BlockBeats

REVOLUTIONIZE MUSIC USING BLOCKCHAIN

BlockBeats:

Revolutionize Music Streaming with Blockchain

Karnaz Obaidullah

Table of Contents

Executive Summary

The purpose of this project is to conduct an explorative study into how to manage data using database design based on a use case. For this project I look at database design through the lens of a music streaming platform named BlockBeats.

The project's database design proposal outlines the structure and functionality of the relational database system, incorporating user requirements and industry standards. The database model includes entities such as Users, Artists, Albums, Songs, Playlists, Listening History, Revenue, and Blockchain Transactions, enabling efficient management of music-related data.

The project is composed of 5 Milestones. Throughout the project milestones, from EER and UML diagrams to logical model implementation in MySQL and NoSQL environments, rigorous testing and optimization have been conducted to ensure the database's reliability, scalability, and performance.

The journey begins with the development of Entity-Relationship (EER) and Unified Modeling Language (UML) diagrams. These visual representations provide a conceptual blueprint for the database structure, laying the groundwork for subsequent stages of implementation. By mapping out the relationships between entities such as Users, Artists, and Songs, BlockBeats gains clarity and insight into the intricate network of data interactions.

Building upon the conceptual foundation established in Milestone 1, BlockBeats transitions to the creation of a logical model in the form of a relational database. Primary and foreign keys are defined, ensuring data integrity and coherence across tables. This milestone marks a pivotal step towards the practical implementation of the database, setting the stage for efficient data management and retrieval.

With the relational model in place, BlockBeats proceeds to implement the database using MySQL, a robust and widely used relational database management system. Rigorous testing and optimization are conducted to validate the database's reliability, scalability, and performance. Key queries demonstrate the database's capability to extract valuable insights and support strategic decision-making, such as analyzing revenue generation, identifying popular songs and artists, and understanding user behaviors.

As BlockBeats prepares to transition from development to deployment, focus shifts towards creating an application interface for accessing the database. This is done using Python and Jupyter Lab and helps us visualize the data to help us gain insights in the data.

Finally, BlockBeats explores the implementation of the database in a NoSQL environment, specifically Neo4j. By leveraging graph structures, BlockBeats aims to visualize and analyze complex relationships between entities such as Artists, Songs, and Albums.
In conclusion, the BlockBeats project presents a comprehensive solution to the challenges faced by the music industry, offering a transformative platform that empowers artists, engages users, and ensures the integrity of music distribution and consumption. With its innovative approach and robust database infrastructure, BlockBeats is poised to disrupt the status quo and shape the future of music streaming.

Database Design Project Proposal

Background

The days of buying CDs and cassettes are gone and the era of streaming is underway. However, this has been a problem for Artists. They grapple with the dilemma of whether streaming services ultimately benefit or harm their bottom line.

There is a growing problem in the music industry of *ownership,* particularly concerning the distribution of revenue and control over artistic content in the era of streaming services and digital consumption.

The adoption of blockchain technology in the music industry addresses these concerns by offering decentralized distribution channels, transparent royalty payments, and immutable ownership records. It provides hope for artists to regain control over their music and receive fair compensation in an industry that has historically favored intermediaries over creators.

Blockchain is a transparent digital ledger that records transactions across a network of computers. Imagine it as a shared spreadsheet duplicated thousands of times across the network, updated and reconciled regularly. Once a transaction is recorded, it cannot be altered or deleted, making the ledger permanent and tamper-proof. This technology ensures transparency, security, and trust in transactions without the need for intermediaries.

About Us

BlockBeats is a music streaming platform for the next generation of music lovers. BlockBeats offers revolutionary music streaming that leverages blockchain technology to provide transparency, fairness, and enhanced rewards for artists and users alike. By combining the convenience of traditional music streaming with the security and transparency of blockchain, BlockBeats aims to disrupt the music industry and create a more equitable ecosystem for all stakeholders.

Business Problem Definition

This project will take the perspective of the music streaming service BlockBeats, an early-stage startup, and explore how it could work and provide stakeholders with a sustainable business model that benefits the Artists and Users of streaming services, as well as the business and its investors too.

Business Model

BlockBeats has 3 main revenue streams:

1. **Subscription Fees (Paid Streaming Service):**
    - BlockBeats offers a premium subscription tier granting users ad-free access to an extensive music library, exclusive content, and premium features like offline listening and high-definition audio quality.
    - Users can subscribe to monthly or yearly plans.

2. **Advertisement Revenue (Free Streaming Service):**
    - BlockBeats provides a free tier with basic features, allowing users to access the platform's content at no cost but with targeted advertisements.
    - Advertisers pay BlockBeats for ad placements, leveraging user demographics, listening habits, and engagement metrics to reach their target audience effectively.
    - This advertising revenue stream complements the subscription fees and monetizes the free user base.
3. **Blockchain-Based Direct Payments to Artists:**
    - BlockBeats integrates blockchain technology to enable micropayments for music streaming.
    - Users have the option to pay using cryptocurrency or blockchain-based tokens directly to the artists.
    - This innovative approach fosters a fair and transparent revenue-sharing model, ensuring that artists receive immediate compensation for their work.
    - BlockBeats earns a nominal transaction fee for facilitating these micropayments, contributing to its overall revenue.

Perspectives from Different Users

1. **Database Administrators**:
    - Database administrators will ensure the reliability, security, and performance of BlockBeats' database system. They handle tasks like monitoring, backup, recovery, and performance tuning. They also need to collaborate with developers to implement database designs and access controls that align with BlockBeats' needs.
2. **Database Designers**:
    - Database designers will create the structure and organization of BlockBeats' database to efficiently manage music-related data. They work with other stakeholders to define data models and schema designs, considering factors like data integrity and scalability for an optimal user experience.
3. **Data Analysts**:
    - Data analysts need to extract insights from BlockBeats' database to inform strategic decisions and product development. They use SQL queries, data visualization, and statistical analysis to identify trends and user behaviors that drive user engagement and business growth.
4. **Database Developers**:
    - Database developers will implement and maintain BlockBeats' database system, designing efficient schemas and queries. They work with programming languages and management systems to integrate the database with BlockBeats' application logic for seamless data flow.
5. **Information Architect:**
    - The Information Architect will collaborate with database designers, administrators, and analysts to align the database schema with the user needs and business goals of BlockBeats, optimizing the overall information architecture.

Theory for Database Design

As we will apply for Series A funding, we need to design our database properly to show potential venture capitalists and angel investors the value we can provide as a business.

We have the following entity types:

1. User:
   a. UserID (Primary Key): Unique identifier for each user.
   b. Username: Name chosen by the user for display.
   c. Email: Email address of the user.
   d. RegistrationDate: Date and time when the user registered.
   e. Country: Country of residence of the user.
   f. Age: Age of the user.
   g. Name: Name of user.
2. Artist:
   a. ArtistID (Primary Key): Unique identifier for each artist.
   b. Genre: Type of music the artist produces.
   c. Username: Name chosen by the artist for display.
   d. Email: Email address of the artist.
   e. RegistrationDate: Date and time when the artist registered.
   f. Country: Country of origin of the artist.
   g. Age: Age of the artist.
   h. Name: Name of the artist.
3. NonArtist:
   a. NonArtistID (Primary Key): Unique identifier for each non-artist.
   b. Username: Name chosen by the non-artist for display.
   c. Email: Email address of the non-artist user.
   d. RegistrationDate: Date and time when the non-artist user registered.
   e. Country: Country of residence of the non-artist user.
   f. Age: Age of the non-artist user.
   g. Name: Name of non-artist user.
4. Album:
   a. AlbumID (Primary Key): Unique identifier for each album.
   b. Title: Title of the album.
   c. ArtistID (Foreign Key referencing Artists): ID of the artist who created the album.
   d. ReleaseDate: Date when the album was released.
   e. Genre: Genre of music the album belongs to.
5. Song:
   a. SongID (Primary Key): Unique identifier for each song.
   b. Title: Title of the song.
   c. ArtistID (Foreign Key referencing Artists): ID of the artist who created the song.
   d. AlbumID (Foreign Key referencing Albums): ID of the album to which the song belongs.
   e. Duration: Length of the song in seconds.
   f. ReleaseDate: Date when the song was released.

6. Playlist:
    a. PlaylistID (Primary Key): Unique identifier for each playlist.
    b. UserID (Foreign Key referencing Users): ID of the user who created the playlist.
    c. Title: Title of the playlist.
    d. Description: Description or notes about the playlist.
    e. CreationDate: Date and time when the playlist was created.
7. PlaylistSongs:
    a. PlaylistID (Foreign Key referencing Playlists): ID of the playlist to which the song belongs.
    b. SongID (Foreign Key referencing Songs): ID of the song in the playlist.
8. ListeningHistory:
    a. HistoryID (Primary Key): Unique identifier for each entry in the listening history.
    b. UserID (Foreign Key referencing Users): ID of the user.
    c. SongID (Foreign Key referencing Songs): ID of the song listened to.
    d. Timestamp: Date and time when the song was listened to.
9. Revenue:
    a. TransactionID (Primary Key): A unique identifier for the revenue transaction.
    b. UserID (Foreign Key referencing Users): ID of the user generating revenue.
    c. Source: Sources of Revenue are SubscriptionRevenue, AdvertisingRevenue, and BlockchainTransactionFee.
    d. Date: Date on which revenue was earned.
    e. Amount: The Amount of Revenue generated in USD$.
10. BlockchainTransactions:
    a. TransactionID (Primary Key): Unique identifier for each blockchain transaction.
    b. UserID (Foreign Key referencing Users): ID of the user involved in the transaction.
    c. Amount: Value amount of cryptocurrency or tokens involved in the transaction given in USD$
    d. ArtistID (Foreign Key referencing Artists): ID of the artist receiving the payment.
    e. Timestamp: Date and time when the transaction occurred.
11. ArtistEarnings:
    a. ArtistID (Foreign Key referencing Artists): ID of the artist earning from blockchain transactions.
    b. TransactionID (Foreign Key referencing BlockchainTransactions): ID of the blockchain transaction associated with the earning.
    c. Amount: Value amount of cryptocurrency or tokens involved in the transaction given in USD$
    d. Timestamp: Date and time when the earning was recorded.

