MUSIC STREAMING SERVICE

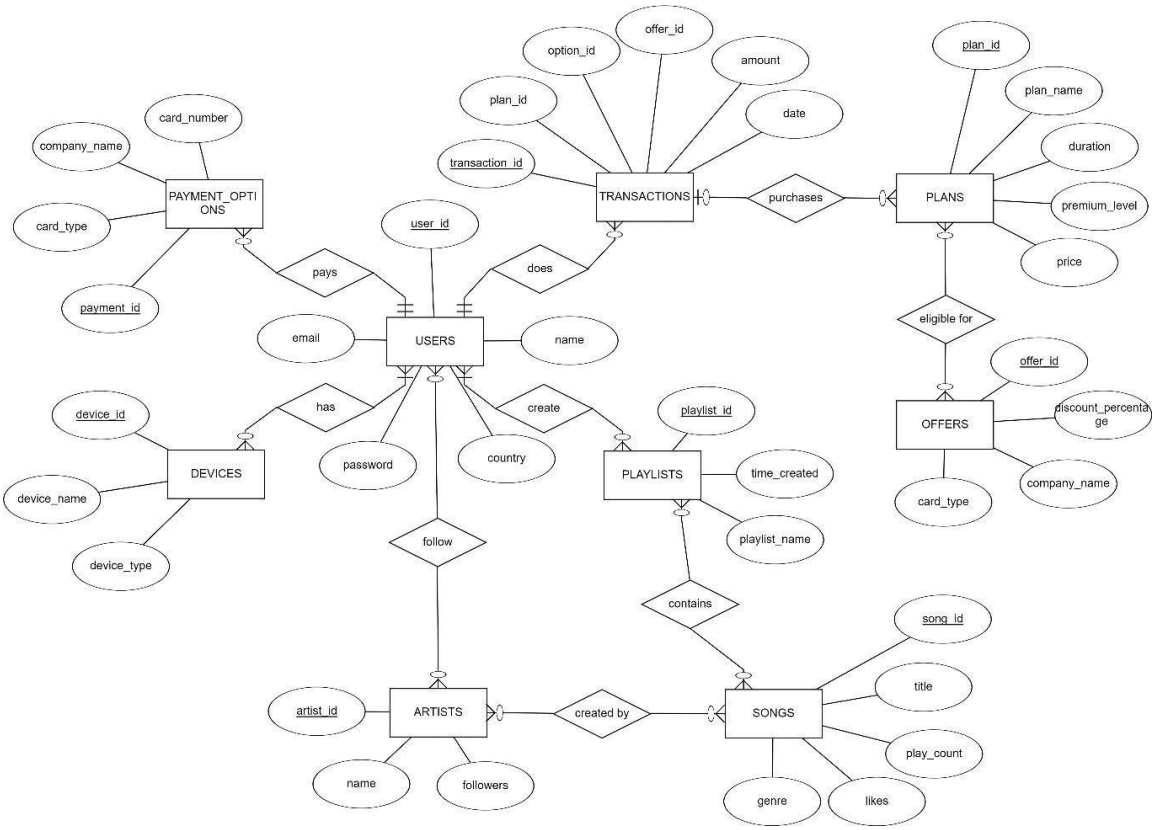
Background:

In the digital age, music streaming services have become a cornerstone of entertainment, allowing users to access a vast library of songs, create personalized playlists, and discover new artists. These platforms rely on sophisticated data models to manage user information, content, and interactions efficiently. This project aims to outline the foundational data structure required to support a music streaming service that caters to diverse user needs while ensuring a seamless and engaging experience.

Objective:

The primary goal is to design a robust and scalable data model that supports the core functionalities of a music streaming service. This includes managing user profiles, handling music content, facilitating playlist creation, processing transactions for premium plans, and offering promotional discounts. The data model must ensure data integrity, facilitate complex queries for recommendations, and support analytics for business insights.

Enhanced – Entity Relationship Model:



Entities and Relationships:

Users: Central to the service, users have attributes like user_id, name, email, password, and country. This entity supports the creation of personalized user profiles.

Playlists: Users can create multiple playlists (playlist_id, user_id, playlist_name, time_created), enabling them to organize songs according to their preferences.

Songs: The core content of the service, each song (song_id, title, genre, artist_id, likes, play_count) is associated with an artist and can be liked and played multiple times.

Artists: Artists (artist_id, name, followers) create songs and can be followed by users, fostering a community around music creators.

Payment Options: Users can have multiple payment options (user_id, card_type, company_name, card_number), enabling flexibility in transaction processes.

Devices: To accommodate various listening preferences, users can register multiple devices (device_id, device_name, device_type, user_id).

Transactions: Transactions (transaction_id, user_id, plan_id, option_id, offer_id, amount, date) record the purchase of subscription plans, linking users to the plans and offers availed.

Plans: Subscription plans (plan_id, plan_name, duration, premium_level, price) offer users access to premium features based on different tiers.

Offers: Offers (offer_id, card_type, company_name, discount_percentage) provide discounts on subscription plans, encouraging users to upgrade.

Challenges and Considerations:

- The data model must efficiently handle many-to-many relationships, such as between playlists and songs, to support dynamic content management and user interactions.
- Ensuring data security, especially for user information and payment options, is critical.
- The model should be scalable to accommodate a growing library of songs, increasing numbers of users, and expanding transaction records.

