | Single table with all data | Simple but problematic structure |
| 1NF | Split into separate tables | Eliminated repeating groups |
| 2NF | Added foreign key relationships | Removed partial dependencies |
| 3NF | Final relationship optimization | Eliminated transitive dependencies |

**VI. SQL Queries**

**ARTIST TABLE**

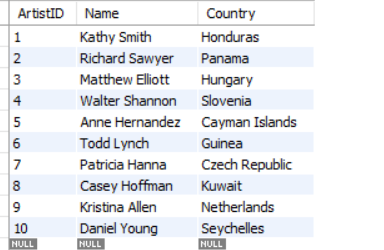
CREATE TABLE Artist (

ArtistID INT PRIMARY KEY,

Name VARCHAR(100),

Country VARCHAR(100)

);



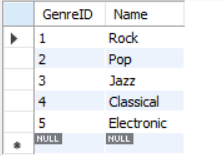
**GENRE TABLE**

CREATE TABLE Genre (

GenreID INT PRIMARY KEY,

Name VARCHAR(100)

);

****

**ALBUM TABLE**

CREATE TABLE Album (

AlbumID INT PRIMARY KEY,

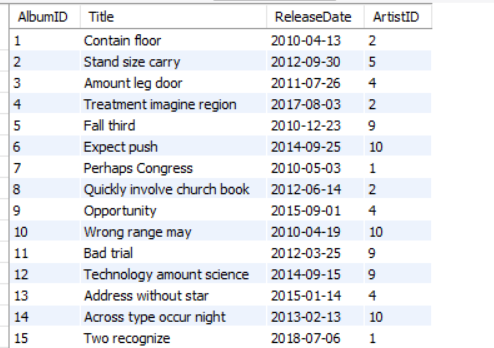
Title VARCHAR(100),

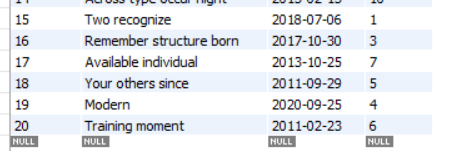
ReleaseDate DATE,

ArtistID INT,

FOREIGN KEY (ArtistID) REFERENCES Artist(ArtistID)

);





**SONG TABLE**

CREATE TABLE Song (

SongID INT PRIMARY KEY,

Title VARCHAR(100),

Duration INT,

ReleaseDate DATE,

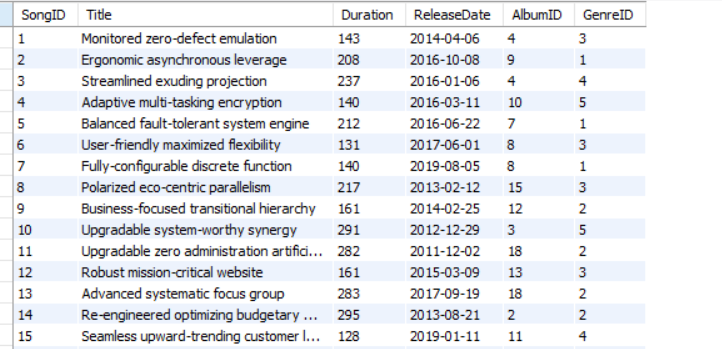
AlbumID INT,

GenreID INT,

FOREIGN KEY (AlbumID) REFERENCES Album(AlbumID),

FOREIGN KEY (GenreID) REFERENCES Genre(GenreID)

);



(There are 500 songs, this is just a snippet of the first 15)

**STREAMS TABLE**

CREATE TABLE Stream (

StreamID INT PRIMARY KEY,

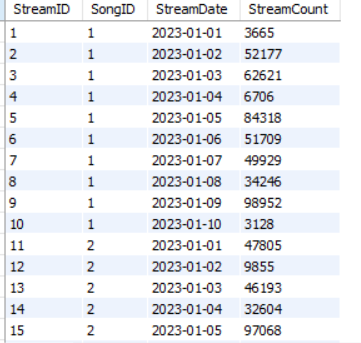
SongID INT,

StreamDate DATE,

StreamCount INT,

FOREIGN KEY (SongID) REFERENCES Song(SongID)

);



(There are 5000 stream, this is just a snippet of the first 15)

**1. Top 10 Most Streamed Songs**

SELECT s.SongID, so.Title, SUM(s.StreamCount) AS TotalStreams

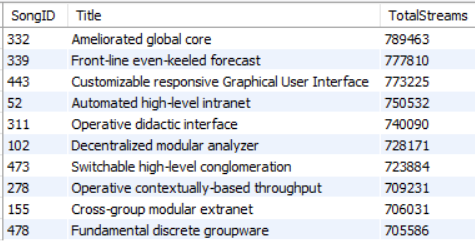
FROM Stream s

JOIN Song so ON s.SongID = so.SongID

GROUP BY s.SongID, so.Title

ORDER BY TotalStreams DESC

LIMIT 10;



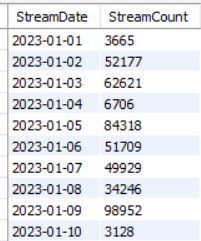
**2. Daily Stream Count for a Specific Song**

SELECT StreamDate, StreamCount

FROM Stream

WHERE SongID = 1

ORDER BY StreamDate;



