**MUSIC SYSTEM-MUSIC BASED MANAGEMENT SYSTEM**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology/Master of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

**Submit**

**T.Abhilash (AP22110010140)**

**B.Roshini(AP22110010141)**

**K.Manas Chowdary (AP22110010164)**

**R. Vignesh(AP22110010176)**

**Vignesh Maddela(AP22110010185)**



**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**04,2024**

# Report Contents:-

1. Identification of Project related to DBMS (project Title)
2. Project Background
3. Description of the Project
4. ER Diagram Creation (use any online tools to draw ER diagram)
5. Description of ER diagram
6. Conversion of ER diagram into Tables
7. Description of Tables
8. Normalization of tables up to 3-NF
9. Creation of Data in the tables (at least 5 tables)
10. Write Sql queries (subqueries, aggregate functions, joins) on the created tables
11. Creation of 5 views using the tables

**PROJECT TITLE:** MusicDB - Music Database Management System

**Project Description:**

**Overview:**

MusicDB is a comprehensive database management system designed to organize and manage music collections efficiently. The system aims to provide users with a user-friendly interface to store, retrieve, and manage music-related data effectively.

**Features:**

**Song Management:**

Allows users to add new songs to the database with details such as title, artist, language, genre, and duration.

Provides functionality to edit and update song information.

Supports deletion of songs from the database.

**Artist Management:**

Enables users to add new artists to the database along with their genre and language information.

Facilitates the editing and updating of artist details.

Allows deletion of artists from the database**.**

**Metadata Management:**

Provides robust metadata management capabilities to ensure accurate organization and categorization of music data.

Supports the addition, modification, and deletion of metadata attributes.

Ensures data integrity and consistency through proper validation and constraints.

**Search Functionality:**

Implements a powerful search feature allowing users to search for songs based on various criteria such as title, artist, language, genre, or duration.

Provides advanced search options including wildcard searches, fuzzy searches, and filtering options.

**User Authentication and Authorization:**

Implements user authentication to ensure secure access to the system.

Enforces role-based access control to restrict unauthorized access to sensitive functionalities.

**User Interface:**

Offers an intuitive and user-friendly interface for easy navigation and interaction.

Supports customizable views and layouts to cater to different user preferences.

**Technology Stack:**

**Database Management System:** MySQL, PostgreSQL, or SQLite

Programming Language: Python (for backend development), HTML/CSS/JavaScript (for frontend development)

Frameworks/Libraries: Flask/Django (for backend development), Bootstrap (for frontend design)

**Implementation:**

**Frontend:** Develop a web-based user interface using HTML, CSS, and JavaScript.

**Backend:** Implement the server-side logic using Python with Flask or Django framework**.**

**Database:** Design and implement the database schema using MySQL, PostgreSQL, or SQLite.

Integrate frontend with backend using RESTful APIs for data exchange.

Benefits:

Efficiently organizes and manages music collections, reducing manual efforts and improving productivity.

Provides quick and easy access to music data through powerful search functionality.