Requirements

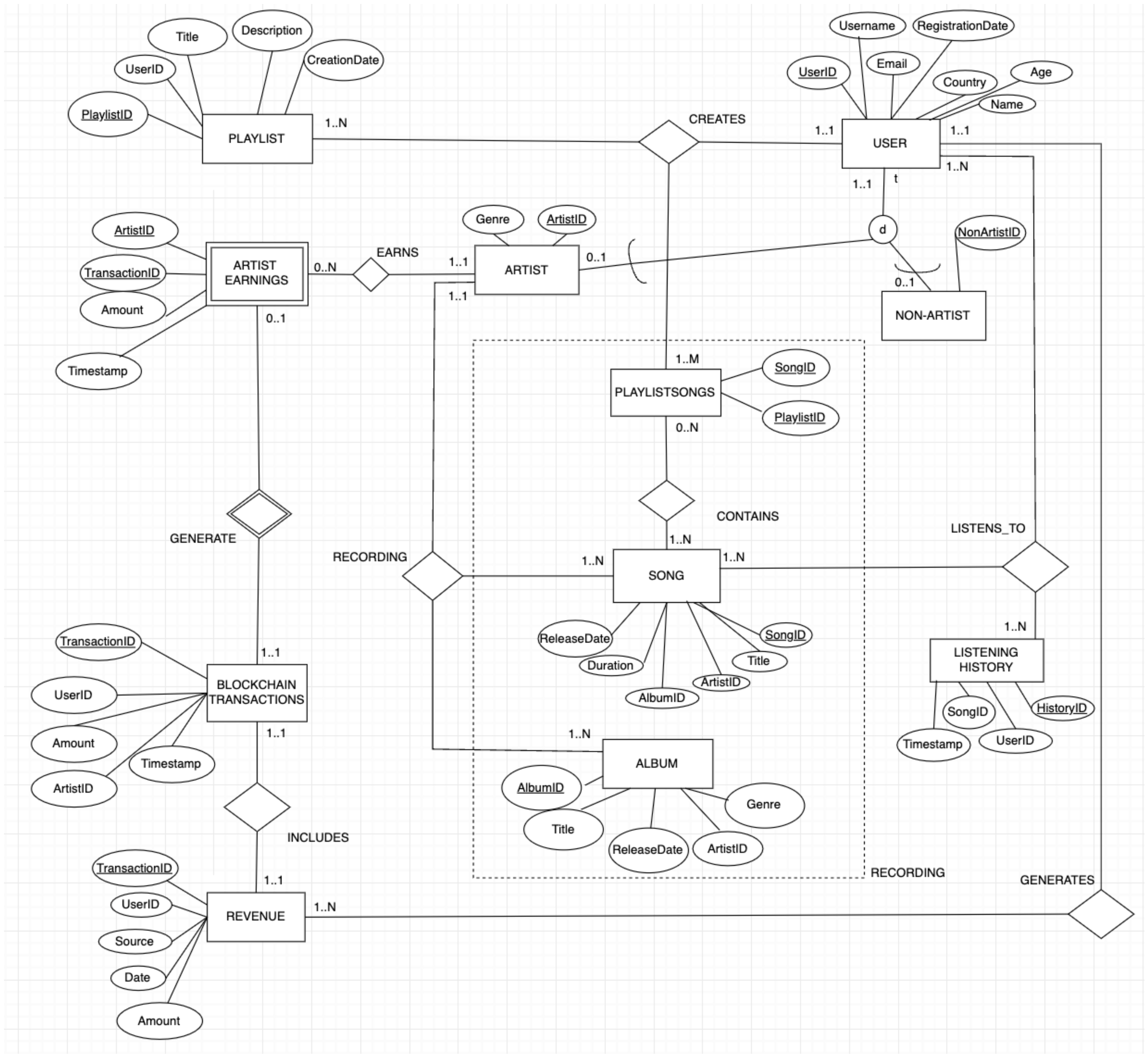
The requirements of our database system are as follows:

1. Each user can be an artist or a non-artist, but an artist or non-artist must be a user. This is a **One-to-One Relationship (User to Artist or NonArtist)**
2. Each user can have multiple playlists, but each playlist belongs to only one user. This is a **One-to-Many Relationship (Users to Playlists)**
3. An artist can have multiple albums, but each album belongs to only one artist. This is a **One-to-Many Relationship (Artists to Albums)**

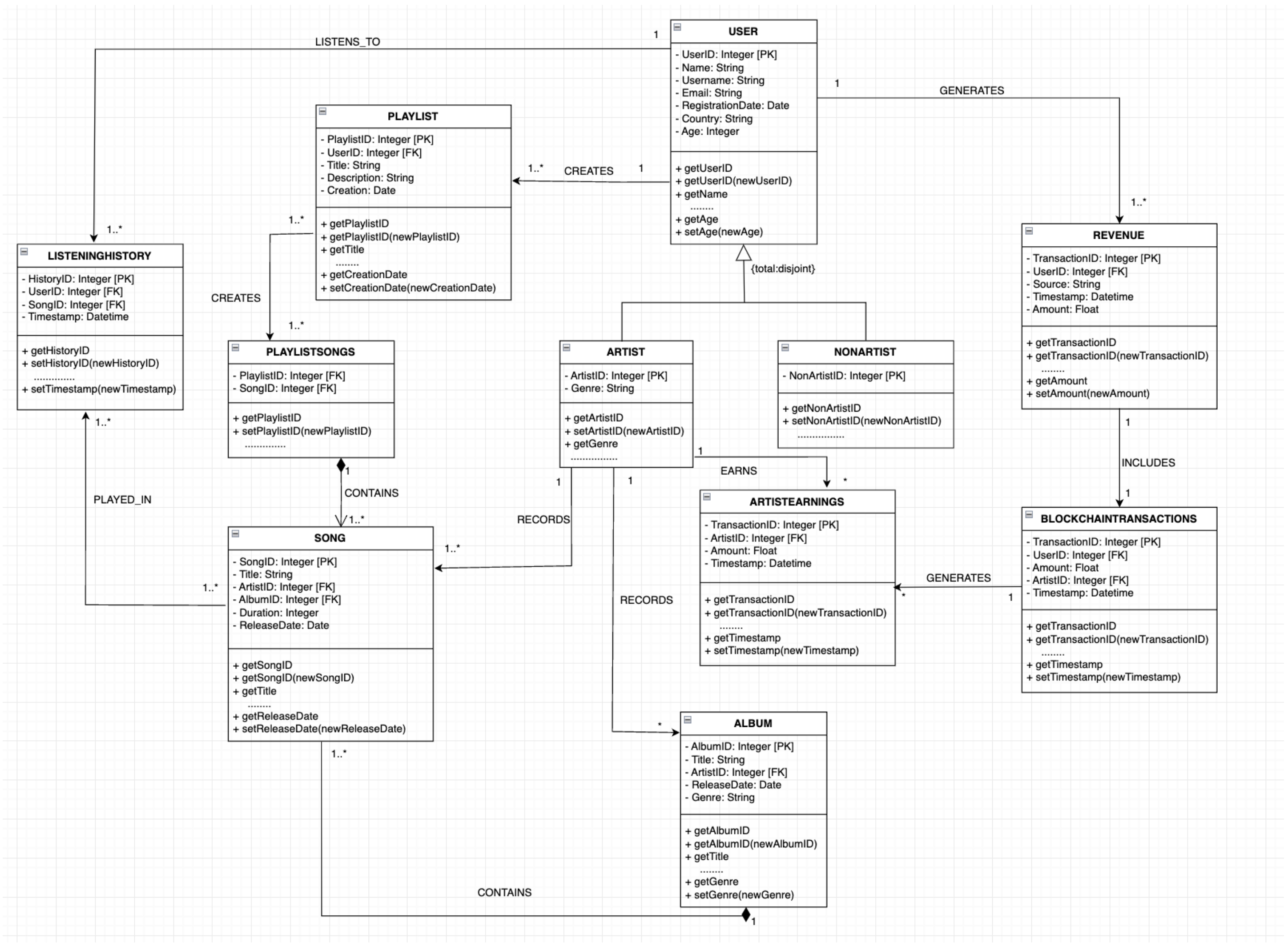
4. An artist can have multiple songs, but each song belongs to only one artist. This is a **One-to-Many Relationship (Artists to Songs)**
5. An album can have multiple songs, but each song belongs to only one album. This is a **One-to-Many Relationship (Albums to Songs)**
6. A playlist can have multiple songs, but each song can belong to only one playlist. This table is a junction table to implement a many-to-many relationship between playlists and songs. This is a **One-to-Many Relationship (Playlists to PlaylistSongs)**
7. A playlist can contain multiple songs, and a song can be included in multiple playlists. This is a **Many-to-Many Relationship (Playlists** to **Songs** (via **PlaylistSongs**)
8. Each user can have multiple entries in their listening history, but each entry belongs to only one user. This is a **One-to-Many Relationship (Users to ListeningHistory)**
9. Each song can have multiple entries in the listening history, many listening histories can have one song. This is a **One-to-Many Relationship (Songs to ListeningHistory)**
10. A user can listen to multiple songs, and a song can be listened to by multiple users. This is a **Many-to-Many Relationship (Users** to **Songs** via **ListeningHistory**)
11. Each user can have multiple revenue transactions (SubscriptionRevenue, AdvertisingRevenue, BlockchainTransactionFee) and therefore one user can generate multiple revenue transactions, but each revenue transaction is associated with only one user. This is a **One-to-Many Relationship (User** to **Revenue)**
12. Each transaction in Revenue relation that will come from blockchain transactions corresponds to one blockchain transaction entry in BlockchainTransactions relation. This relation holds all blockchain transactions. This is a **One-to-One Relationship (Revenue to BlockchainTransactions)**
13. Multiple blockchain transactions can involve a single user. Each entry in the BlockchainTransactions entity is associated with a user from the Users entity, representing the user's involvement in blockchain transactions. This is a **Many-to-One Relationship (BlockchainTransactions** to **Users** via **Revenue)**
14. Multiple blockchain transactions can involve a single artist. Each entry in the BlockchainTransactions entity is associated with an artist from the Artists entity, representing payments made to that artist through blockchain transactions. This is a **Many-to-One Relationship (BlockchainTransactions** to **Artists** via **ArtistEarnings**)
15. Each earning entry corresponds to one blockchain transaction. ArtistEarnings relation only holds blockchain transactions for one artist. This is a **One-to-One Relationship (ArtistEarnings to BlockchainTransactions)**
16. Each artist can have multiple earnings. This is because each artist can earn from multiple blockchain transactions. Each entry in Artist Earnings belongs to one Artist only. This is a **One-to-Many Relationship (Artists to ArtistEarnings)**