**3. Artists with Most Albums**

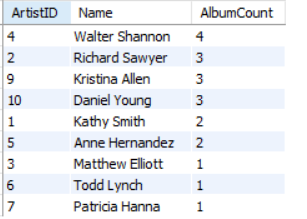
SELECT a.ArtistID, ar.Name, COUNT(a.AlbumID) AS AlbumCount

FROM Album a

JOIN Artist ar ON a.ArtistID = ar.ArtistID

GROUP BY a.ArtistID, ar.Name

ORDER BY AlbumCount DESC;



**4. Average Song Duration by Genre**

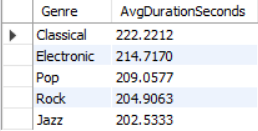
SELECT g.Name AS Genre, AVG(s.Duration) AS AvgDurationSeconds

FROM Song s

JOIN Genre g ON s.GenreID = g.GenreID

GROUP BY g.Name

ORDER BY AvgDurationSeconds DESC;



**5. Busiest Streaming Days**

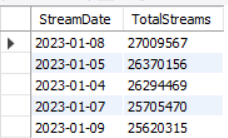
SELECT StreamDate, SUM(StreamCount) AS TotalStreams

FROM Stream

GROUP BY StreamDate

ORDER BY TotalStreams DESC

LIMIT 5;



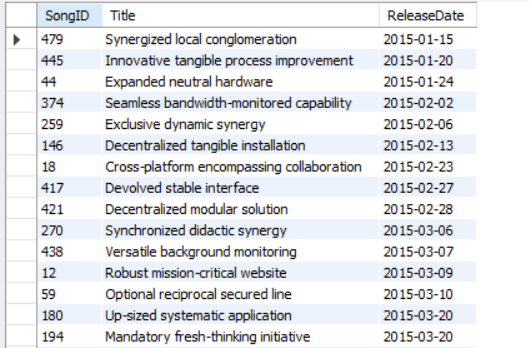
**6. Songs Released in 2015**

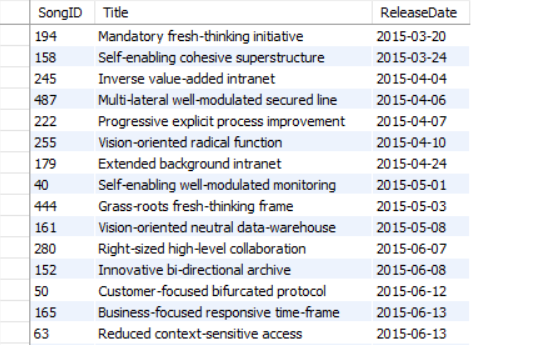
SELECT SongID, Title, ReleaseDate

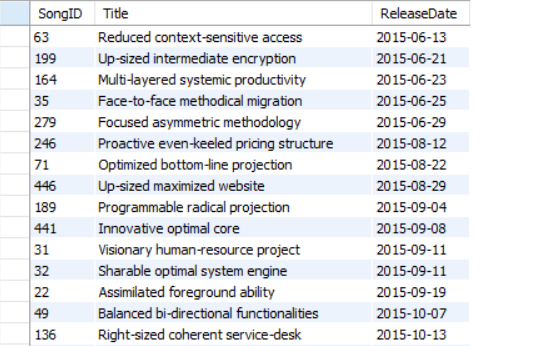
FROM Song

WHERE YEAR(ReleaseDate) = 2015

ORDER BY ReleaseDate;







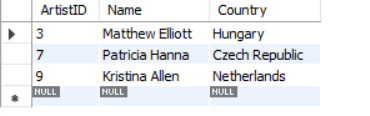


**7. Artists from Specific Countries**

SELECT ArtistID, Name, Country

FROM Artist

WHERE Country IN ('Netherlands', 'Hungary', 'Czech Republic');



**8. Albums with Multiple Genres**

SELECT a.AlbumID, a.Title, COUNT(DISTINCT s.GenreID) AS GenreCount

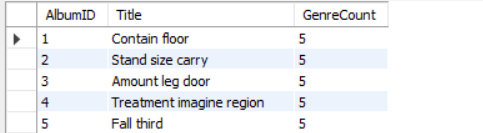
FROM Album a

JOIN Song s ON a.AlbumID = s.AlbumID

GROUP BY a.AlbumID, a.Title

HAVING COUNT(DISTINCT s.GenreID) > 1

ORDER BY GenreCount DESC;



**9. Stream Growth Rate for Top Song**

SELECT s1.StreamDate, s1.StreamCount,

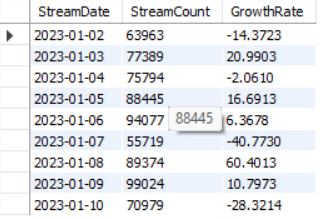
(s1.StreamCount - s2.StreamCount) / s2.StreamCount \* 100 AS GrowthRate

FROM Stream s1

JOIN Stream s2 ON s1.SongID = s2.SongID AND s1.StreamDate = s2.StreamDate + INTERVAL 1 DAY

WHERE s1.SongID = (SELECT SongID FROM Stream GROUP BY SongID ORDER BY SUM(StreamCount) DESC LIMIT 1)

ORDER BY s1.StreamDate;



**10. Artists Without Albums**

SELECT ar.ArtistID, ar.Name

FROM Artist ar

LEFT JOIN Album al ON ar.ArtistID = al.ArtistID

WHERE al.AlbumID IS NULL;



