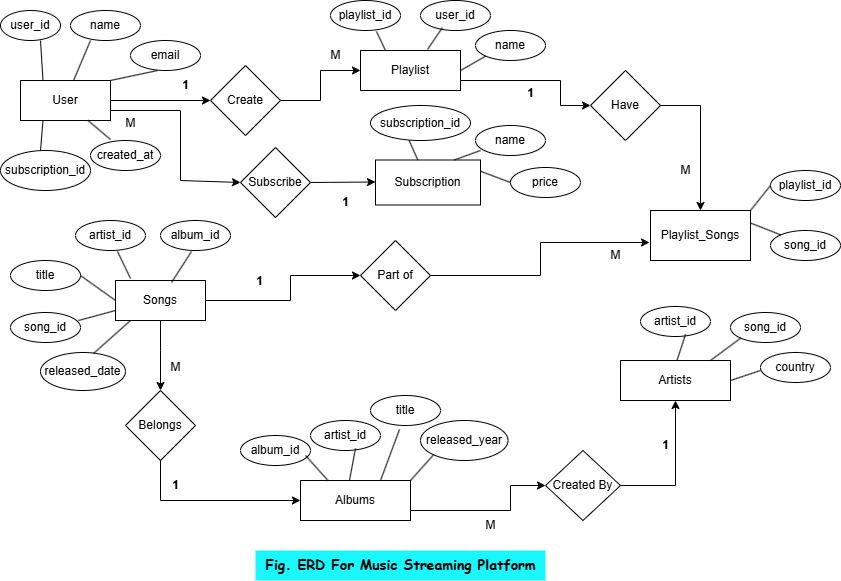
## **Database Assignment on Music Streaming Platform🎵**

* **Step 1: Design an ERD for the assigned topic.**

**Entities:** Users, Songs, Playlists, Subscriptions, Artists, Albums



* **Step 2: Identify Entities, Attributes, Relationships & Cardinality**

**Entities, Attributes:**

 **Users** (user\_id, name, email, password, subscription\_id, created\_at)

 **Songs** (song\_id, title, artist\_id, album\_id, genre, duration, release\_date)

 **Artists** (artist\_id, name, country, debut\_year)

 **Albums** (album\_id, title, artist\_id, release\_year)

 **Playlists** (playlist\_id, user\_id, name, created\_at)

 **Subscriptions** (subscription\_id, name, price, duration)

**Relationships:**

1. A user can have multiple playlists.
2. A playlist can have multiple songs.
3. A song belongs to an album and is created by an artist.
4. Users can subscribe to a plan.
5. Users can like songs and playlists.

* **Step 3: Normalize the Database (1NF, 2NF, 3NF)**

## **Initial Table Structure**

## Before normalization, we might have a table like this:

| **user\_id** | **user\_name** | **email** | **playlist\_id** | **playlist\_name** | **song\_id** | **song\_title** | **artist\_name** | **album\_name** | **Subscription** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Alice | alice@email.com | 101 | Chill Vibes | 201 | Blank Space | Taylor Swift | 1989 | Premium |
| 1 | Alice | alice@email.com | 101 | Chill Vibes | 202 | Shape of You | Ed Sheeran | Divide | Premium |
| 2 | Bob | bob@email.com | 102 | Workout Mix | 203 | Perfect | Ed Sheeran | Divide | Free |

**Problems in Present Table:**

* **Repeating groups**: Multiple songs under the same playlist for a user.
* **Multivalued attributes**: A playlist contains multiple songs, violating atomicity.

1. **First Normal Form (1NF):**

Rules of 1NF

1. Each column must contain atomic (indivisible) values.
2. Each row must have a unique identifier (Primary Key).
3. No repeating groups or arrays.

To achieve 1NF, we separate playlists and songs into different rows instead of grouping them together.

| **user\_id** | **user\_name** | **email** | **playlist\_id** | **playlist\_name** | **song\_id** | **song\_title** | **artist\_name** | **album\_name** | **subscription** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | Alice | alice@email.com | 101 | Chill Vibes | 201 | Blank Space | Taylor Swift | 1989 | Premium |
| 1 | Alice | alice@email.com | 101 | Chill Vibes | 202 | Shape of You | Ed Sheeran | Divide | Premium |
| 2 | Bob | bob@email.com | 102 | Workout Mix | 203 | Perfect | Ed Sheeran | Divide | Free |

**After This :**

* Each field contains only atomic values.
* Each row represents a single piece of information (no grouping of songs).