Reference Data:

Users Table:

UserID	Name	Email	Password	Country
1	Alex Smith	alex.smith@example.com	hashed_password_str1	USA
2	Maria Garcia	maria.garcia@example.com	hashed_password_str2	Canada
3	John Doe	john.doe@example.com	hashed_password_str3	UK

Artists Table:

ArtistID	Name	Followers
1	The Midnight	2500000
2	Synth Kid	1500000
3	Retro Runner	850000

Songs Table:

SongID	Title	Genre	ArtistID	Likes	Play Count
1	Lost Boy	Synthwave	1	500000	1500000
2	Neon Sun	Synthwave	2	350000	1200000
3	Flashback	Synthwave	3	250000	950000

Devices Table:

DeviceID	DeviceName	DeviceType	UserID
1	Alex's iPhone	iOS	1
2	Maria's Galaxy	Android	2
3	John's Pixel	Android	3

Plans Table:

PlanID	PlanName	Duration	PremiumLevel	Price
1	Premium	30	1	9.99
2	Family	30	1	14.99
3	Student	30	1	4.99

Offers Table:

OfferID	CardType	CompanyName	DiscountPercentage
1	Visa	Bank of Example	10
2	MasterCard	Mega Bank	15
3	Amex	Credit Union Hero	20

Transactional Data:

Transactions Table:

TransactionID	UserID	PlanID	OptionID	OfferID	Amount	Date
1	1	1	1	null	9.99	2024-02-08
2	2	2	null	1	13.49	2024-02-07
3	3	3	null	null	4.99	2024-02-09

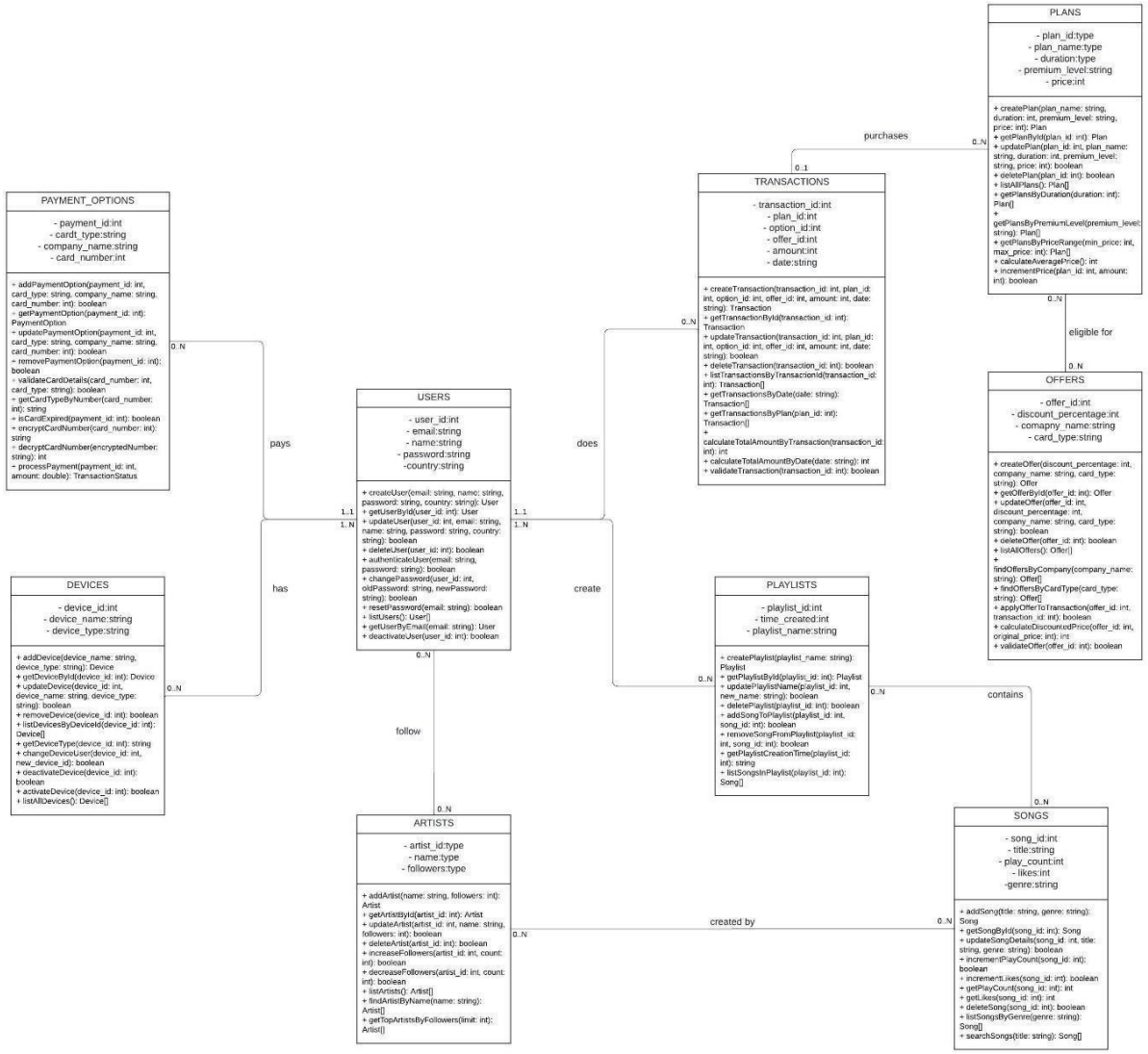
Songs in Playlist Table:

PlaylistID	SongID
1	1
1	2
2	3

Playlists Table:

PlaylistID	UserID	PlaylistName	TimeCreated
1	1	Chill Beats	2024-02-01 10:00:00
2	2	Workout Mix	2024-02-02 15:30:00
3	3	Study Session	2024-02-03 20:45:00

UML Class Diagram:



Mapping Conceptual model to Relational model :

Table Name – BOLD, Primary Key – Underlined, Foreign Key – Italics and blue in color

Payment_Options (payment_id, card_type, company_name, card_number, *user_id*) *user_id* foreign key refers to User_id in the relation **USERS:** Null not allowed

Devices (device_id, device_name, device_type)

Users (user_id, email, name, password, country)

Has (*user_id, device_id*)

User_id and device_id combined are the primary keys.

User_id foreign key refers to user_id in the relation **Users:** Null not allowed

Device_id foreign key refers to device_id in the relation **Devices:** Null not allowed

Transactions (transaction_id, plan_id, option_id, offer_id, amount, date, *user_id*)

User_id foreign key refers to user_id in the relation **Users:** Null not allowed

Playlists (playlist_id, time_created, playlist_name)

Create (*user_id, playlist_id*)

User_id and playlist_id combined are the primary keys.

User_id foreign key refers to user_id in the relation **Users:** Null not allowed

Playlist_id foreign key refers to playlist_id in the relation **Playlists:** Null not allowed

Songs (song_id, title, play_count, likes, genre)

Contains (*playlist_id, song_id*)

song_id and playlist_id combined are the primary keys.