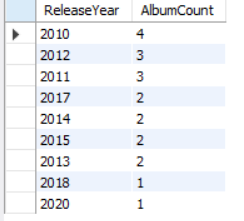
**11. Most Prolific Years for Album Releases**

SELECT YEAR(ReleaseDate) AS ReleaseYear, COUNT(\*) AS AlbumCount

FROM Album

GROUP BY YEAR(ReleaseDate)

ORDER BY AlbumCount DESC;



**12. Genre Popularity by Streams**

SELECT g.Name AS Genre, SUM(st.StreamCount) AS TotalStreams

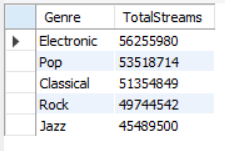
FROM Genre g

JOIN Song s ON g.GenreID = s.GenreID

JOIN Stream st ON s.SongID = st.SongID

GROUP BY g.Name

ORDER BY TotalStreams DESC;



**13. Longest Songs by Genre**

SELECT g.Name AS Genre, s.Title, s.Duration

FROM Song s

JOIN Genre g ON s.GenreID = g.GenreID

WHERE (s.GenreID, s.Duration) IN (

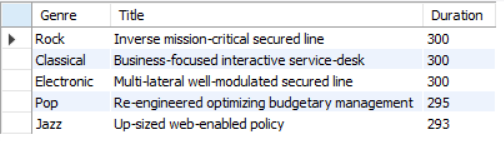
SELECT GenreID, MAX(Duration)

FROM Song

GROUP BY GenreID

)

ORDER BY s.Duration DESC;



**14. Artist Stream Performance**

SELECT ar.ArtistID, ar.Name, SUM(st.StreamCount) AS TotalStreams

FROM Artist ar

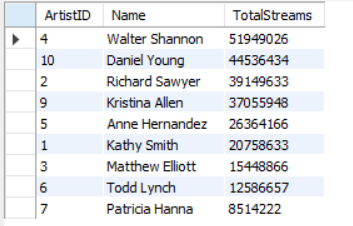
JOIN Album al ON ar.ArtistID = al.ArtistID

JOIN Song s ON al.AlbumID = s.AlbumID

JOIN Stream st ON s.SongID = st.SongID

GROUP BY ar.ArtistID, ar.Name

ORDER BY TotalStreams DESC;



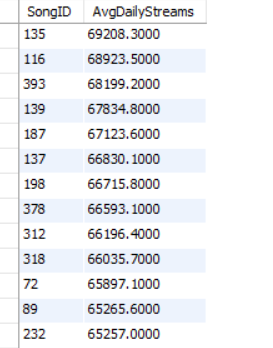
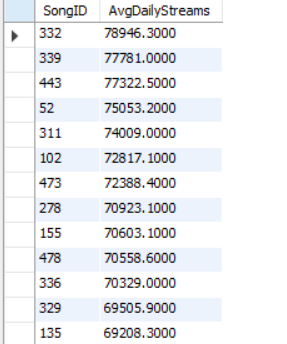
**15. Average Daily Streams per Song**

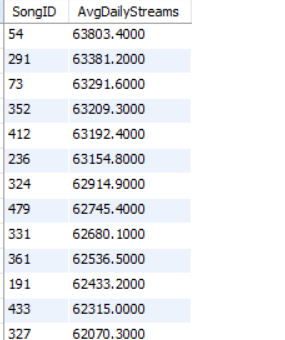
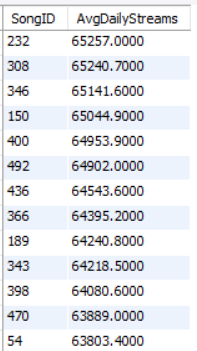
SELECT SongID, AVG(StreamCount) AS AvgDailyStreams

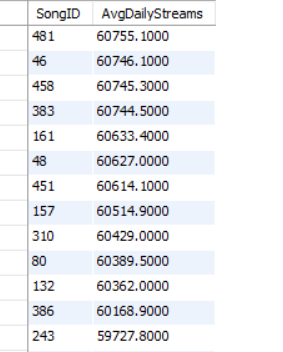
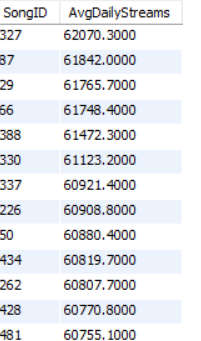
FROM Stream

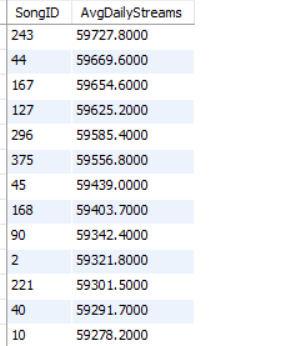
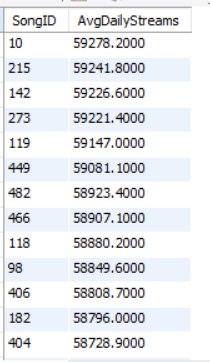
GROUP BY SongID