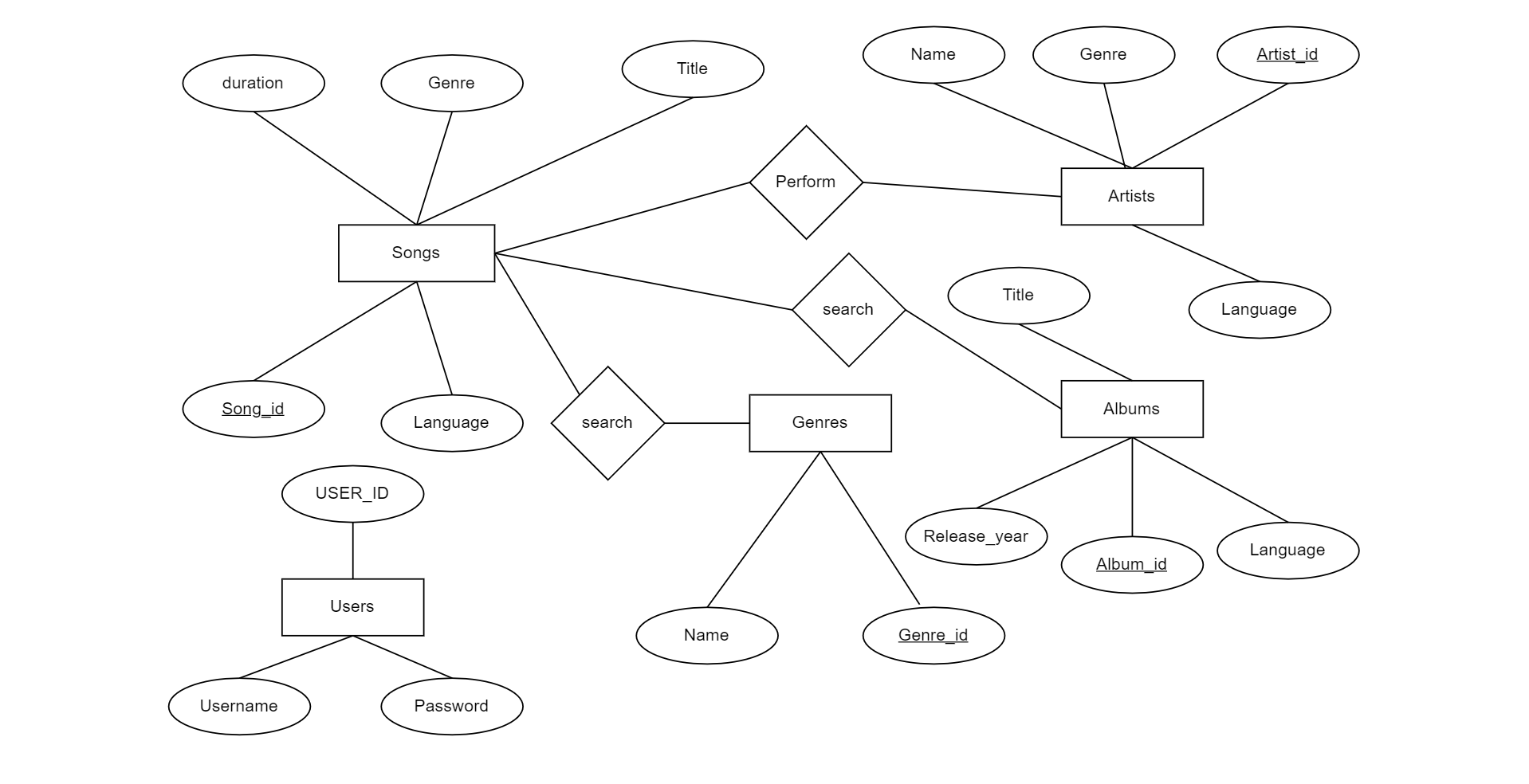
Ensures data integrity and consistency through robust metadata management features.

Enhances user experience with a user-friendly interface and customizable options.

**Verdict :**

MusicDB aims to streamline the management of music collections by offering a comprehensive database management system. With its intuitive interface, powerful search capabilities, and robust metadata management features, MusicDB empowers users to efficiently organize, retrieve, and manage their music data effectively.

**ER Diagram:**



**Description of The ER Diagram:-**

* + 1. An ER diagram is a graphical representation of the entities, attributes, relationships, and cardinality constraints within a database schema.
    2. It provides a visual overview of the data model and helps stakeholders understand the structure and organization of the database.
    3. An Entity Relationship Diagram (ER Diagram) is a visual representation that explains the relationships among entities in a agriculture management system. Let’s break it down simply:

**Entities:**

1. **SONGS:**

**Attributes:**

**Song\_ID (PK):** Unique identifier for each song.

**Title:** Name of the song.

**Language:** Language in which the song is sung.

**Genre:** Genre of the song (e.g., Pop, Rock, Classical).

**Duration:** Length of the song.

**Description:** The Songs entity represents individual songs in the music collection. Each song has a unique Song\_ID and includes details such as title, language, genre, and duration

2**.ARTISTS:**

**Attributes:**

**Artist\_ID (PK):** Unique identifier for each artist.

**Name:** Name of the artist or band.

**Genre:** Genre of the artist's music.

**Language:** Language in which the artist predominantly performs.

**Description:** The Artists entity represents individual artists or bands associated with songs in the music collection. Each artist has a unique Artist\_ID and includes details such as name, genre, and language.

3.**ALBUMS:**

**Attributes:**

**Album\_ID (PK):** Unique identifier for each album.

**Title:** Title of the album.

**Language:** Language in which the album is released.

**Release\_Year:** Year in which the album was released.

**Description:** The Albums entity represents albums containing multiple songs. Each album has a unique Album\_ID and includes details such as title, language, and release year.

4.**ATTRIBUTES:**

**Attributes:**

**Genre\_ID (PK):** Unique identifier for each genre.

**Name:** Name of the genre.

**Description:** The Genres entity represents musical genres used to categorize songs and albums in the music collection. Each genre has a unique Genre\_ID and includes a name describing the genre (e.g., Pop, Rock, Hip-hop).