Song_id foreign key refers to song_id in the relation **Songs:** Null not allowed

Playlist_id foreign key refers to playlist_id in the relation **Playlists:** Null not allowed

Artists (artist_id, name, followers)

Follows (*user_id, artist_id*)

User_id and artist_id combined are the primary keys.

User_id foreign key refers to user_id in the relation **Users:** Null not allowed

artist_id foreign key refers to artist_id in the relation **Artists:** Null not allowed

Plans (plan_id, plan_name, duration, premium_level, price)

Offers (offer_id, discount_percentage, company_name, card_type)

Eligible_for (*plan_id, offer_id*)

Plan_id and offer_id combined are the primary keys.

Plan_id foreign key refers to plan_id in the relation **Plans:** Null not allowed

Offer_id foreign key refers to offer_id in the relation **Offers:** Null not allowed

Purchases (transaction_id, *plan_id*)

Plan_id and transaction_id combined are the primary keys.

Plan_id foreign key refers to plan_id in the relation **Plans:** Null not allowed

Transaction_id foreign key refers to transaction_id in the relation **Transactions:** Null not allowed

SQL Queries:

Query 1: Top Devices Used by Users
Analytical Purpose: The analytical purpose of this query is to identify the most popular device types used by users, based on the frequency of each device type in the data. This information could be valuable for understanding user preferences, optimizing user experiences across different devices, or making decisions about which devices to prioritize in development and testing.

```
SELECT device_type, COUNT(*) AS UserCount
FROM devices
JOIN has ON devices.device_iD = has.device_iD
GROUP BY device_type
ORDER BY UserCount DESC
LIMIT 10;
```

device_type	UserCount
Tablet	14029
Phone	13722
Laptop	13679
Desktop	13529

Query 2: Monthly Revenue from Subscriptions
Analytical Purpose: The analytical purpose of this query is to provide a monthly breakdown of total revenue over time. This information is useful for assessing financial performance, budgeting, forecasting, and comparing revenue across different months.

```
SELECT DATE_FORMAT(date, '%Y-%m') AS PurchaseMonth, Concat('$ ',round(SUM(amount),2)) AS
TotalRevenue
FROM transactions
GROUP BY PurchaseMonth
ORDER BY PurchaseMonth
LIMIT 10;
```

PurchaseMonth	TotalRevenue
2021-01	$ 20217.31
2021-02	$ 18452
2021-03	$ 20732.69
2021-04	$ 19609.29
2021-05	$ 20283.77
2021-06	$ 19730.45
2021-07	$ 19614.29
2021-08	$ 19872.19
2021-09	$ 19669.23
2021-10	$ 20895.62

Query 3: Subscription Plan Popularity
Analytical Purpose: The analytical purpose of this query is to analyze the popularity of different subscription plans by counting the number of subscribers for each plan. This information helps understand which plans are most attractive to customers and can inform pricing, marketing, and product development strategies.

```
SELECT plan_name, COUNT(purchases.plan_iD) AS Subscribers
FROM plans
JOIN purchases ON plans.plan_iD = purchases.plan_iD
GROUP BY plan_name
ORDER BY Subscribers DESC
LIMIT 10;
```

plan_name	Subscribers
Student	3789
Free	3770
Family	3752
Premium	3689

Query 4: Ranking Artists by Popularity
Analytical Purpose: The analytical purpose of this query is to rank artists based on their popularity by counting the number of followers for each artist. This information helps gauge artist popularity, understand fan engagement, and potentially inform decisions related to promotions, partnerships, or content featuring popular artists.

```
SELECT artist, COUNT(follows.user_id) AS FollowerCount,
    RANK() OVER (ORDER BY COUNT(follows.user_id) DESC) AS PopularityRank
FROM artist_table_SQL
JOIN follows ON artist_table_SQL.artist_iD = follows.user_id
GROUP BY artist
ORDER BY FollowerCount DESC
LIMIT 10;
```

artist	FollowerCount	PopularityRank
Katy Perry	513	1
Lady Gaga	497	2
Justin Bieber	483	3
Maroon 5	472	4
Bruno Mars	297	5
Pitbull	285	6
The Chainsmokers	269	7
Ed Sheeran	267	8
Jennifer Lopez	263	9
David Guetta	262	10

Query 5: Number of Songs by Artist
Analytical Purpose: The analytical purpose of this query to identify the top 10 most prolific artists based on the number of songs they have recorded. This information can be useful for understanding artist productivity, identifying key contributors to a music library, and potentially guiding decisions about featuring or promoting certain artists based on the depth of their discography.

```
SELECT artist, COUNT(*) AS NumberOfSongs
FROM Artist_table_SQL
JOIN Songs_table ON Artist_table_SQL.artist_ID = Songs_table.artist_ID
GROUP BY artist
ORDER BY NumberOfSongs DESC
LIMIT 10;
```

artist	NumberOfSongs
Rihanna	9
Katy Perry	9
Lady Gaga	8
Kesha	7
Bruno Mars	7
The Black Eyed Peas	5
Britney Spears	5
Jennifer Lopez	5
Nicki Minaj	4
BeyoncÃ©	4

Query 6: Estimating User Lifetime Value
Analytical Purpose: The analytical purpose is to estimate the total revenue generated by each user over their lifetime. This helps identify high-value users, understand user spending patterns, and potentially inform decisions around user acquisition, retention, and targeted marketing efforts to maximize customer LTV.

```
SELECT DISTINCT Users.user_id, Users.email,
SUM(Plans.price) OVER (PARTITION BY Users.user_id) AS estimated_ltv
FROM Users
JOIN Transactions ON Users.user_id = Transactions.user_id
JOIN Plans ON Transactions.plan_id = Plans.plan_id
LIMIT 10;
```

user_id	email	estimated_ltv
1	jamie.smith@example.com	14.99
2	drew.williams@example.com	14.99
4	quinn.johnson@example.com	14.99
6	drew.garcia@example.com	29.98
7	alex.rodriguez@example.com	14.98
8	morgan.brown@example.com	19.98
9	morgan.wilson@example.com	24.97
12	alex.miller@example.com	14.99
13	drew.wilson@example.com	34.97
15	alex.jones@example.com	9.99