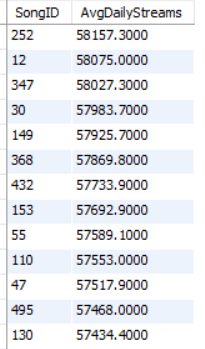
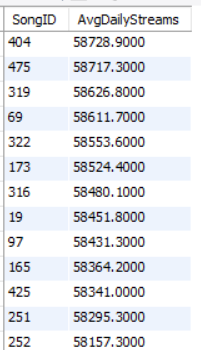
ORDER BY AvgDailyStreams DESC;

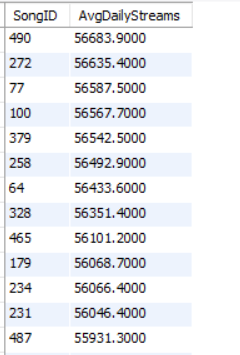
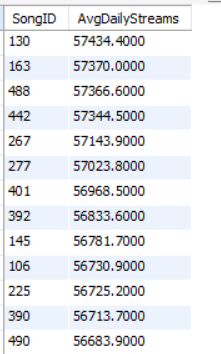


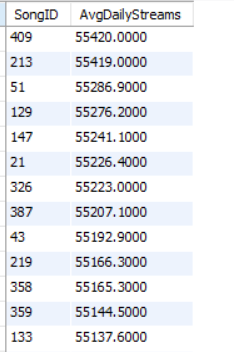
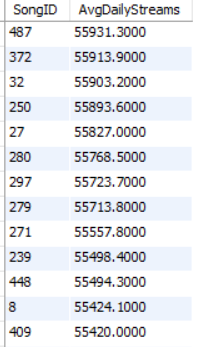


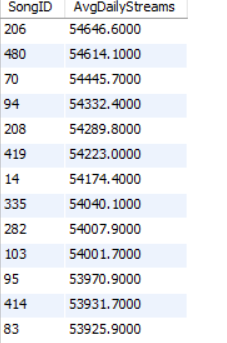
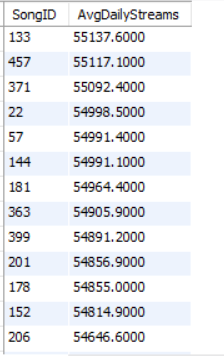


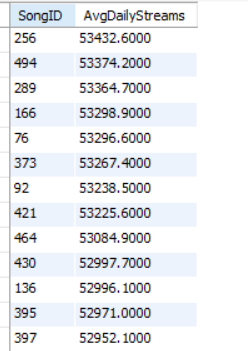
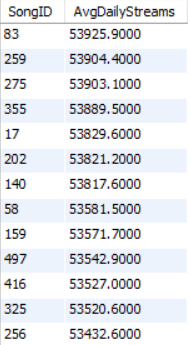


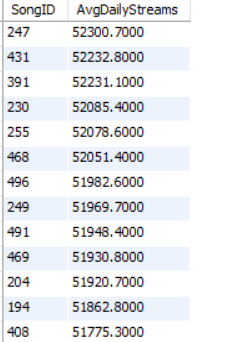
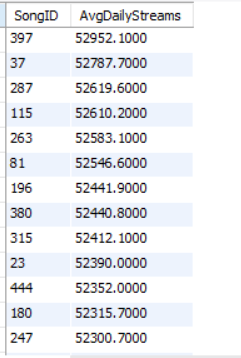












**16. Albums with Longest Average Song Duration**

SELECT a.AlbumID, a.Title, AVG(s.Duration) AS AvgSongDuration

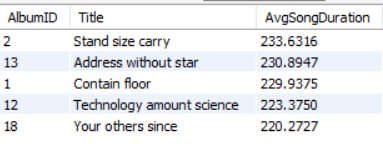
FROM Album a

JOIN Song s ON a.AlbumID = s.AlbumID

GROUP BY a.AlbumID, a.Title

ORDER BY AvgSongDuration DESC

LIMIT 5;



**17. Artists with Songs in Multiple Genres**

SELECT ar.ArtistID, ar.Name, COUNT(DISTINCT s.GenreID) AS GenreCount

FROM Artist ar

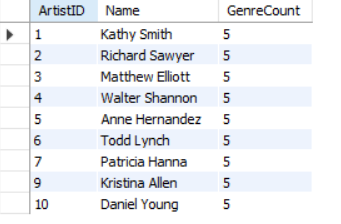
JOIN Album al ON ar.ArtistID = al.ArtistID

JOIN Song s ON al.AlbumID = s.AlbumID

GROUP BY ar.ArtistID, ar.Name

HAVING COUNT(DISTINCT s.GenreID) > 1

ORDER BY GenreCount DESC;



**18. Newest Songs with High Streams**

SELECT s.SongID, s.Title, s.ReleaseDate, SUM(st.StreamCount) AS TotalStreams

FROM Song s

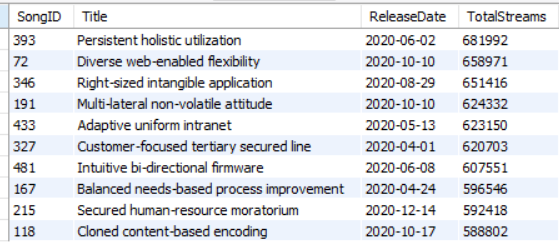
JOIN Stream st ON s.SongID = st.SongID

WHERE s.ReleaseDate >= '2020-01-01'

GROUP BY s.SongID, s.Title, s.ReleaseDate

ORDER BY TotalStreams DESC

LIMIT 10;