Milestone 1: Conceptual Data Modeling (EER and UML Diagrams)

EER Diagram:



UML Diagram:


USER
- UserID: Integer [PK]
- Name: String
- Username: String
- Email: String
- RegistrationDate: Date
- Country: String
- Age: Integer
+ getUserID
+ getUserID(newUserID)
+ getName
....
+ getAge
+ setAge(newAge)
LISTENS_TO
1
1..*
GENERATES
1
1..*
CREATES
1
1..*
{total:disjoint}
PLAYLIST
- PlaylistID: Integer [PK]
- UserID: Integer [FK]
- Title: String
- Description: String
- Creation: Date
+ getPlaylistID
+ getPlaylistID(newPlaylistID)
+ getTitle
....
+ getCreationDate
+ setCreationDate(newCreationDate)
CREATES
1..*
1..*
LISTENINGHISTORY
- HistoryID: Integer [PK]
- UserID: Integer [FK]
- SongID: Integer [FK]
- Timestamp: Datetime
+ getHistoryID
+ setHistoryID(newHistoryID)
....
+ setTimestamp(newTimestamp)
PLAYLISTSONGS
- PlaylistID: Integer [FK]
- SongID: Integer [FK]
+ getPlaylistID
+ setPlaylistID(newPlaylistID)
....
ARTIST
- ArtistID: Integer [PK]
- Genre: String
+ getArtistID
+ setArtistID(newArtistID)
+ getGenre
....
NONARTIST
- NonArtistID: Integer [PK]
+ getNonArtistID
+ setNonArtistID(newNonArtistID)
....
REVENUE
- TransactionID: Integer [PK]
- UserID: Integer [FK]
- Source: String
- Timestamp: Datetime
- Amount: Float
+ getTransactionID
+ getTransactionID(newTransactionID)
....
+ getAmount
+ setAmount(newAmount)
1
INCLUDES
1
1..*
PLAYED_IN
1
CONTAINS
1..*
1
RECORDS
1
1
EARNS
*
ARTISTEARNINGS
- TransactionID: Integer [PK]
- ArtistID: Integer [FK]
- Amount: Float
- Timestamp: Datetime
+ getTransactionID
+ getTransactionID(newTransactionID)
....
+ getTimestamp
+ setTimestamp(newTimestamp)
BLOCKCHAINTRANSACTIONS
- TransactionID: Integer [PK]
- UserID: Integer [FK]
- Amount: Float
- ArtistID: Integer [FK]
- Timestamp: Datetime
+ getTransactionID
+ getTransactionID(newTransactionID)
....
+ getTimestamp
+ setTimestamp(newTimestamp)
GENERATES
*
1
SONG
- SongID: Integer [PK]
- Title: String
- ArtistID: Integer [FK]
- AlbumID: Integer [FK]
- Duration: Integer
- ReleaseDate: Date
+ getSongID
+ getSongID(newSongID)
+ getTitle
....
+ getReleaseDate
+ setReleaseDate(newReleaseDate)
1..*
1..*
RECORDS
*
ALBUM
- AlbumID: Integer [PK]
- Title: String
- ArtistID: Integer [FK]
- ReleaseDate: Date
- Genre: String
+ getAlbumID
+ getAlbumID(newAlbumID)
+ getTitle
....
+ getGenre
+ setGenre(newGenre)
1..*
CONTAINS
1

Milestone 2: Logical Model (Relational Model)

Primary Key: Underlined **Foreign. Key: *Italicized***

The relational model is as follows:

1. User (UserID, Name, Username, Email, RegistrationDate, Country, Age)

2. Artist (ArtistID, Name, Genre, Country, Age, *UserID,* Username, Email, RegistrationDate)
   - *UserID* is a Foreign Key referencing UserID in User relation and NOT NULL

3. Non-Artist (NonArtistID, Name, Country, Age, *UserID,* Username, Email, RegistrationDate)
   - *UserID* is a Foreign Key referencing UserID in User relation and NOT NULL

4. Album (AlbumID, Title, *ArtistID*, ReleaseDate, Genre)
   - *ArtistID* is a Foreign Key referencing ArtistID in Artist relation and NOT NULL

5. Song (SongID, Title, *ArtistID*, *AlbumID*, Duration, ReleaseDate)
   - *ArtistID* is a Foreign Key referencing ArtistID in Artist relation and NOT NULL
   - *AlbumID* is a Foreign Key referencing AlbumID in Album relation and NOT NULL

6. Playlist (PlaylistID, *UserID*, Title, Description, CreationDate)
   - *UserID* is a Foreign Key referencing UserID in User relation and NOT NULL

7. PlaylistSongs (*PlaylistID, SongID*)
   - *PlaylistID* is a Foreign Key referencing PlaylistID in Playlist relation and NOT NULL
   - *SongID* is a Foreign Key referencing SongID in Song relation and NOT NULL

8. ListeningHistory (HistoryID, *UserID*, *SongID*, Timestamp)
   - *UserID* is a Foreign Key referencing UserID in User relation and NOT NULL
   - *SongID* is a Foreign Key referencing SongID in Song relation and NOT NULL

9. Revenue (TransactionID, *UserID,* Source, Date, Amount)
   - *UserID* is a Foreign Key referencing UserID in User relation and NOT NULL

10. BlockchainTransactions (TransactionID, *UserID*, Amount, *ArtistID*, Timestamp)
    - *UserID* is a Foreign Key referencing UserID in User relation and NOT NULL
    - *ArtistID* is a Foreign Key referencing ArtistID in Artist relation and NOT NULL

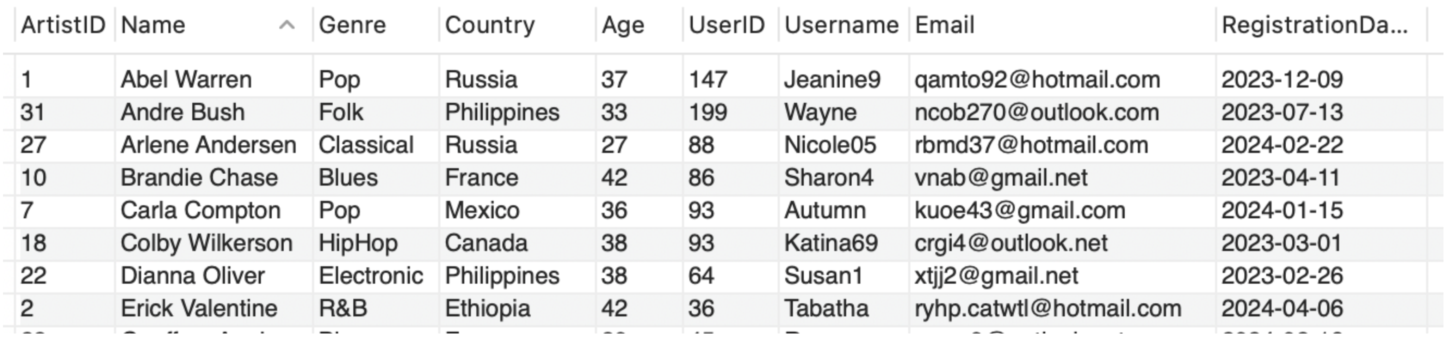
11. ArtistEarnings (*ArtistID*, *TransactionID*, Amount, Timestamp)
    - *ArtistID* is a Foreign Key referencing ArtistID in Artist relation and NOT NULL
    - *TransactionID* is a Foreign Key referencing TransactionID in BlockchainTransactions relation and NOT NULL