Query 7: User Satisfaction Ratio for Songs
Analytical Purpose: The analytical purpose is to measure user engagement and satisfaction with each song. A higher satisfaction ratio indicates that a larger proportion of users who played the song also liked it, suggesting that the song is well-received by the audience. This information can be valuable for understanding user preferences, identifying popular songs, and potentially guiding recommendations or playlist curation to improve user engagement and satisfaction.

```
SELECT song_ids, title,
round(CAST(likes AS FLOAT) / play_count,2) AS satisfaction_ratio
FROM songs_table
WHERE play_count > 0
ORDER BY satisfaction_ratio DESC
LIMIT 10;
```

song_ids	title	satisfaction_ratio
3000	Hey, Soul Sister	10.45
3001	Love The Way You Lie	6
3026	Naturally	5.62
3129	Dance Again	5.08
3015	OMG (feat. will.i.am)	4.84
3089	I'm Into You	4.02
3140	Wake Me Up	3.96
3109	I Knew You Were Trouble.	3.5
3081	You And I	3.37
3090	Papi	3.32

Query 8: Diversity of User Playlists
Analytical Purpose: The analytical purpose is to determine which playlists offer the most variety in terms of musical genres. Playlists with a higher number of unique genres cater to diverse tastes and expose listeners to a wider range of music. This information can be useful for understanding playlist composition, identifying playlists that appeal to eclectic listeners, and potentially guiding playlist creation or recommendation strategies to enhance user engagement and satisfaction by offering more diverse content.

```
SELECT Playlists.playlist_id,
COUNT(DISTINCT songs_table.`top genre`) AS unique_genres
FROM Playlists
JOIN Contains ON Playlists.playlist_id = Contains.playlist_id
JOIN songs_table ON Contains.song_id = songs_table.song_ids
GROUP BY Playlists.playlist_id
ORDER BY unique_genres DESC
LIMIT 10;
```

playlist_id	unique_genres
3524	3
910	3
4484	3
7500	3
5113	3
354	2
13	2
384	2
834	2
623	2

Query 9: Average Song Likes by Genre
Analytical Purpose: The analytical purpose is to determine which playlists offer the most variety in terms of musical genres. Playlists with a higher number of unique genres cater to diverse tastes and expose listeners to a wider range of music. This information can be useful for understanding playlist composition, identifying playlists that appeal to eclectic listeners, and potentially guiding playlist creation or recommendation strategies to enhance user engagement and satisfaction by offering more diverse content.

```
SELECT `top genre`, round(AVG(likes),0) AS average_likes FROM songs_table
GROUP BY `top genre`
ORDER BY average_likes DESC
LIMIT 10;
```

top genre	average_likes
neo mellow	533
canadian hip hop	501
acoustic pop	490
art pop	487
detroit hip hop	474
british soul	429
chicago rap	416
indie pop	410
baroque pop	406
canadian pop	360

Query 10: User Ranking by Total Spend
Analytical Purpose: The analytical purpose is to identify the highest-spending users and rank them based on their total expenditure. This information can be valuable for understanding user behavior, identifying top customers, and potentially informing targeted marketing, loyalty programs, or personalized promotions for high-value users. By focusing on the top 10 spenders, the query provides insights into the most significant contributors to revenue and helps prioritize efforts to retain and engage these valuable customers.

```
WITH UserSpend AS (
  SELECT
    users.user_id,
    users.name as user_name,
    SUM(transactions.amount) AS total_spend
  FROM transactions
  JOIN users ON transactions.user_id = users.user_id
  GROUP BY users.user_id,users.name
)
SELECT
  user_name,
  concat('$ ',round(total_spend,2)) as Total_Spend,
  RANK() OVER (ORDER BY total_spend DESC) AS spend_rank
FROM UserSpend
LIMIT 10;
```

user_name	Total_Spend	spend_rank
Taylor Miller	$ 284.14	1
Riley Brown	$ 273.65	2
Riley Miller	$ 223.44	3
Peyton Johnson	$ 218.03	4
Casey Johnson	$ 217.74	5
Taylor Jones	$ 212.61	6
Taylor Davis	$ 212.2	7
Peyton Miller	$ 211.45	8
Jordan Garcia	$ 209.23	9
Casey Brown	$ 203.9	10

No SQL Queries:

Query 1: Active Users by Country
Analytical Purpose: Purpose: Get a count of active users by country, assisting in regional marketing and content localization strategies.

```
db.Users.aggregate([
 {
  $group: {
   _id: "$country",
   active_users: { $sum: 1 }
  }
 },
 {
  $sort: { active_users: -1 }
 },
 {
  $limit: 10
 }
]);
```

```
[
  { _id: 'Canada', active_users: 2079 },
  { _id: 'France', active_users: 2027 },
  { _id: 'USA', active_users: 2002 },
  { _id: 'Germany', active_users: 1970 },
  { _id: 'UK', active_users: 1922 }
]
dheeraj>
```

Query 2: Most Used Payment Options
Analytical Purpose: Identify the most commonly used payment options, useful for optimizing payment processing partnerships.

```
db.Payment_Options.aggregate([
 {
  $group: {
   _id: "$card_type",
   count: { $sum: 1 }
  }
 },
 {
  $sort: { count: -1 }
 },
 {
  $limit: 10
 }
]);
```

Query 3: Top Spending Users
Analytical Purpose: Identify the top spending users by analyzing transactions, useful for targeting premium service offers.

```
db.Transactions.aggregate([
 {
  $group: {
   _id: "$user_id",
   total_spent: { $sum: "$amount" }
  }
 },
 {
  $lookup: {
   from: "Users",
   localField: "_id",
   foreignField: "user_id",
   as: "user_info"
  }
 },
 {
  $project: {
   user_id: "$_id",
   user_email: { $arrayElemAt: ["$user_info.email", 0] },
   total_spent: 1
  }
 },
 {
  $sort: { total_spent: -1 }
 },
 {
  $limit: 10
 }
]);
```

```
[
  {
    _id: 6651,
    total_spent: 284.14,
    user_id: 6651,
    user_email: 'taylor.miller@example.com'
  },
  {
    _id: 1060,
    total_spent: 273.65,
    user_id: 1060,
    user_email: 'riley.brown@example.com'
  },
  {
    _id: 4687,
    total_spent: 223.44,
    user_id: 4687,
    user_email: 'riley.miller@example.com'
  },
  {
    _id: 9423,
    total_spent: 218.03,
    user_id: 9423,
    user_email: 'peyton.johnson@example.com'
  },
  {
    _id: 646,
    total_spent: 217.74,
    user_id: 646,
    user_email: 'casey.johnson@example.com'
  },
```