**19. Artist Consistency Analysis (Do artists stick to one genre?)**

SELECT

ar.Name AS Artist,

COUNT(DISTINCT s.GenreID) AS GenresUsed,

GROUP\_CONCAT(DISTINCT g.Name) AS GenreList

FROM Artist ar

JOIN Album al ON ar.ArtistID = al.ArtistID

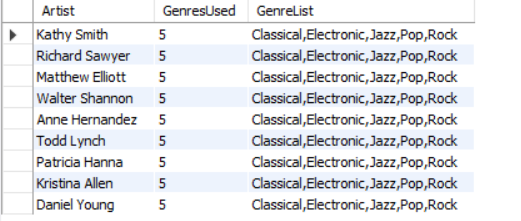
JOIN Song s ON al.AlbumID = s.AlbumID

JOIN Genre g ON s.GenreID = g.GenreID

GROUP BY ar.ArtistID

HAVING COUNT(DISTINCT s.GenreID) > 1

ORDER BY GenresUsed DESC;



**20. Album Release Strategy (Seasonal patterns**)

SELECT

CASE

WHEN MONTH(ReleaseDate) BETWEEN 3 AND 5 THEN 'Spring'

WHEN MONTH(ReleaseDate) BETWEEN 6 AND 8 THEN 'Summer'

WHEN MONTH(ReleaseDate) BETWEEN 9 AND 11 THEN 'Fall'

ELSE 'Winter'

END AS Season,

COUNT(\*) AS AlbumsReleased,

ROUND(COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM Album), 1) AS Percentage

FROM Album

GROUP BY Season

ORDER BY AlbumsReleased DESC;

**21. Song Longevity Analysis (Older vs newer song performance)**

SELECT

CASE

WHEN YEAR(s.ReleaseDate) < 2015 THEN 'Classic (Pre-2015)'

WHEN YEAR(s.ReleaseDate) BETWEEN 2015 AND 2018 THEN 'Mid-Period (2015-2018)'

ELSE 'Recent (2019+)'

END AS AgeCategory,

AVG(st.StreamCount) AS AvgDailyStreams,

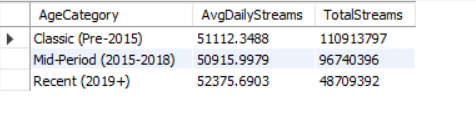
SUM(st.StreamCount) AS TotalStreams

FROM Song s

JOIN Stream st ON s.SongID = st.SongID

GROUP BY AgeCategory

ORDER BY TotalStreams DESC;



**22. Stream Decay Analysis (How streams drop over time)**

SELECT

s.Title AS Song,

DATEDIFF(MAX(st.StreamDate), MIN(st.StreamDate)) AS DaysTracked,

MIN(st.StreamCount) AS MinDailyStreams,

MAX(st.StreamCount) AS MaxDailyStreams,

ROUND(MIN(st.StreamCount) \* 100.0 / MAX(st.StreamCount), 1) AS PercentDrop

FROM Song s

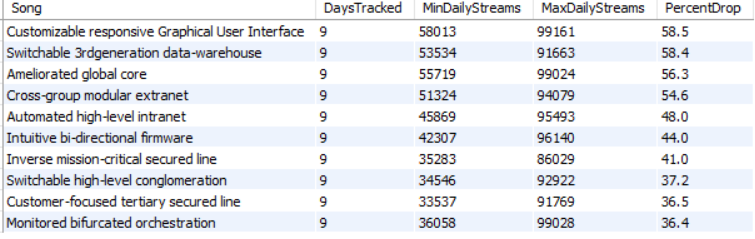
JOIN Stream st ON s.SongID = st.SongID

GROUP BY s.SongID

HAVING DaysTracked > 5

ORDER BY PercentDrop DESC

LIMIT 10;



**23. Album "Cohesion" Score (Do albums have consistent song lengths?)**

SELECT

a.Title AS Album,

ar.Name AS Artist,

COUNT(\*) AS Songs,

ROUND(STDDEV(s.Duration), 1) AS DurationStdDev,

AVG(s.Duration) AS AvgDuration,

ROUND(STDDEV(s.Duration) \* 100.0 / AVG(s.Duration), 1) AS CohesionScore

FROM Album a

JOIN Song s ON a.AlbumID = s.AlbumID

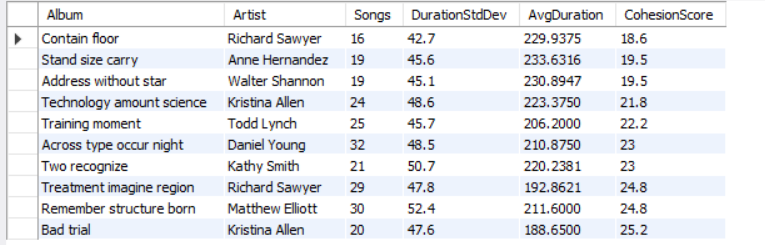
JOIN Artist ar ON a.ArtistID = ar.ArtistID

GROUP BY a.AlbumID

HAVING COUNT(\*) > 3

ORDER BY CohesionScore ASC

LIMIT 10;