Milestone 3: Implementation in MySQL

The database was created in MySQLWorkbench and the following queries were performed:

Query 1: Select all Artists over 25

```
SELECT * FROM Artist
WHERE Age > 25;
```

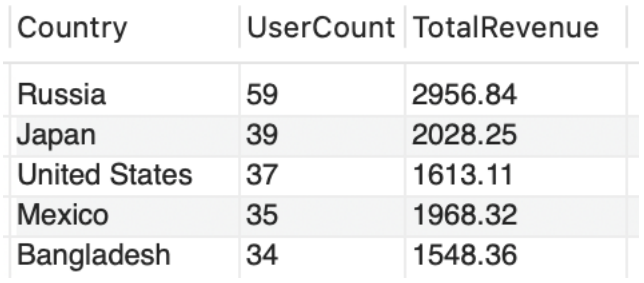


ArtistID	Name	Genre	Country	Age	UserID	Username	Email	RegistrationDa...
1	Abel Warren	Pop	Russia	37	147	Jeanine9	qamto92@hotmail.com	2023-12-09
31	Andre Bush	Folk	Philippines	33	199	Wayne	ncob270@outlook.com	2023-07-13
27	Arlene Andersen	Classical	Russia	27	88	Nicole05	rbmd37@hotmail.com	2024-02-22
10	Brandie Chase	Blues	France	42	86	Sharon4	vnab@gmail.net	2023-04-11
7	Carla Compton	Pop	Mexico	36	93	Autumn	kuoe43@gmail.com	2024-01-15
18	Colby Wilkerson	HipHop	Canada	38	93	Katina69	crgi4@outlook.net	2023-03-01
22	Dianna Oliver	Electronic	Philippines	38	64	Susan1	xtjj2@gmail.net	2023-02-26
2	Erick Valentine	R&B	Ethiopia	42	36	Tabatha	ryhp.catwtl@hotmail.com	2024-04-06

This is a simple query to select all artists over 25 years of age

Query 2: Which 5 countries has the most users and generates the most revenue?

```
SELECT u.Country, COUNT(u.UserID) AS UserCount, SUM(r.Amount) AS TotalRevenue
FROM User u
INNER JOIN Revenue r ON u.UserID = r.UserID
GROUP BY u.Country
ORDER BY UserCount DESC, TotalRevenue DESC
LIMIT 5;
```

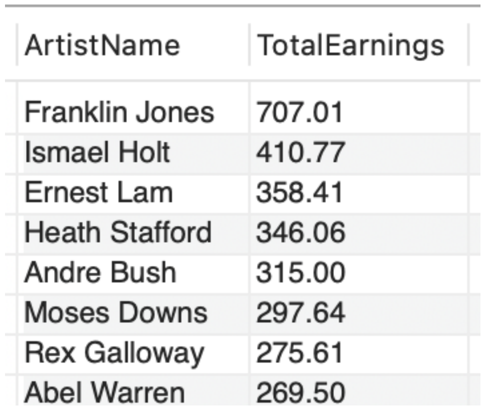


Country	UserCount	TotalRevenue
Russia	59	2956.84
Japan	39	2028.25
United States	37	1613.11
Mexico	35	1968.32
Bangladesh	34	1548.36

This is an Aggregate and Inner Join query to get the count of users and generated revenue by country

Query 3: Identify artists with the highest earnings from blockchain transactions

```
SELECT a.Name AS ArtistName, SUM(ae.Amount) AS TotalEarnings
FROM ArtistEarnings ae
JOIN Artist a ON ae.ArtistID = a.ArtistID
GROUP BY a.ArtistID, a.Name
ORDER BY TotalEarnings DESC;
```

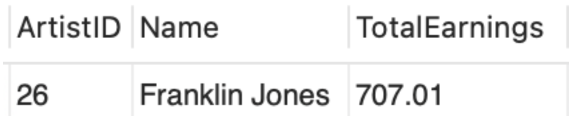


ArtistName	TotalEarnings
Franklin Jones	707.01
Ismael Holt	410.77
Ernest Lam	358.41
Heath Stafford	346.06
Andre Bush	315.00
Moses Downs	297.64
Rex Galloway	275.61
Abel Warren	269.50

This is an Aggregate and Inner Join query to get the artists with the highest earnings

Query 4: Find artist with the highest total earnings

```
WITH ArtistTotalEarnings AS (
   SELECT a.ArtistID, a.Name, SUM(ae.Amount) AS TotalEarnings
   FROM Artist a
   JOIN ArtistEarnings ae ON a.ArtistID = ae.ArtistID
   GROUP BY a.ArtistID, a.Name
)
SELECT ArtistID, Name, TotalEarnings
FROM ArtistTotalEarnings
WHERE TotalEarnings = (SELECT MAX(TotalEarnings) FROM ArtistTotalEarnings);
```

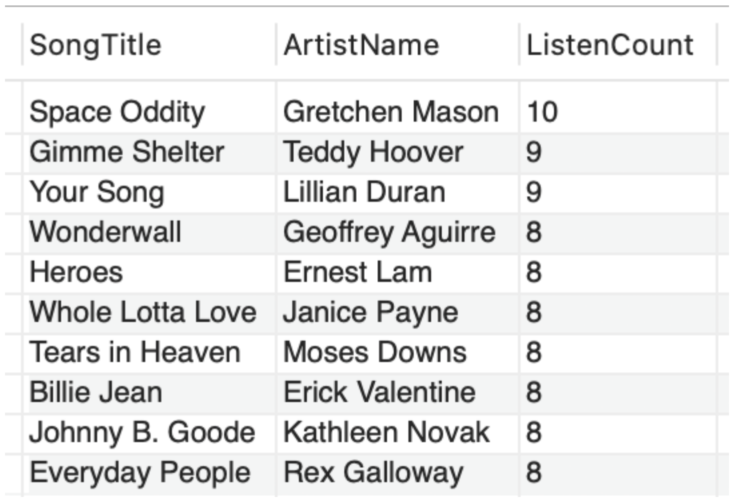


ArtistID	Name	TotalEarnings
26	Franklin Jones	707.01

This is a nested query showing the artist that has the highest earnings

Query 5: List the top 10 most listened-to songs along with their artists

```
SELECT s.Title AS SongTitle,
      a.Name AS ArtistName,
      (SELECT COUNT(*)
       FROM ListeningHistory lh
       WHERE s.SongID = lh.SongID) AS ListenCount
FROM Song s
JOIN Artist a ON s.ArtistID = a.ArtistID
ORDER BY ListenCount DESC
LIMIT 10;
```

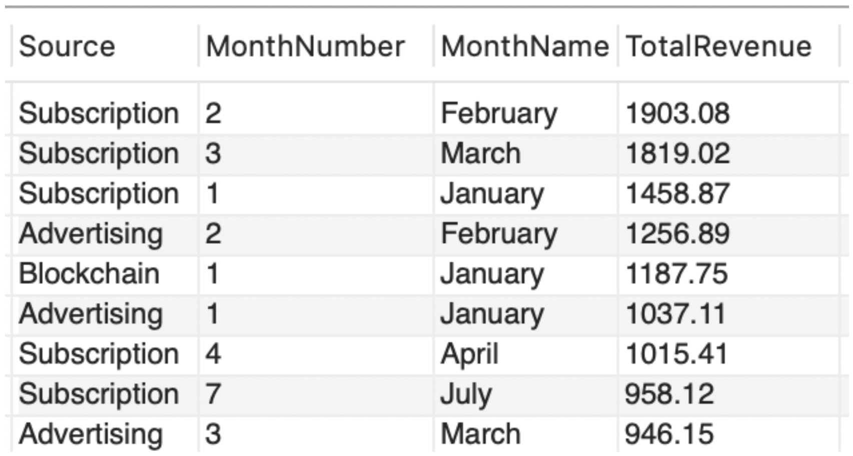


SongTitle	ArtistName	ListenCount
Space Oddity	Gretchen Mason	10
Gimme Shelter	Teddy Hoover	9
Your Song	Lillian Duran	9
Wonderwall	Geoffrey Aguirre	8
Heroes	Ernest Lam	8
Whole Lotta Love	Janice Payne	8
Tears in Heaven	Moses Downs	8
Billie Jean	Erick Valentine	8
Johnny B. Goode	Kathleen Novak	8
Everyday People	Rex Galloway	8

This is a correlated query since the subquery is evaluated for each row returned by the outer query, showing the 10 most listened to songs along with their artists

Query 6: Calculate the total revenue generated each month from each source of revenue

```
SELECT Source, EXTRACT(MONTH FROM Date) AS MonthNumber,
    MONTHNAME(Date) AS MonthName,
    SUM(Amount) AS TotalRevenue
FROM Revenue
WHERE Source IN ('Subscription', 'Advertising', 'Blockchain')
GROUP BY Source, EXTRACT(MONTH FROM Date), MONTHNAME(Date)
ORDER BY TotalRevenue DESC;
```



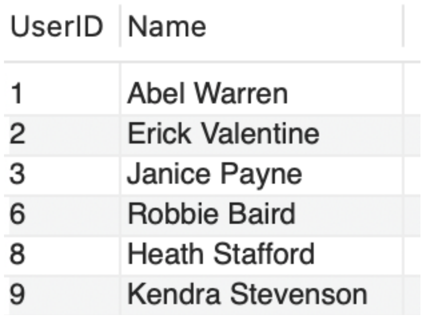
Source	MonthNumber	MonthName	TotalRevenue
Subscription	2	February	1903.08
Subscription	3	March	1819.02
Subscription	1	January	1458.87
Advertising	2	February	1256.89
Blockchain	1	January	1187.75
Advertising	1	January	1037.11
Subscription	4	April	1015.41
Subscription	7	July	958.12
Advertising	3	March	946.15

The query contains subqueries in the **SELECT** clause (**EXTRACT(MONTH FROM Date)** and **MONTHNAME(Date)**), allowing additional computations to be performed on the selected data.