5.**USERS:**

**Attributes:**

**User\_ID (PK):** Unique identifier for each user.

**Username:** Username used for authentication.

**Password:** Password used for authentication.

**Description:** The Users entity represents users who interact with the MusicDB system. Each user has a unique User\_ID and includes authentication credentials such as username and password.

These entities collectively form the foundation of the MusicDB project, enabling the organization, management, and retrieval of music-related data within the system

**ER DIAGRAMS TO TABLE CONVERSION:**

Songs table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Song\_id | Title | Genre | Duration | Language |
| INT | VARCHAR(225) | VARCHAR(50) | TIME | INT |

Artists Table:

|  |  |  |  |
| --- | --- | --- | --- |
| Artist\_id | Name | Genre | Language |
| INT | VARCHAR(255) | VARCHAR(50) | VARCHAR(50) |

Albums Table:

|  |  |  |  |
| --- | --- | --- | --- |
| Album\_id | Title | Language | Release\_Year |
| INT | VARCHAR(225) | VARCHAR(50) | YEAR |

Genre Table:

|  |  |
| --- | --- |
| Genre\_id | Name |
| INT | Varchar(50) |

Users table:

|  |  |  |
| --- | --- | --- |
| User\_id | Username | Password |
| INT | VARCHAR(225) | VARCHAR(225) |

**Conversion of ER diagram into tables:**

CREATE TABLE Songs (

Song\_id INT PRIMARY KEY,

Title VARCHAR(255) NOT NULL,

Genre VARCHAR(50),

Duration TIME,

Language VARCHAR(50)

);

CREATE TABLE Artists (

Artist\_id INT PRIMARY KEY,

Name VARCHAR(255) NOT NULL,

Genre VARCHAR(50),

Language VARCHAR(50)

);

CREATE TABLE Albums (

Album\_id INT PRIMARY KEY,

Title VARCHAR(255) NOT NULL ,

Language VARCHAR(50),

Release\_Year YEAR

);

CREATE TABLE Genre\_id (

Genre\_id INT PRIMARY KEY,

Name VARCHAR(50),

);

CREATE TABLE Users (

User\_id INT PRIMARY KEY,

Username VARCHAR(255) ,

Password VARCHAR(255),

);

**Description of the Table:**

**1.Songs Table:**

**Description:** The Songs table stores information about individual songs in the music database. Each record represents a single song and includes details such as the song's title, language, genre, duration, and the album it belongs to (if applicable).

**Attributes:**

**Song\_ID:** Unique identifier for each song.

**Title:** Name of the song.

**Language:** Language in which the song is sung.

**Genre:** Genre of the song.

**Duration:** Length of the song.

**Album\_ID:** Foreign key referencing the Album\_ID of the Albums table, indicating the album to which the song belongs

**2.Artists Table:**

**Artists Table:**

**Description:** The Artists table stores information about individual artists or bands associated with songs in the music database. Each record represents a single artist and includes details such as the artist's name, genre, and language of performance.

Attributes:

**Artist\_ID:** Unique identifier for each artist.

**Name:** Name of the artist or band.

**Genre:** Genre of the artist's music.

**Language:** Language in which the artist predominantly performs.

**3.AlbumsTable:**

**Description:** The Albums table stores information about albums containing multiple songs in the music database. Each record represents a single album and includes details such as the album's title, language, and release year.

**Attributes:**

**Album\_ID:** Unique identifier for each album.

**Title:** Title of the album.

**Language:** Language in which the album is released.

**Release\_Year:** Year in which the album was released.

**4.Genres Table:**

**Genres Table:**

**Description:** The Genres table stores information about musical genres used to categorize songs in the music database. Each record represents a single genre and includes details such as the genre's name.

**Attributes:**

**Genre\_ID:** Unique identifier for each genre.

**Name:** Name of the genre.

**5.UsersTable:**

**Description:** The Users table stores information about users who interact with the music database. Each record represents a single user and includes details such as the user's username and password.

**Attributes:**

**User\_ID:** Unique identifier for each user.

**Username:** Username used for authentication.

**Password:** Password used for authentication.

Junction Table for Performs Relationship (Performs):

**Description:** The Performs table represents the many-to-many relationship between songs and artists, indicating which artists perform which songs.

**Attributes:**

Song\_ID: Foreign key referencing the Song\_ID of the Songs table.

Artist\_ID: Foreign key referencing the Artist\_ID of the Artists table.