Query 4: Track User Device Changes
Analytical Purpose: Monitor how frequently users change their devices, indicating engagement and potential for device-related offers.

```
db.Has.aggregate([
 {
  $lookup: {
   from: "Users",
   localField: "user_id",
   foreignField: "user_id",
   as: "user_info"
  }
 },
 {
  $unwind: "$user_info"
 },
 {
  $group: {
   _id: "$user_id",
   device_changes: { $sum: 1 },
   user_email: { $first: "$user_info.email" }
  }
 },
```

```
 {
  $sort: { device_changes: -1 }
 },
 {
  $limit: 10
 }
]);
```

```
[
  {
    _id: 1306,
    device_changes: 3,
    user_email: 'taylor.johnson@example.com'
  },
  {
    _id: 93,
    device_changes: 9,
    user_email: 'taylor.miller@example.com'
  },
  {
    _id: 3617,
    device_changes: 8,
    user_email: 'peyton.brown@example.com'
  },
  {
    _id: 4575,
    device_changes: 9,
    user_email: 'quinn.rodriguez@example.com'
  },
  {
    _id: 6863,
    device_changes: 4,
    user_email: 'jordan.johnson@example.com'
  },
  {
    _id: 9963,
    device_changes: 2,
    user_email: 'alex.miller@example.com'
  },
  {
    _id: 2480,
    device_changes: 4,
    user_email: 'quinn.rodriguez@example.com'
  },
  {
    _id: 6018,
    device_changes: 10,
    user_email: 'drew.smith@example.com'
```

Connecting Python with MySQL:

In [1]:
```
#pip install mysql-connector-python
```

```
Collecting mysql-connector-python
  Downloading mysql_connector_python-8.3.0-cp310-cp310-win_amd64.whl (15.4 MB)
     ---------------------------------------- 15.4/15.4 MB 7.3 MB/s eta 0:00:00
Installing collected packages: mysql-connector-python
Successfully installed mysql-connector-python-8.3.0
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
#importing module
import mysql.connector
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
```

In [4]:
```
import mysql.connector
from mysql.connector import errorcode
try:
    cnx=mysql.connector.connect(host='127.0.0.1',
                                database='project',
                                user='root',
                                password='Vicky@1240')
except mysql.connector.Error as err:
    if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
        print("Something is wrong with your user name or password")
    elif err.errno == errorcode.ER_BAD_DB_ERROR:
        print("Database does not exist")
    else:
        print(err)
else:
    print("Successfully connected")
    mycursor = cnx.cursor()
```

```
Successfully connected
```

Query 1: Top Devices Used by Users

Analytical Purpose: The analytical purpose of this query is to identify the most popular device types used by users, based on the frequency of each device type in the data. This information could be valuable for understanding user preferences, optimizing user experiences across different devices, or making decisions about which devices to prioritize in development and testing.

In [41]:
```
mycursor.execute('''SELECT device_type, COUNT(*) AS UserCount FROM devices JOIN has ON
                    devices.device_iD = has.device_iD GROUP BY device_type ORDER BY UserCount DESC;''')
result_2=pd.DataFrame(mycursor.fetchall(), columns=['Device Type', 'User Count'])
```

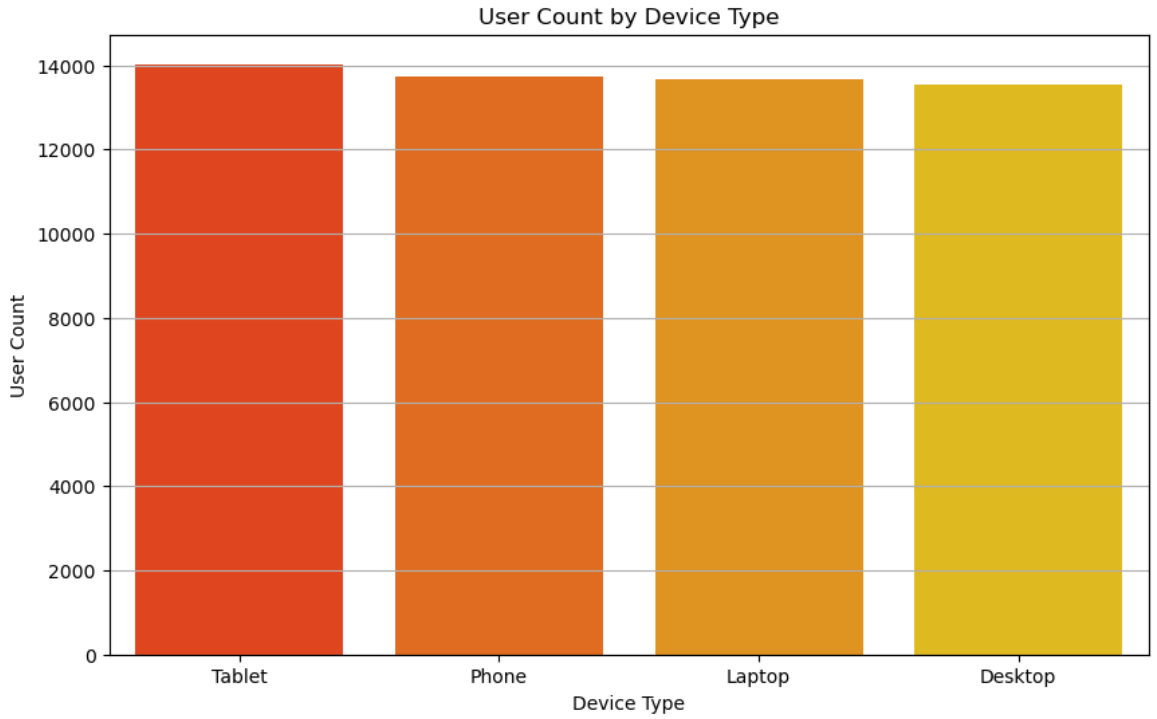
In [42]:
```
result_2
```

Out[42]:

	Device Type	User Count
0	Tablet	14029
1	Phone	13722
2	Laptop	13679
3	Desktop	13529

Visualization:

In [19]:
```
# Plotting a vertical bar chart for device types and user counts
plt.figure(figsize=(10, 6))
sns.barplot(x=result_2['Device Type'], y=result_2['User Count'], palette='autumn')
plt.title('User Count by Device Type')
plt.xlabel('Device Type')
plt.ylabel('User Count')
plt.grid(axis='y')
plt.show()
```



Query 2: Monthly Revenue from Subscriptions

Analytical Purpose: The analytical purpose of this query is to provide a monthly breakdown of total revenue over time. This information is useful for assessing financial performance, budgeting, forecasting, and comparing revenue across different months.

```
In [43]: mycursor.execute('''SELECT DATE_FORMAT(date, '%Y-%m') AS PurchaseMonth, round(SUM(amount),2) AS TotalRevenue
                              FROM transactions GROUP BY PurchaseMonth ORDER BY PurchaseMonth LIMIT 10;''')
         result_3=pd.DataFrame(mycursor.fetchall(), columns=['Purchase Month', 'Total Revenue'])
```

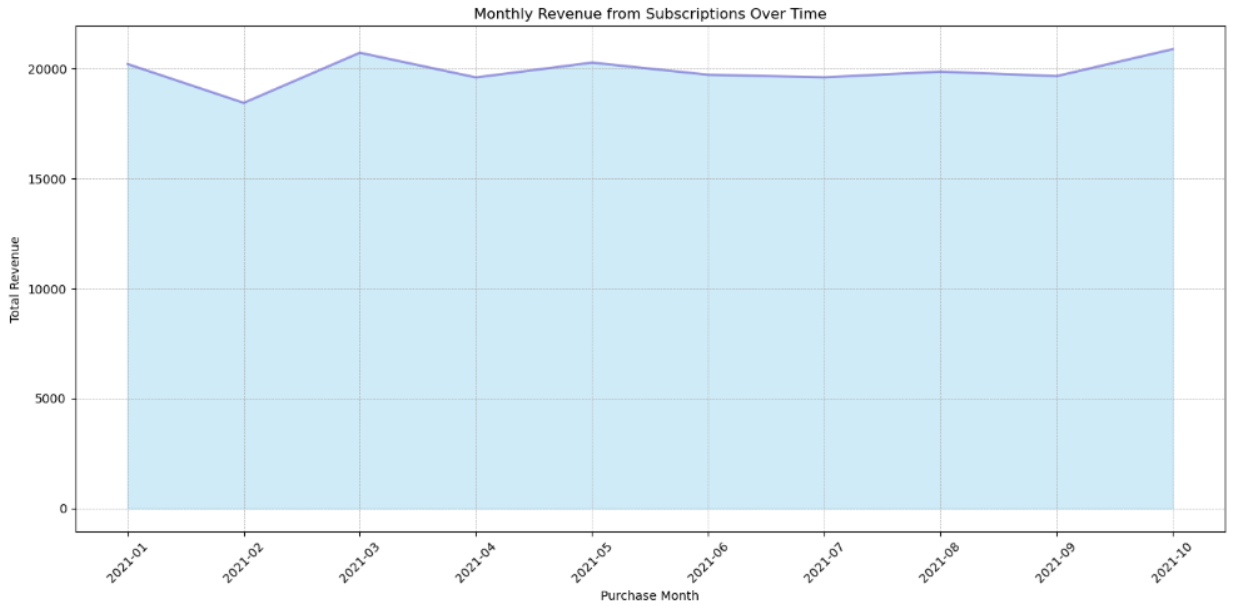
```
In [44]: result_3
```

Out[44]:

	Purchase Month	Total Revenue
0	2021-01	20217.31
1	2021-02	18452.00
2	2021-03	20732.69
3	2021-04	19609.29
4	2021-05	20283.77
5	2021-06	19730.45
6	2021-07	19614.29
7	2021-08	19872.19
8	2021-09	19669.23
9	2021-10	20895.62

Visualization:

```
In [45]: # Plotting
         plt.figure(figsize=(14, 7))
         plt.fill_between(result_3['Purchase Month'], result_3['Total Revenue'], color="skyblue", alpha=0.4)
         plt.plot(result_3['Purchase Month'], result_3['Total Revenue'], color="Slateblue", alpha=0.6, linewidth=2)
         plt.title('Monthly Revenue from Subscriptions Over Time')
         plt.xlabel('Purchase Month')
         plt.ylabel('Total Revenue')
         plt.xticks(rotation=45)
         plt.grid(True, which='both', linestyle='--', linewidth=0.5)
         plt.tight_layout()
         plt.show()
```



Query 3: Subscription Plan Popularity

Analytical Purpose: The analytical purpose of this query is to analyze the popularity of different subscription plans by counting the number of subscribers for each plan. This information helps understand which plans are most attractive to customers and can inform pricing, marketing, and product development strategies.

```
In [14]: mycursor.execute('''SELECT plan_name, COUNT(purchases.plan_iD) AS Subscribers FROM plans JOIN purchases
                    ON plans.plan_iD = purchases.plan_iD GROUP BY plan_name ORDER BY Subscribers DESC;''')
         result_4=pd.DataFrame(mycursor.fetchall(), columns=['Plan Name', 'Number of Subscribers'])
```