Query 7: Find users who have revenue transactions exceeding the average revenue amount

```
SELECT DISTINCT UserID, Name
```

```
FROM User
WHERE UserID = ANY (
   SELECT UserID
   FROM Revenue
   GROUP BY UserID
   HAVING AVG(Amount) > (SELECT AVG(Amount) FROM Revenue)
);
```

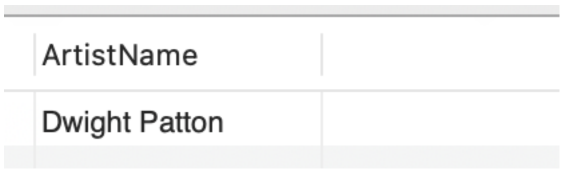


UserID	Name
1	Abel Warren
2	Erick Valentine
3	Janice Payne
6	Robbie Baird
8	Heath Stafford
9	Kendra Stevenson

The query uses the **ANY** clause to compare with the average revenue amount, satisfying the **>ANY** condition.

Query 8: Find artists that have recorded songs but not albums

```
SELECT DISTINCT a.Name AS ArtistName
FROM Artist a
JOIN Song s ON a.ArtistID = s.ArtistID
WHERE NOT EXISTS (
   SELECT 1
   FROM Album al
   WHERE al.ArtistID = a.ArtistID
);
```

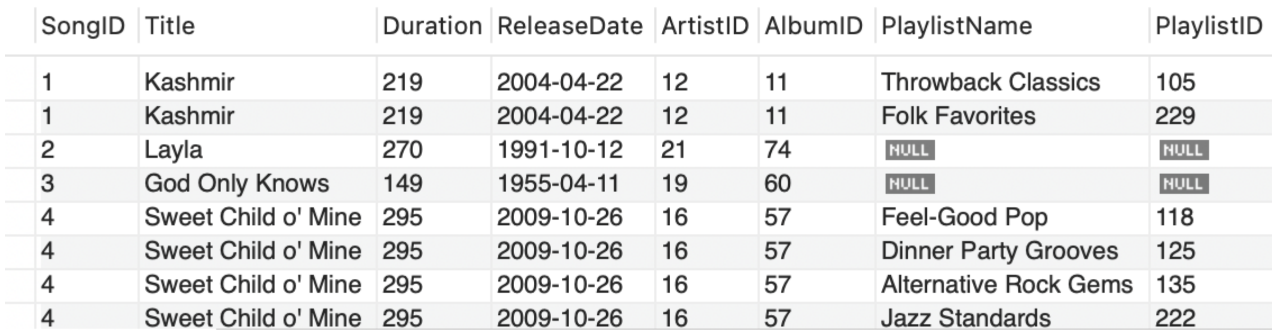


ArtistName
Dwight Patton

The query uses the **NOT EXISTS** clause to find artists that have recorded songs but not albums.

Query 9: Select all songs that have duration of higher than 120 seconds with their along with their corresponding playlist name and playlist ID

```
SELECT s.*, p.Title AS PlaylistName, p.PlaylistID
FROM Song s
LEFT JOIN PlaylistSongs ps ON s.SongID = ps.SongID
LEFT JOIN Playlist p ON ps.PlaylistID = p.PlaylistID
LEFT JOIN Artist a ON s.ArtistID = a.ArtistID
WHERE s.Duration > 120;
```



SongID	Title	Duration	ReleaseDate	ArtistID	AlbumID	PlaylistName	PlaylistID
1	Kashmir	219	2004-04-22	12	11	Throwback Classics	105
1	Kashmir	219	2004-04-22	12	11	Folk Favorites	229
2	Layla	270	1991-10-12	21	74	NULL	NULL
3	God Only Knows	149	1955-04-11	19	60	NULL	NULL
4	Sweet Child o' Mine	295	2009-10-26	16	57	Feel-Good Pop	118
4	Sweet Child o' Mine	295	2009-10-26	16	57	Dinner Party Grooves	125
4	Sweet Child o' Mine	295	2009-10-26	16	57	Alternative Rock Gems	135
4	Sweet Child o' Mine	295	2009-10-26	16	57	Jazz Standards	222

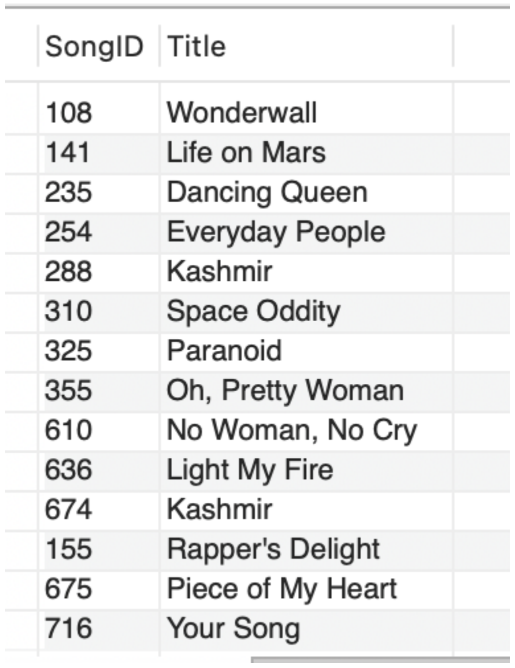
The **LEFT JOIN** operations are used to ensure that all songs are included in the result, even if they are not present in any playlists.

Query 10: To combine songs from the playlist with the most songs and playlist 122

```
-- Songs from the playlist with the most songs
SELECT ps.SongID, s.Title
FROM PlaylistSongs ps
JOIN (
   SELECT PlaylistID
   FROM PlaylistSongs
   GROUP BY PlaylistID
   ORDER BY COUNT(SongID) DESC
   LIMIT 1
) AS max_playlist ON ps.PlaylistID = max_playlist.PlaylistID
JOIN Song s ON ps.SongID = s.SongID

UNION

-- Songs from playlist 122
SELECT s.SongID, s.Title
FROM Song s
JOIN PlaylistSongs ps ON s.SongID = ps.SongID
WHERE ps.PlaylistID = 122;
```