**Issues still present:**

* **Partial dependencies:** Playlist details depend only on playlist\_id, not on user\_id.
* **Transitive dependencies:** subscription depends on user\_id, which is not directly related to playlists. Hence we try to for the improvement using Second Normal Form.

1. **Second Normal Form (2NF)**

**Rules of 2NF**

1. The table must be in **1NF**.
2. **Remove Partial Dependencies** (A column should depend on the whole primary key, not just a part of it).
3. Create separate tables for independent entities.

Here, Partial Dependencies are exist because of:

* subscription only depends on **user\_id**, not on **playlist\_id** or **song\_id.**
* **artist\_name** and **album\_name** depend on **song\_id**, not on **playlist\_id.**

#### **Users Table Playlists Table**

| **playlist\_id** | **user\_id** | **playlist\_name** |
| --- | --- | --- |
| 101 | 1 | Chill Vibes |
| 102 | 2 | Workout Mix |

| **user\_id** | **user\_name** | **email** | **subscription** |
| --- | --- | --- | --- |
| 1 | Alice | alice@email.com | Premium |
| 2 | Bob | [bob@email.com](mailto:bob@email.com) | Free |

**Songs Table**

| **song\_id** | **song\_title** | **artist\_id** | **album\_id** |
| --- | --- | --- | --- |
| 201 | Blank Space | 1 | 1 |
| 202 | Shape of You | 2 | 2 |
| 203 | Perfect | 2 | 2 |

**Artists Table Albums Table**

| **artist\_id** | **artist\_name** |
| --- | --- |
| 1 | Taylor Swift |
| 2 | Ed Sheeran |

| **album\_id** | **album\_name** |
| --- | --- |
| 1 | 1989 |
| 2 | Divide |

**Playlist-Songs (Bridge Table)**

| **playlist\_id** | **song\_id** |
| --- | --- |
| 101 | 201 |
| 101 | 202 |
| 102 | 203 |

After this know all tables are in 2NF;

* Removed partial dependencies: Playlist and user details are in separate tables.
* Each table focuses on a single entity with relevant attributes.

Still we noticed that **Transitive dependencies** is present in subscription should have its own table since it depends on user\_id indirectly.

1. **Third Normal Form (3NF)**

**Rules of 3NF**

1. The table must be in 2NF.
2. Remove transitive dependencies (A non-key column should not depend on another non-key column).

**Identifying Transitive Dependencies**

* subscription depends on **user\_id**, but **user\_id** should only reference a subscription ID.
* Move subscription details into a separate table.

#### **Users Table Subscriptions Table**

| **subscription\_id** | **subscription\_name** | **price** |
| --- | --- | --- |
| 1 | Premium | 9.99 |
| 2 | Free | 0.00 |

| **user\_id** | **user\_name** | **email** | **subscription\_id** |
| --- | --- | --- | --- |
| 1 | Alice | alice@email.com | 1 |
| 2 | Bob | bob@email.com | 2 |

| **playlist\_id** | **user\_id** | **playlist\_name** |
| --- | --- | --- |
| 101 | 1 | Chill Vibes |
| 102 | 2 | Workout Mix |

**Playlists Table**

**Songs Table**

| **song\_id** | **song\_title** | **artist\_id** | **album\_id** |
| --- | --- | --- | --- |
| 201 | Blank Space | 1 | 1 |
| 202 | Shape of You | 2 | 2 |
| 203 | Perfect | 2 | 2 |

| **album\_id** | **album\_name** |
| --- | --- |
| 1 | 1989 |
| 2 | Divide |

| **artist\_id** | **artist\_name** |
| --- | --- |
| 1 | Taylor Swift |
| 2 | Ed Sheeran |

| **playlist\_id** | **song\_id** |
| --- | --- |
| 101 | 201 |
| 101 | 202 |
| 102 | 203 |

**Artists Table Albums Table Playlist-Songs (Bridge Table)**

Now, in above tables no Transitive Dependencies are Present;

* Better database efficiency and maintainability.
* Each table focuses only on its specific role.
* **Step 4: Implement SQL Database Schema**