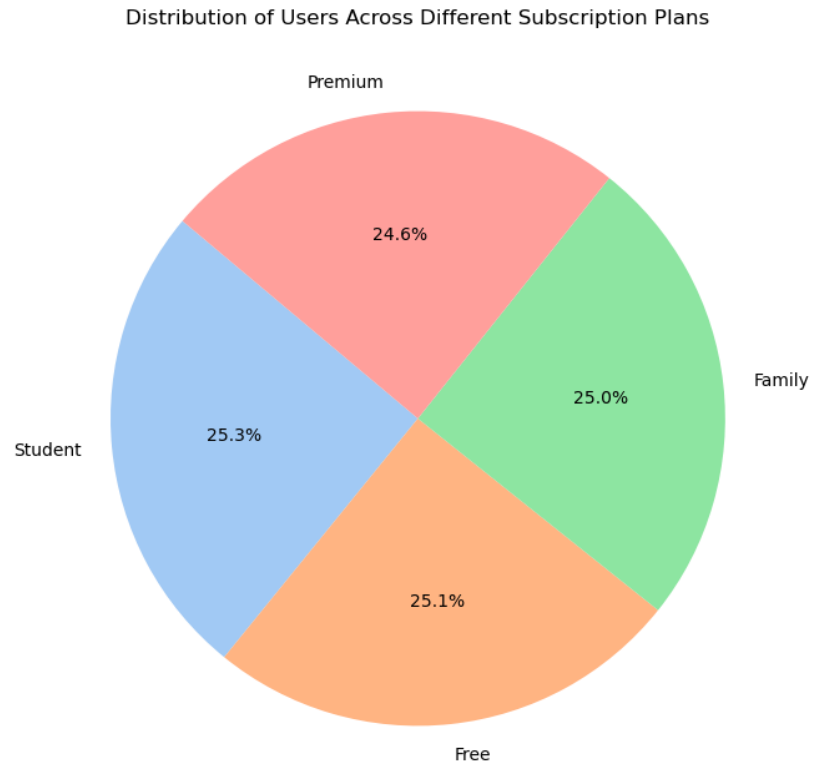
```
In [15]: result_4
```

Out[15]:

	Plan Name	Number of Subscribers
0	Student	3789
1	Free	3770
2	Family	3752
3	Premium	3689

Visualization:

```
In [24]: # Plotting the pie chart
         plt.figure(figsize=(10, 8))
         plt.pie(result_4['Number of Subscribers'], labels=result_4['Plan Name'], autopct='%1.1f%%', startangle=140, colors=sns.color_pale
         plt.title('Distribution of Users Across Different Subscription Plans')
         plt.show()
```



Query 4: Ranking Artists by Popularity

Analytical Purpose: The analytical purpose of this query is to rank artists based on their popularity by counting the number of followers for each artist. This information helps gauge artist popularity, understand fan engagement, and potentially inform decisions related to promotions, partnerships, or content featuring popular artists.

In [16]:
```
mycursor.execute('''SELECT artist, COUNT(follows.user_id) AS FollowerCount, RANK() OVER (ORDER BY COUNT(follows.user_id) DESC)
        AS PopularityRank FROM artist_table_SQL JOIN follows ON artist_table_SQL.artist_iD = follows.user_id
        GROUP BY artist ORDER BY FollowerCount DESC LIMIT 10;''')
result_5=pd.DataFrame(mycursor.fetchall(), columns=['Artist Name', 'Number of Followers', 'Popularity Rank'])
```

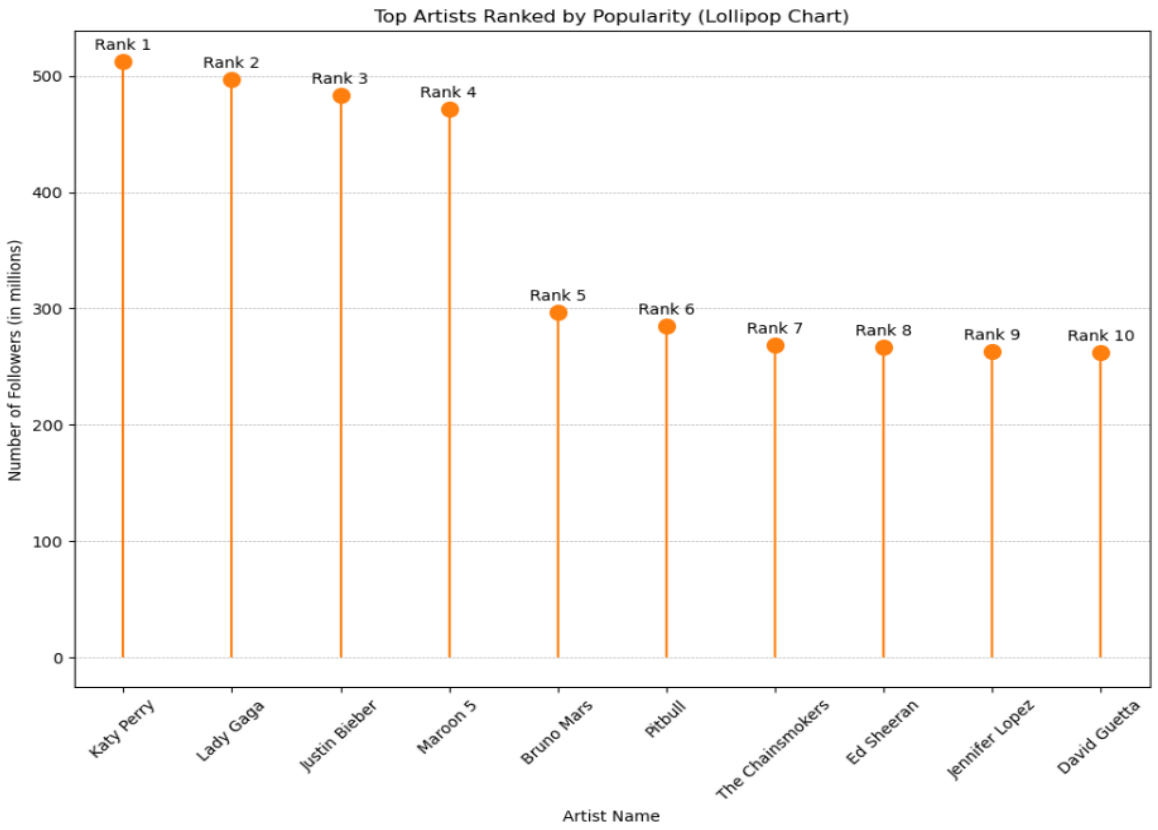
In [17]:
```
result_5
```

Out[17]:

	Artist Name	Number of Followers	Popularity Rank
0	Katy Perry	513	1
1	Lady Gaga	497	2
2	Justin Bieber	483	3
3	Maroon 5	472	4
4	Bruno Mars	297	5
5	Pitbull	285	6
6	The Chainsmokers	269	7
7	Ed Sheeran	267	8
8	Jennifer Lopez	263	9
9	David Guetta	262	10

Visualization:

In [40]:
```
# Creating a lollipop chart for artist popularity
plt.figure(figsize=(10, 8))

popularity_rank = result_5['Popularity Rank']

# The 'stem' part
plt.stem(result_5['Artist Name'], result_5['Number of Followers'], linefmt="C1-", basefmt=" ")

# The 'candy' part
plt.scatter(result_5['Artist Name'], result_5['Number of Followers'], color="C1", s=100, label=result_5['Number of Followers'], z

# Adding the popularity rank as text labels next to the 'candies'
for i, (artist, followers) in enumerate(zip(result_5['Artist Name'], result_5['Number of Followers'])):
    plt.text(i, followers + 10, f'Rank {popularity_rank[i]}', ha='center')

plt.title('Top Artists Ranked by Popularity (Lollipop Chart)')
plt.xlabel('Artist Name')
plt.ylabel('Number of Followers (in millions)')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', linewidth=0.5)
plt.tight_layout()  # Adjust layout to fit all x labels
plt.show()
```



Query 5: Number of Songs by Artist

Analytical Purpose: The analytical purpose of this query to identify the top 10 most prolific artists based on the number of songs they have recorded. This information can be useful for understanding artist productivity, identifying key contributors to a music library, and potentially guiding decisions about featuring or promoting certain artists based on the depth of their discography.

```
In [6]: mycursor.execute('''SELECT artist, COUNT(*) AS NumberOfSongs FROM Artist_table_SQL JOIN Songs_table
                        ON Artist_table_SQL.artist_ID = Songs_table.artist_ID GROUP BY artist ORDER BY NumberOfSongs DESC LIMIT 10;''')
        result_1=pd.DataFrame(mycursor.fetchall(), columns=['Artist Name', 'Number of Songs'])
```

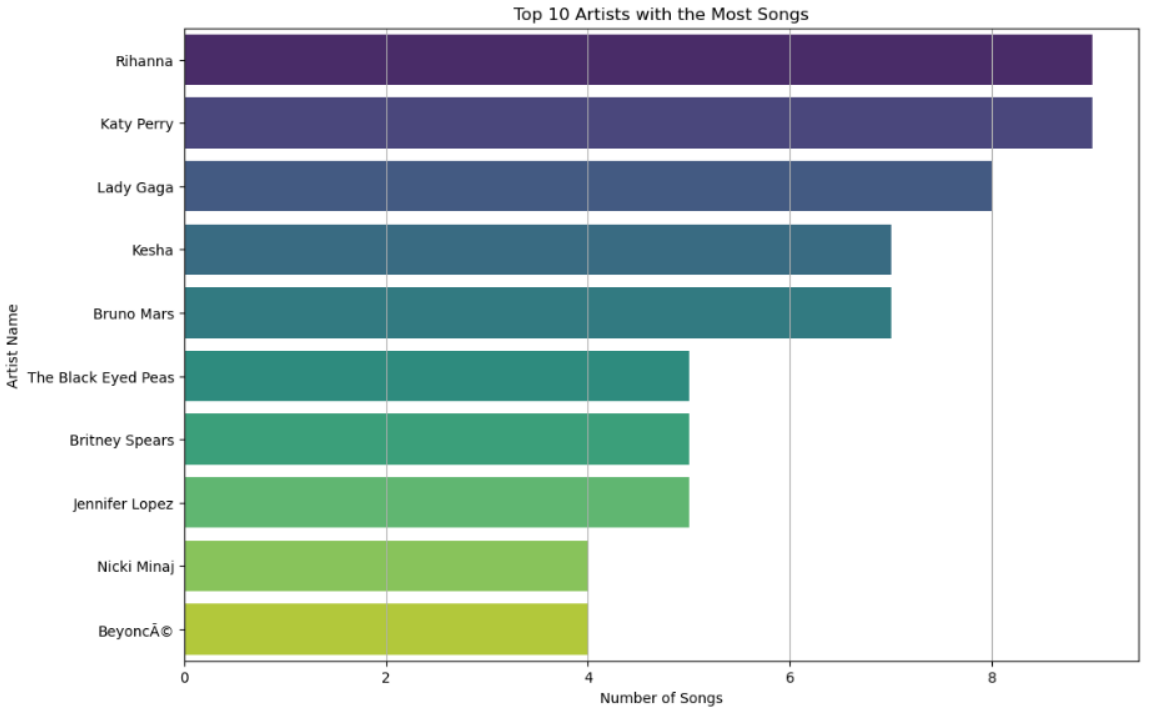
```
In [7]: result_1
```

Out[7]:

	Artist Name	Number of Songs
0	Rihanna	9
1	Katy Perry	9
2	Lady Gaga	8
3	Kesha	7
4	Bruno Mars	7
5	The Black Eyed Peas	5
6	Britney Spears	5
7	Jennifer Lopez	5
8	Nicki Minaj	4
9	BeyoncÃ©	4

Visualization:

In [18]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting
plt.figure(figsize=(12, 8))
sns.barplot(x=result_1['Number of Songs'], y=result_1['Artist Name'], palette='viridis')
plt.title('Top 10 Artists with the Most Songs')
plt.xlabel('Number of Songs')
plt.ylabel('Artist Name')
plt.grid(axis='x')
plt.show()
```



Conclusion:

This document presents a comprehensive and well-structured approach to designing and implementing a data model for a music streaming service. The enhanced Entity-Relationship (ER) model and UML class diagram provide a clear understanding of the entities, their attributes, and relationships, forming a solid foundation for the database design.

The SQL and NoSQL queries demonstrate the analytical capabilities of the data model, enabling the service to gain valuable insights into user behavior, artist popularity, revenue trends, and more. These queries, along with their analytical purposes, showcase how the data model can support data-driven decision-making and help optimize various aspects of the service, such as user experience, content curation, and financial planning.

The integration of Python with MySQL and the visualizations created using matplotlib further enhance the analytical potential of the data model. The ability to execute queries and visualize results programmatically allows for efficient and repeatable analysis, facilitating regular reporting and monitoring of key metrics.

Overall, the document serves as a valuable resource for understanding the data requirements, design considerations, and analytical possibilities for a music streaming service. The proposed data model and analytical queries provide a strong starting point for building a robust and data-driven platform that can adapt and grow with the needs of the service and its users.