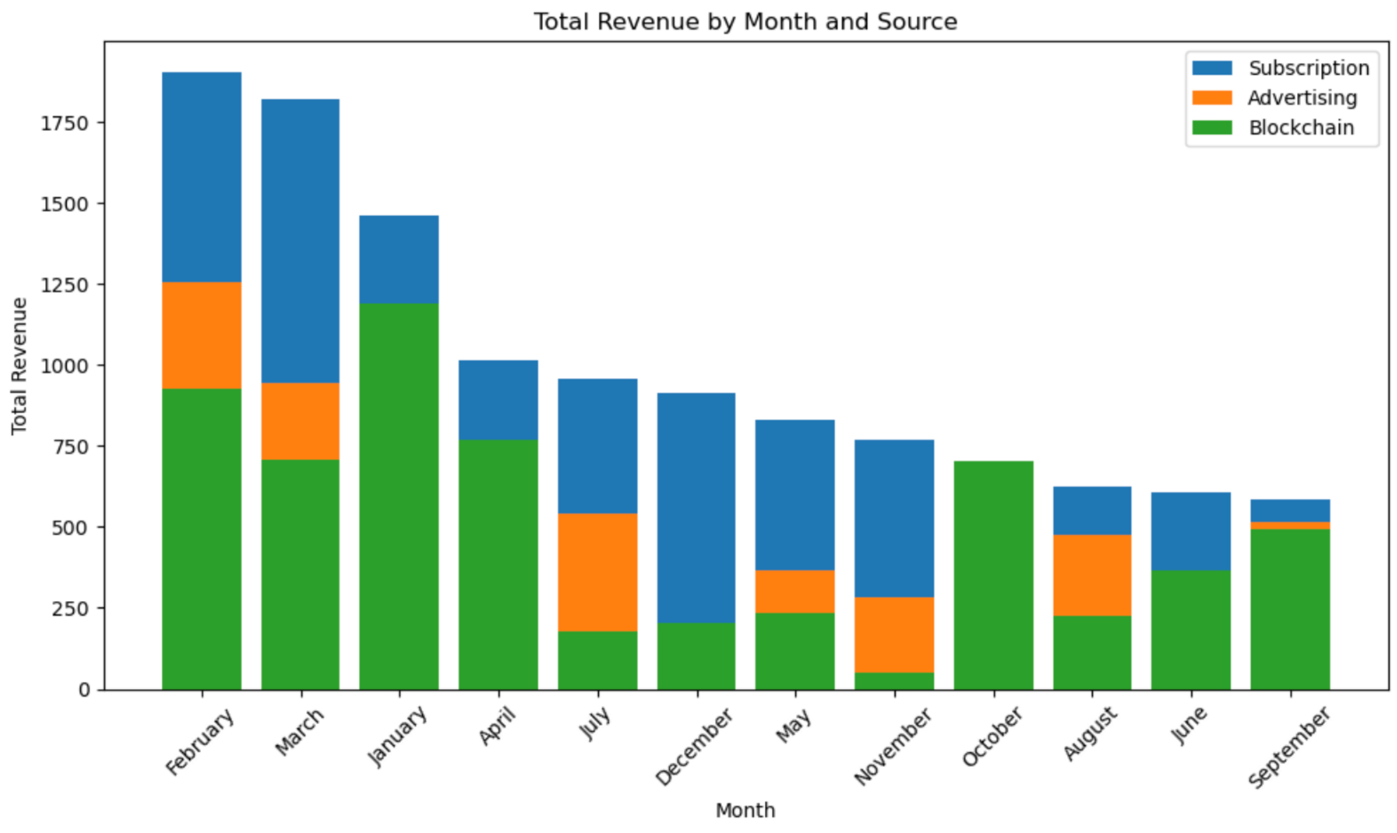


SongID	Title
108	Wonderwall
141	Life on Mars
235	Dancing Queen
254	Everyday People
288	Kashmir
310	Space Oddity
325	Paranoid
355	Oh, Pretty Woman
610	No Woman, No Cry
636	Light My Fire
674	Kashmir
155	Rapper's Delight
675	Piece of My Heart
716	Your Song

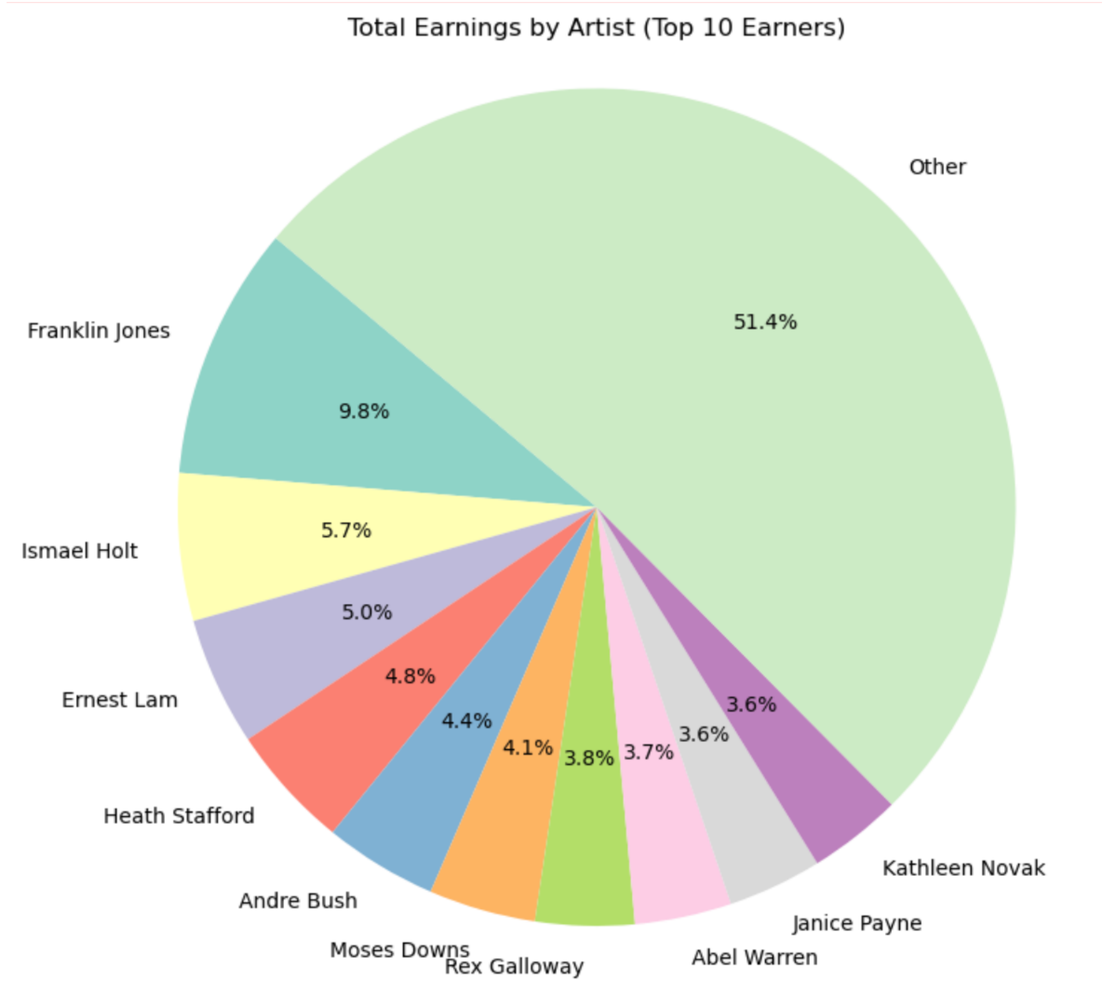
This query retrieves the songs from the playlist with the most songs and combines them with the songs from playlist 122 using the **UNION** set operation.

Milestone 4: Application to Access Database

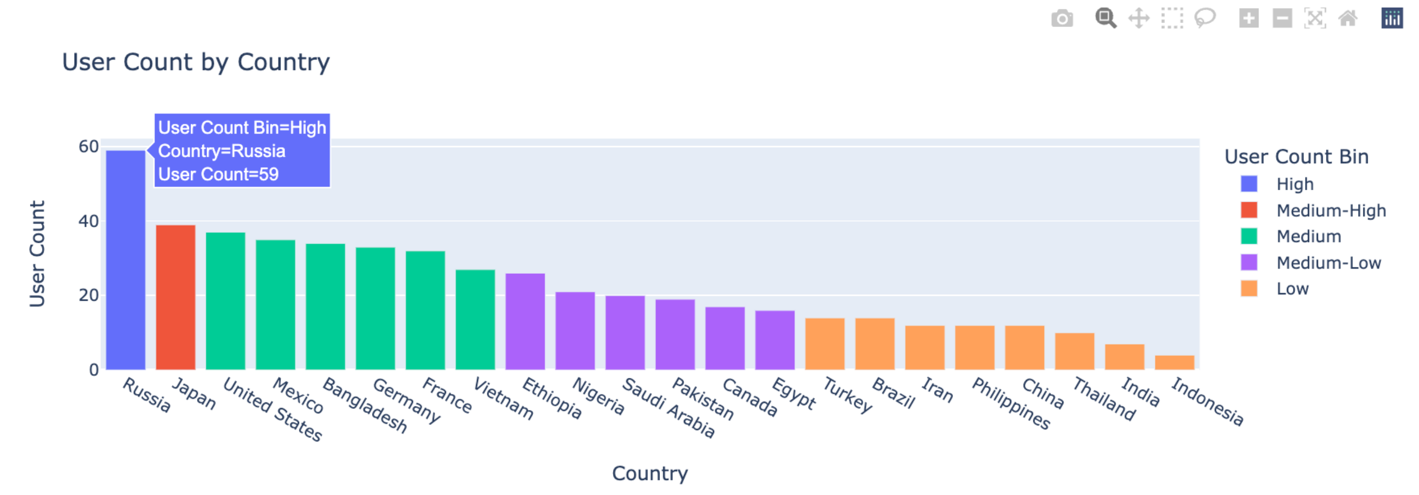
Show the total revenue generated each month from each source of revenue