**24. Release Day of Week Analysis**

SELECT

DAYNAME(s.ReleaseDate) AS ReleaseDay,

COUNT(\*) AS SongsReleased,

AVG(st.StreamCount) AS AvgStreams,

ROUND(COUNT(\*) \* 100.0 / (SELECT COUNT(\*) FROM Song), 1) AS Percentage

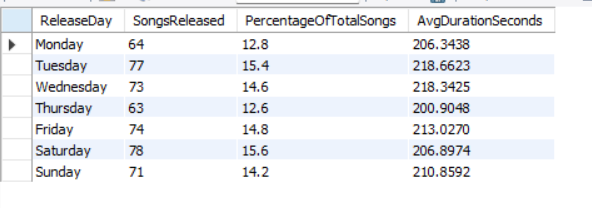
FROM Song s

JOIN Stream st ON s.SongID = st.SongID

GROUP BY ReleaseDay

ORDER BY

FIELD(ReleaseDay, 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday');



**25. Artist Productivity vs Popularity**

SELECT

ar.Name AS Artist,

COUNT(DISTINCT al.AlbumID) AS Albums,

COUNT(DISTINCT s.SongID) AS Songs,

SUM(st.StreamCount) AS TotalStreams,

ROUND(SUM(st.StreamCount) / COUNT(DISTINCT s.SongID), 0) AS StreamsPerSong,

ROUND(COUNT(DISTINCT s.SongID) / NULLIF(COUNT(DISTINCT al.AlbumID), 0), 1) AS SongsPerAlbum

FROM Artist ar

LEFT JOIN Album al ON ar.ArtistID = al.ArtistID

LEFT JOIN Song s ON al.AlbumID = s.AlbumID

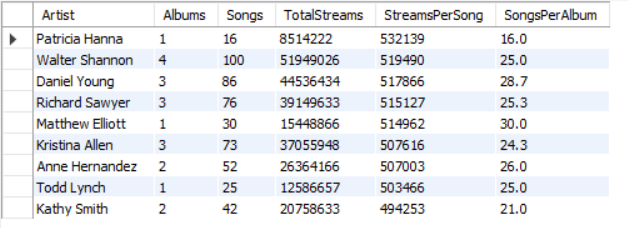
LEFT JOIN Stream st ON s.SongID = st.SongID

GROUP BY ar.ArtistID

HAVING COUNT(DISTINCT s.SongID) > 5

ORDER BY StreamsPerSong DESC

LIMIT 10;



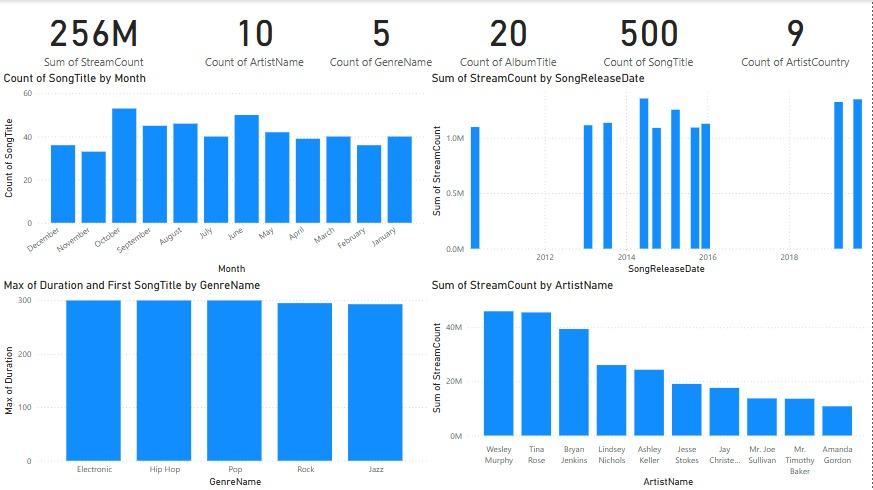
**VI. Project demonstration**

● Tools/software/ libraries used:

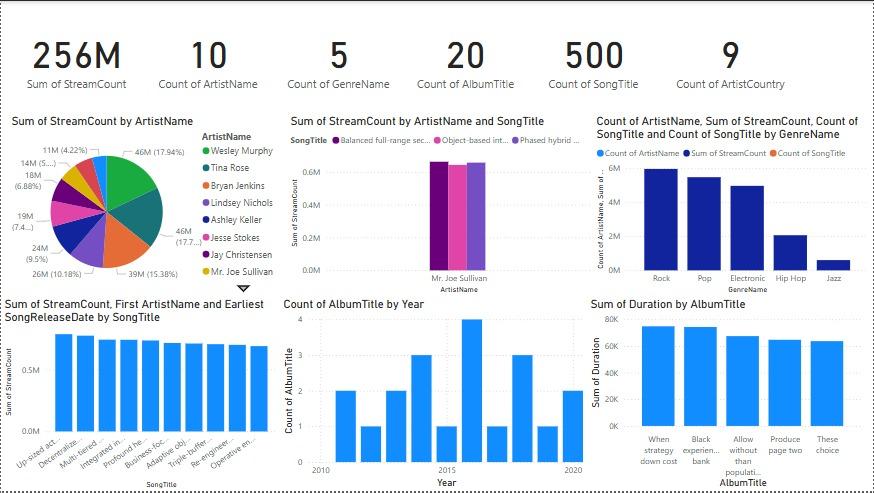
1. MySQL
2. POWERBI
3. Chatgpt(for generating data of false songs and study purposes only)