**Sample Create Query:**

CREATE TABLE Users (

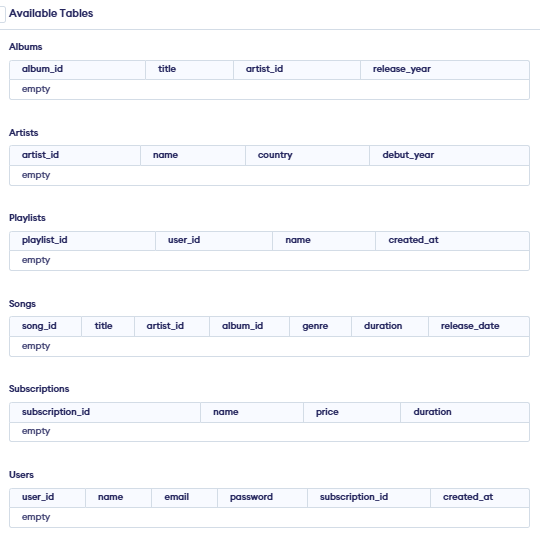
user\_id INT PRIMARY KEY AUTO\_INCREMENT,

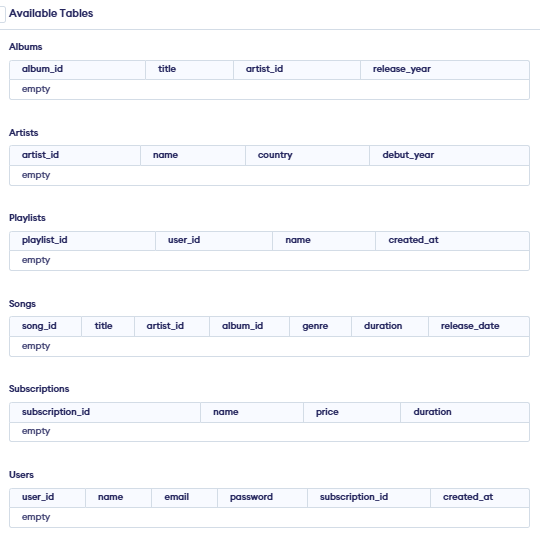
name VARCHAR(100), email VARCHAR(100) UNIQUE,

password VARCHAR(255), subscription\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (subscription\_id) REFERENCES Subscriptions(subscription\_id));

****

****

* **Step 5: Insert Sample Data**

INSERT INTO Albums (album\_id, title)

VALUES (1, 'Aashiqui 2'), (2, 'Ae Dil Hai Mushkil'), (3, 'Tamasha'),

(4, 'Kal Ho Naa Ho'), (5, 'Kabir Singh');

**Albums Table (Sample Table) // For All Tables Refer Folder “Tables”**

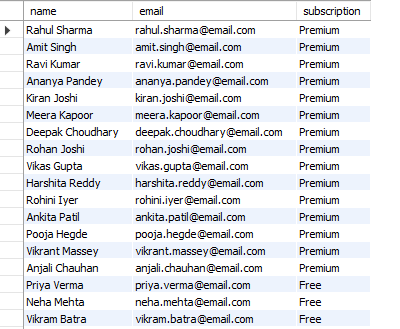


* **Step 6: Write Basic Queries**
* **Get all users and their subscriptions:**

SELECT u.name, u.email, s.name AS subscription

FROM Users u

JOIN Subscriptions s ON u.subscription\_id = s.subscription\_id;



* **Get all songs in a playlist**

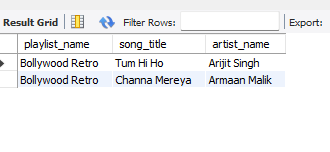
SELECT p.name AS playlist\_name,

s.title AS song\_title, a.name AS artist\_name FROM Playlists p

JOIN Playlist\_Songs ps ON p.playlist\_id = ps.playlist\_id

JOIN Songs s ON ps.song\_id = s.song\_id

JOIN Artists a ON s.artist\_id = a.artist\_id WHERE p.playlist\_id = 101;



* **Get total number of songs per playlist**

SELECT p.name AS playlist\_name, COUNT(ps.song\_id) AS total\_songs

FROM Playlists p JOIN Playlist\_Songs ps ON p.playlist\_id = ps.playlist\_id

GROUP BY p.name;