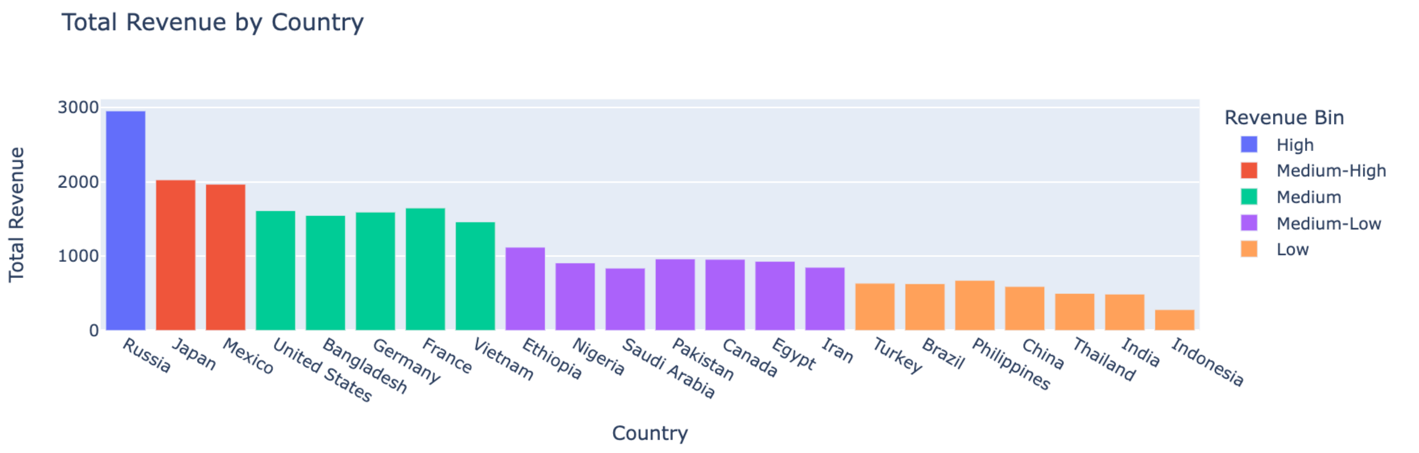


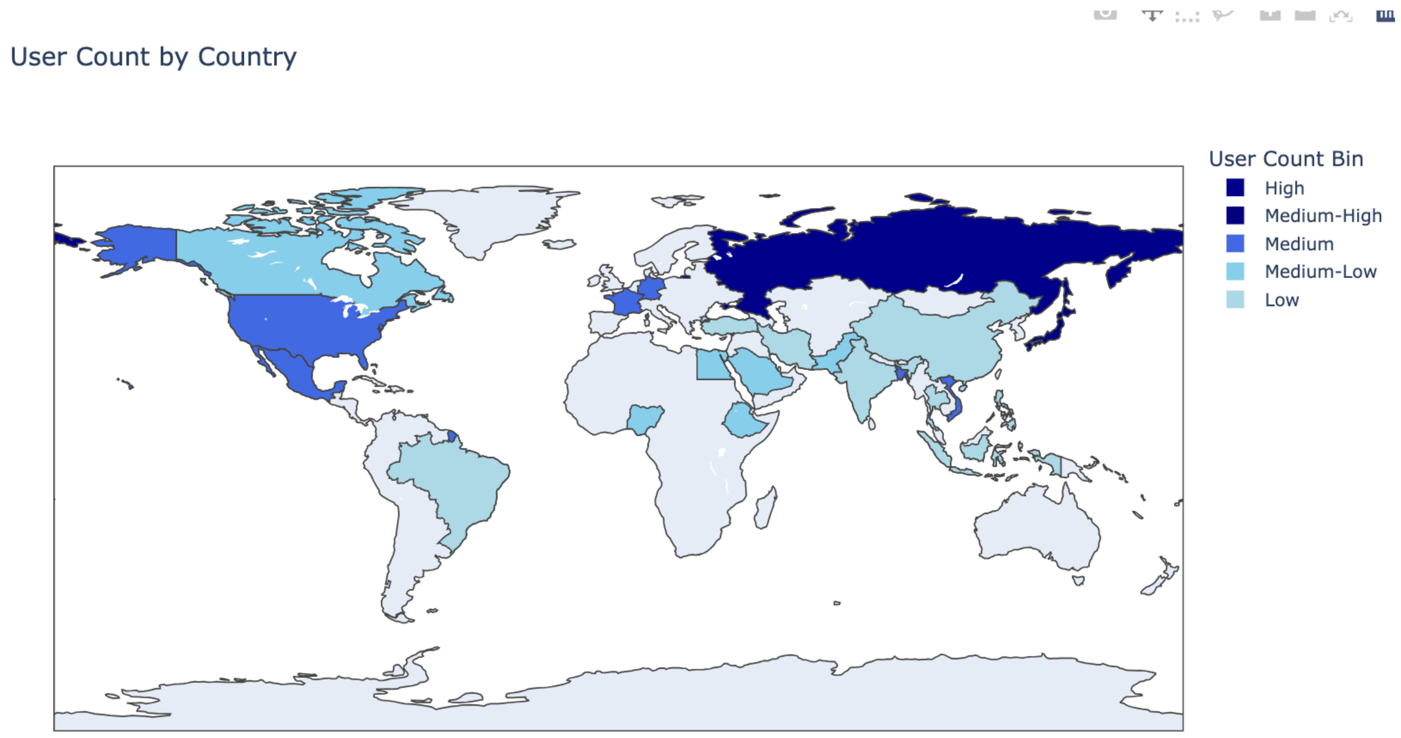
Identify artists with the highest earnings from blockchain transactions

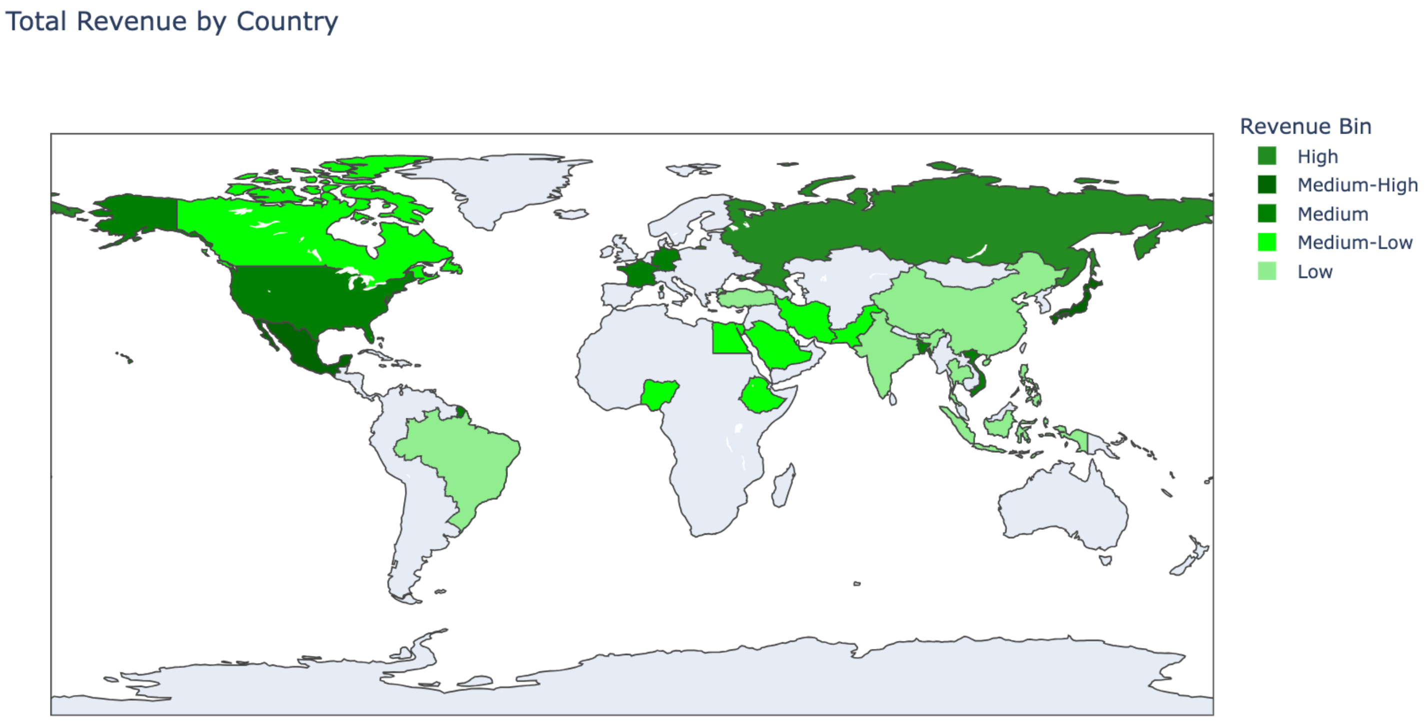


Show countries that have the most users and generates the most revenue





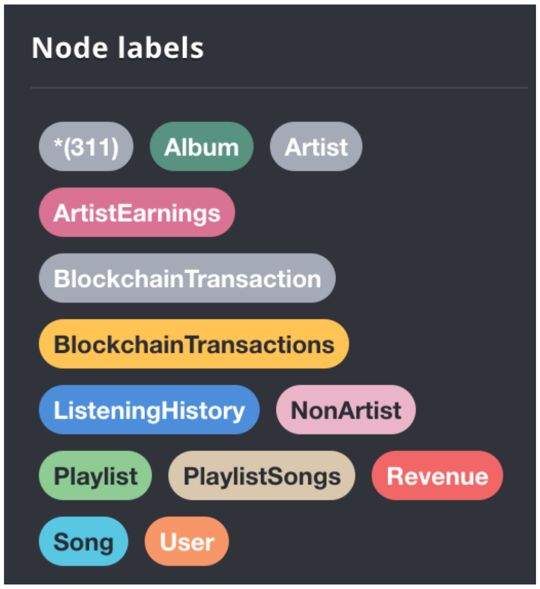



Total Revenue by Country
Revenue Bin
High
Medium-High
Medium
Medium-Low
Low

Milestone 5: Implementation in NoSQL

For my database I used Neo4j for the Implementation in NoSQL. This is because I want to show the relationships between Artists, Songs, Albums, etc. in a Graph structure.