**POWER BI REPRESENTATION:**

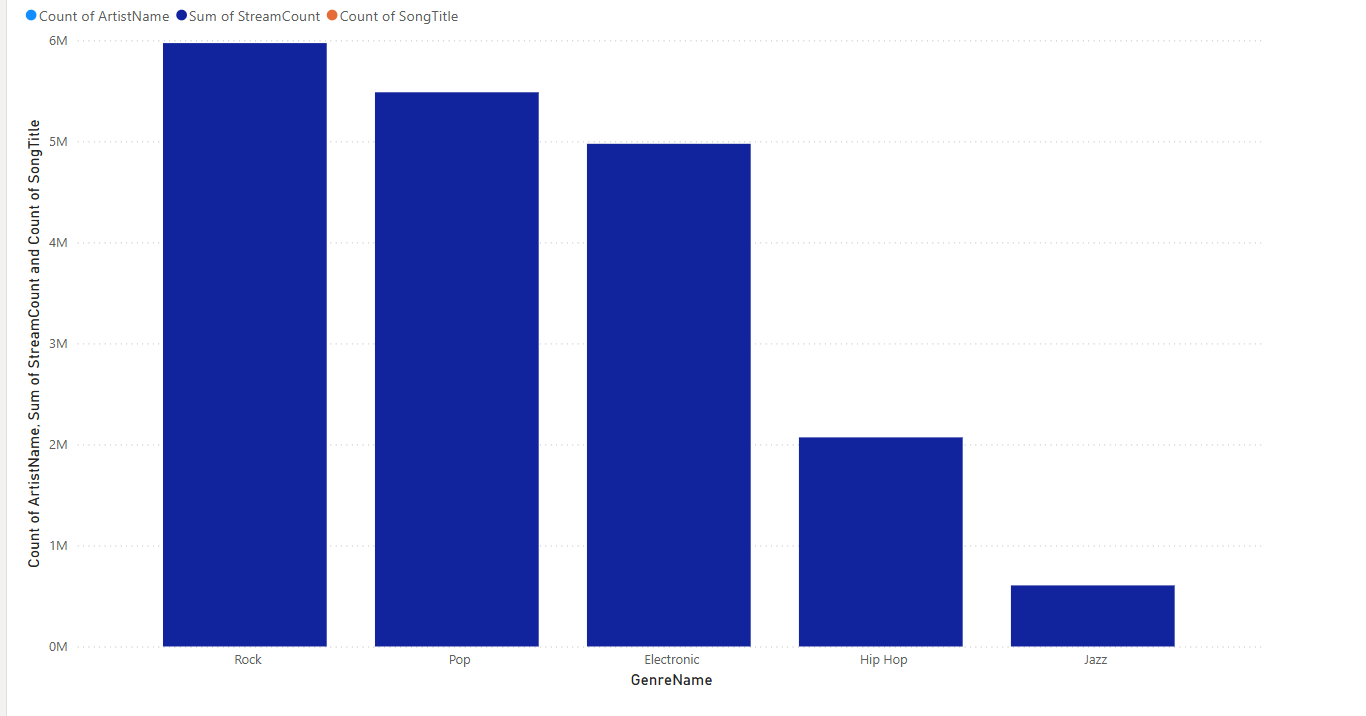
**DASHBOARD 1:**



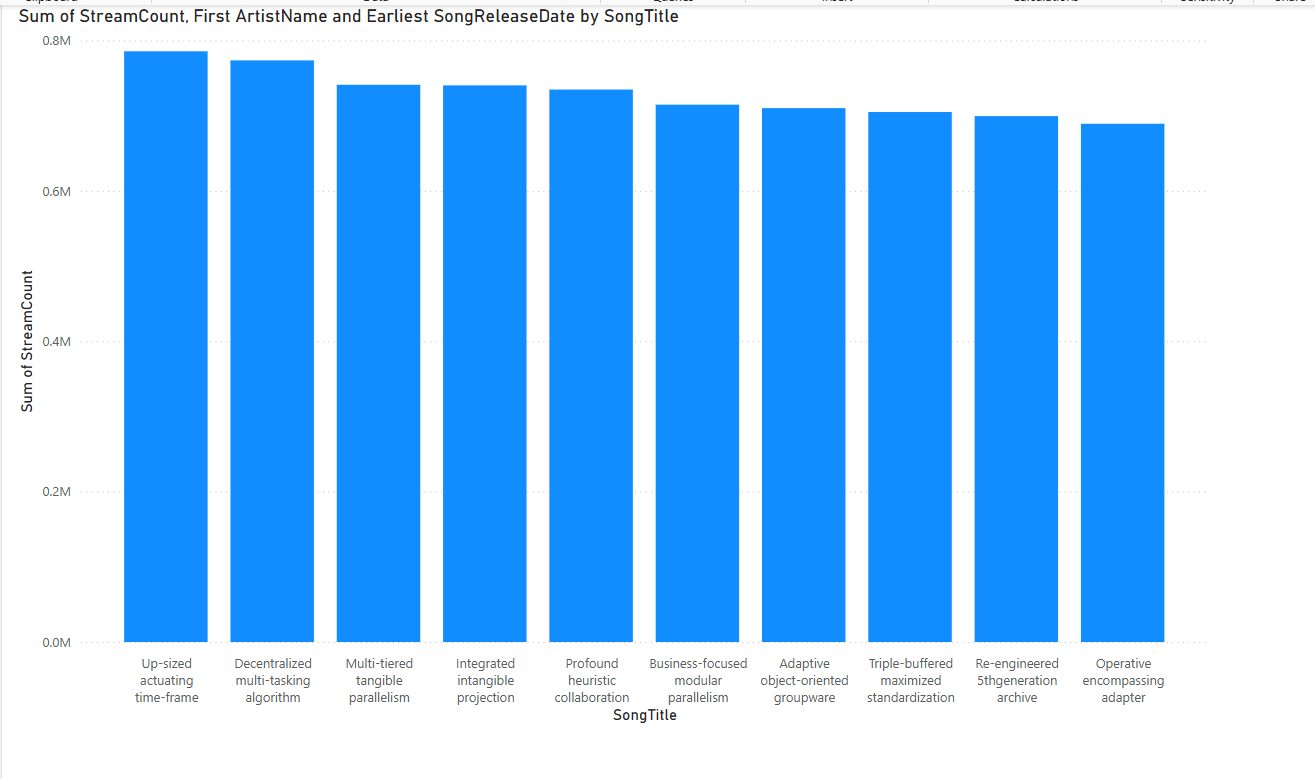
**DASHBOARD 2:**



**Top Genre streamed:**



**Top songs and their total streams:**



**VII. Self -Learning beyond classroom**

One of the most useful things we learned outside of class was how to use Power BI. We had never used it before, but we soon saw how useful it could be for graphing the data we were dealing with.

We learned to connect our MySQL database with Power BI, tidied and normalized our tables, and created dashboards that finally made our data worth looking at. We utilized charts to identify streaming trends, determine what genres were trending, and compare the performances of artists. We also got into using DAX formulas to calculate things like growth rates and averages—stuff that’s way easier to understand when it’s visual.

Learning Power BI on our own gave us a better feel for how real data is used and shared in the industry—and honestly, it was fun to see our project come to life that way.

**VIII. Learning from the Project**

This project gave us a much clearer understanding of how databases actually work beyond the textbook. Designing the schema from scratch—thinking through what tables we needed, how they should connect, and what data types made sense—helped us see how structure affects everything down the line.

Creating SQL queries wasn't so much a matter of seeing right syntax, but thinking straight and fixing genuine data issues. We improved how to utilize joins, groupings, and subqueries, and realized how tweaking the logic minutely can rearrange the outcome entirely.

Power BI work introduced another level. We learned that showing data is as crucial as gathering it. Creating clean, readable dashboards taught us how visuals can make people grasp complicated trends instantly. It also taught us how to view data with a more analytical eye.

In addition to the technical aspect, we also learned in teamwork. Coordinate work, combining disparate parts of the project, and ensuring all fell into place by the end compelled us to interact more effectively and plan ahead.

Overall, it was a solid mix of hands-on practice, problem-solving, and teamwork—all of which we’ll carry into future projects.

**IX. Challenges Faced**

One of the most difficult things was getting our SQL code to do what we wanted it to. Some queries seemed fine at first glance but gave us wrong or incomplete results, or simply refused to run. We debugged them for a while, sometimes it was a small syntax mistake, sometimes we needed to rework how we were joining tables or structuring the logic. It really made me appreciate how much it matters to truly understand how each table relates and how data moves through a query.