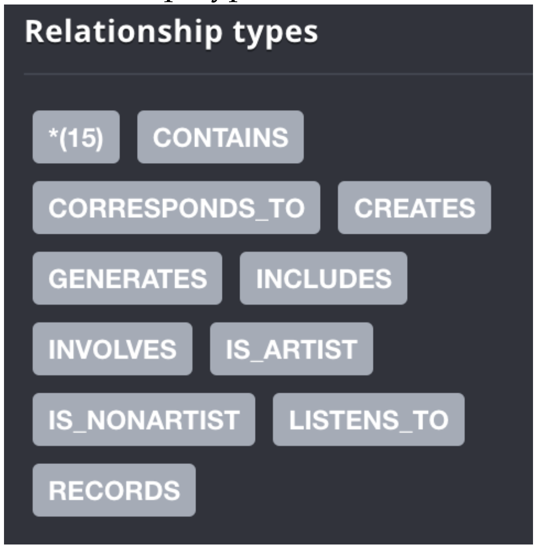
Tables were created for:



Node labels

*(311) Album Artist ArtistEarnings BlockchainTransaction BlockchainTransactions ListeningHistory NonArtist Playlist PlaylistSongs Revenue Song User

Relationship Types created were:



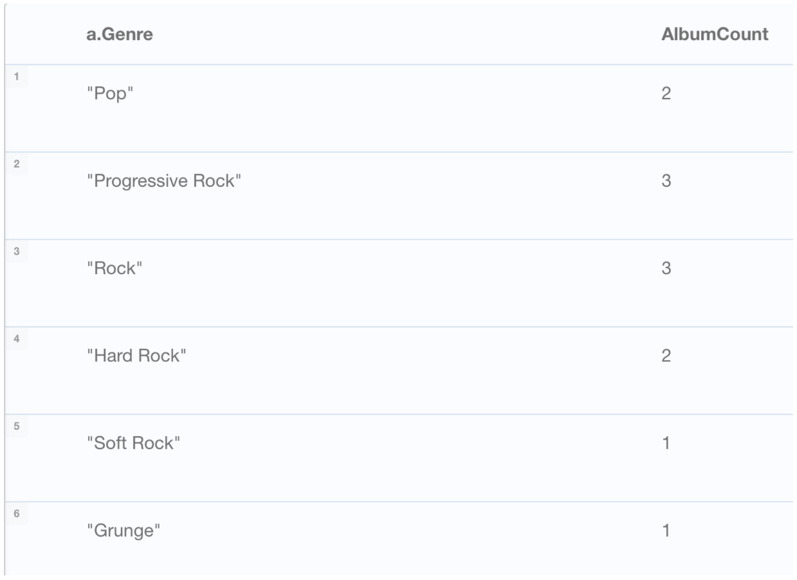
Relationship types

*(15) CONTAINS CORRESPONDS_TO CREATES GENERATES INCLUDES INVOLVES IS_ARTIST IS_NONARTIST LISTENS_TO RECORDS

Now let's see some Cypher queries in action!

Query 1: To find the count of albums for each genre

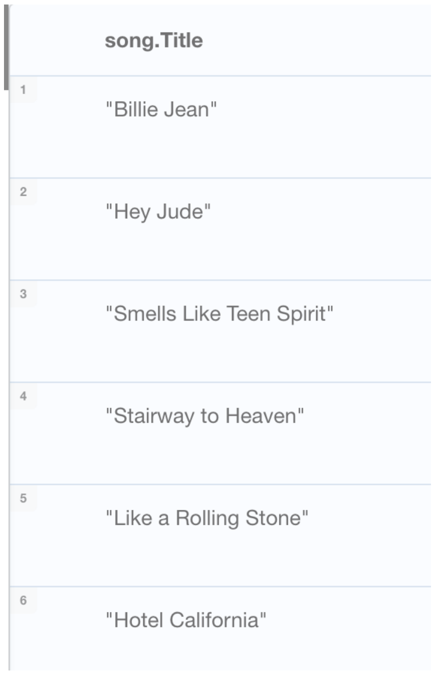
```
MATCH (a:Album)
RETURN a.Genre, COUNT(a) AS AlbumCount
```



	a.Genre	AlbumCount
1	"Pop"	2
2	"Progressive Rock"	3
3	"Rock"	3
4	"Hard Rock"	2
5	"Soft Rock"	1
6	"Grunge"	1

Query 2: show all song titles that have duration above average

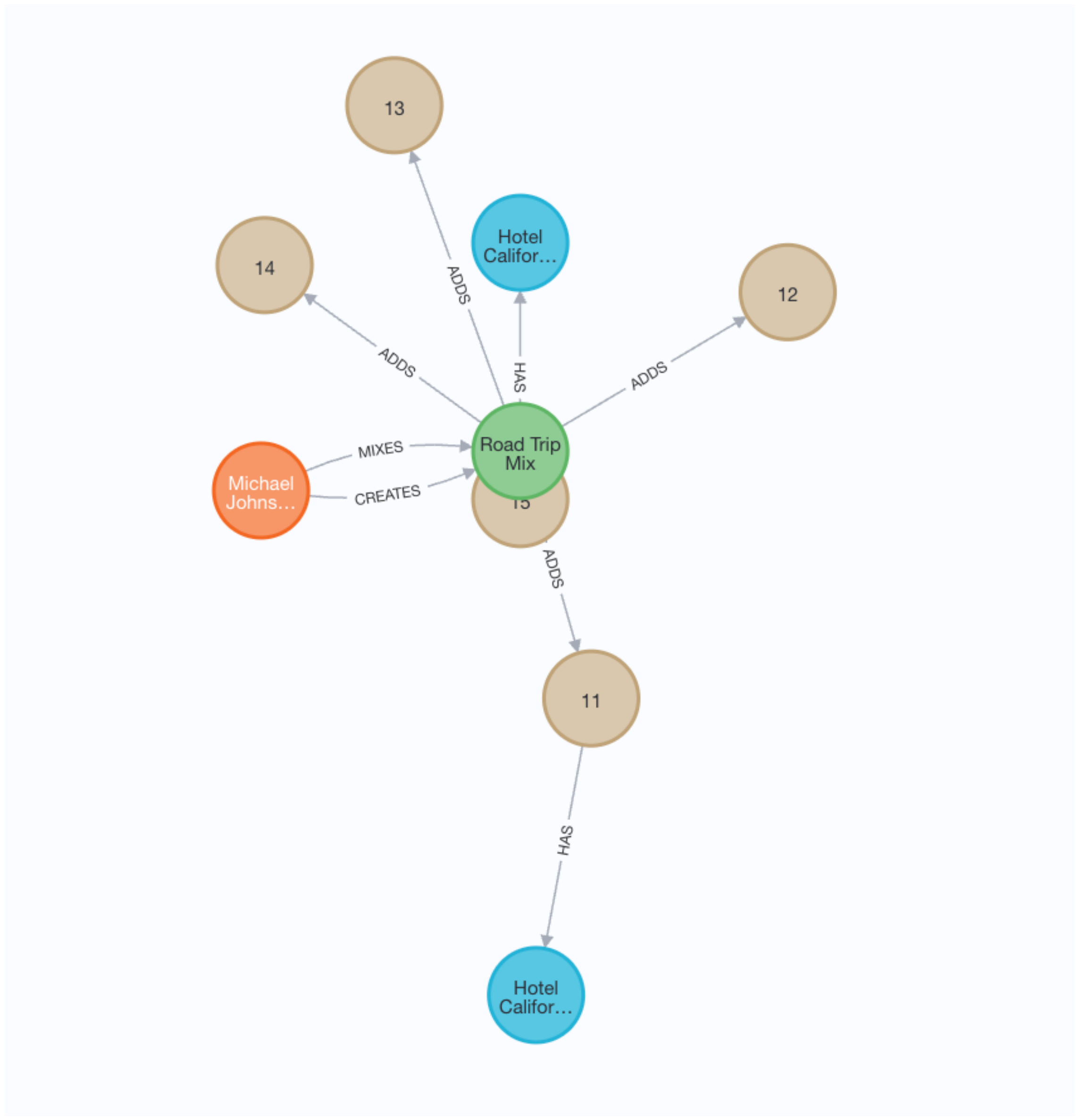
```
MATCH (s:Song)
WITH AVG(s.Duration) AS averageDuration
MATCH (song:Song)
WHERE song.Duration > averageDuration
RETURN song.Title
```



	song.Title
1	"Billie Jean"
2	"Hey Jude"
3	"Smells Like Teen Spirit"
4	"Stairway to Heaven"
5	"Like a Rolling Stone"
6	"Hotel California"

Query 3: this query retrieves all songs with the title 'Hotel California' that are associated with playlists, which in turn are associated with users in the graph database.

MATCH (u:User)--(p:Playlist)--(ps:PlaylistSongs)--(s:Song)
WHERE s.Title = 'Hotel California'
RETURN s



Conclusion and Recommendations

Through meticulous planning and implementation, we have developed a database system that aims to address the complexities and challenges faced by artists, users, and stakeholders in the music industry. However, this was a simple database and can be expanded to optimize the database design and improve the overall effectiveness of the platform.

Recommendations for Improving BlockBeats' Database Design

1. Evaluate the scalability of the database design to accommodate future growth and increased user activity. Consider implementing partitioning strategies, caching mechanisms, and indexing techniques to optimize performance and response times.
2. Enhance data analytics capabilities to derive actionable insights from the database. Invest in advanced analytics tools, machine learning algorithms, and predictive modeling techniques to identify trends, patterns, and opportunities for optimization.
3. Solicit feedback from users, artists, and stakeholders to continuously iterate and improve the database design. Gather user preferences, pain points, and suggestions for enhancement, and incorporate them into future iterations of the database schema and functionality.

By implementing these recommendations, BlockBeats can further optimize its database design to support its mission of revolutionizing the music streaming industry and creating a more equitable ecosystem for artists and users alike.