Learning Power BI was a learning curve too. The interface seemed easy, but as soon as we started establishing relationships and creating DAX formulas, it became complicated. We spent countless hours trying out tutorials and a whole lot of trial-and-error before we could finally create dashboards that told a story.

Team coordination was not bug-free either. With all of us working on separate pieces—SQL, diagrams, Power BI—it wasn't always a question of being on the same page. We would sometimes end up redoing work or wait on each other without even realizing it. We needed to coordinate better by checking in and more clearly breaking up work.

In the end, it all made the project even more worthwhile. We were left with enhanced technical abilities—and a clearer sense of how to work well together.

**X. Conclusion**

This was not just an assignment—it was a whole learning experience that brought together everything we've learned so far, and forced us to think outside the syllabus and use tools and concepts beyond it.

We began with a clear, tangible problem: artists can't keep up with or make sense of how their music is doing on different platforms. From that, we had to create a system that not only stored their data well but made their data meaningful. That involved a lot of time thinking about how to design the database, how to eliminate redundancy through normalization, and how to write the proper SQL queries to ask meaningful questions—such as "What's my most streamed track?" or "What's the most popular genre?"

But it didn't stop at the backend. What actually kept the project going was the Power BI integration. Seeing our data turn into graphs, charts, and interactive dashboards made us realize so much more the importance of good visualization. It's not about having the data—it's about making it available for someone to actually utilize it. And in this case, making it available for artists to make better-informed decisions about their music careers.

We also gained a lot in terms of being a team player. We each played different parts, and we had to have confidence in each other in order to make everything work. From creating the ER diagram, the SQL writing, or trying to decipher Power BI from scratch, everyone brought something of value to the table. That's how real projects actually get done.

Overall, this project showed us how to do a whole lifecycle of a data product—how to scope out a problem and design a solution, release it into the world, iterate on it, and show it so that people can use and understand it. It was challenging, but in a positive way. And most importantly, it showed us how the tools we're learning can be used to have an impact beyond the classroom.

# 