**Inserting the Records in Tables:**

* Inserting values into the Songs table

INSERT INTO Songs (Song\_id, Title, Genre, Duration, Language)

VALUES

(1, 'Bohemian Rhapsody', 'Rock', '00:05:55', 1),

(2, 'Shape of You', 'Pop', '00:03:53', 2),

(3, 'Uptown Funk', 'Funk', '00:04:30', 1),

(4, 'Despacito', 'Reggaeton', '00:03:49', 3),

(5, 'Dance Monkey', 'Pop', '00:03:29', 2);

* Inserting values into the Artists table

INSERT INTO Artists (Artist\_id, Name, Genre, Language)

VALUES

(1, 'Queen', 'Rock', 'English'),

(2, 'Ed Sheeran', 'Pop', 'English'),

(3, 'Mark Ronson', 'Funk', 'English'),

(4, 'Luis Fonsi', 'Reggaeton', 'Spanish'),

(5, 'Tones and I', 'Pop', 'English');

* Inserting values into the Albums table

INSERT INTO Albums (Album\_id, Title, Language, Release\_Year)

VALUES

(1, 'A Night at the Opera', 'English', '1975'),

(2, '÷ (Divide)', 'English', '2017'),

(3, 'Uptown Special', 'English', '2015'),

(4, 'Vida', 'Spanish', '2019'),

(5, 'The Kids Are Coming', 'English', '2019');

* Inserting values into Genre table

INSERT INTO Genre (Genre\_id, Name)

VALUES

(1, 'Rock'),

(2, 'Pop'),

(3, 'Funk'),

(4, 'Reggaeton');

* Inserting values into Users table

INSERT INTO Users (User\_id, Username, Password)

VALUES

(1, 'user1', 'password1'),

(2, 'user2', 'password2'),

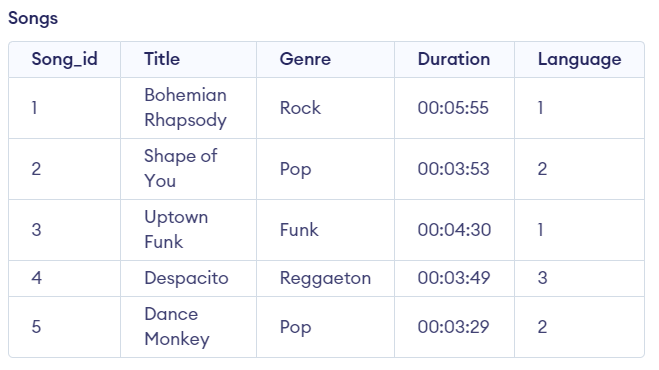
(3, 'user3', 'password3'),

(4, 'user4', 'password4'),

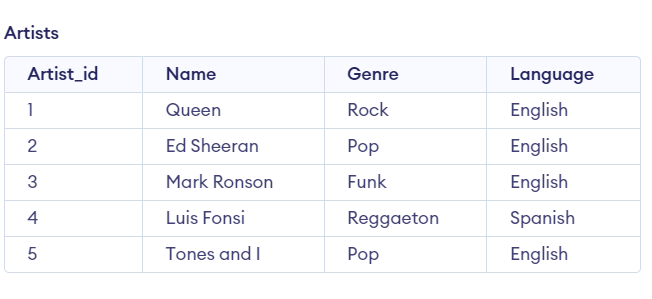
(5, 'user5', 'password5');

**Creation Of Data In Tables:**

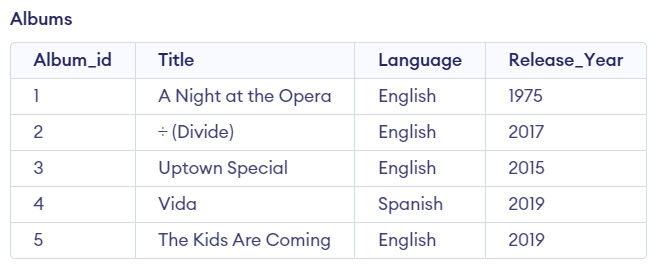
**1)Songs Table:**



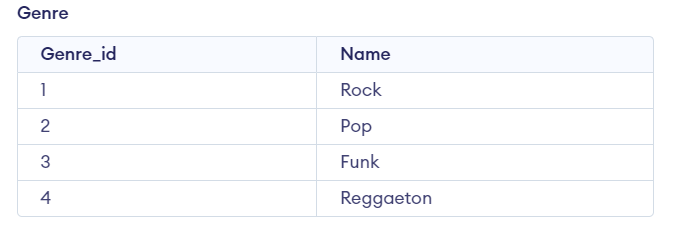
**2)Artists:-**



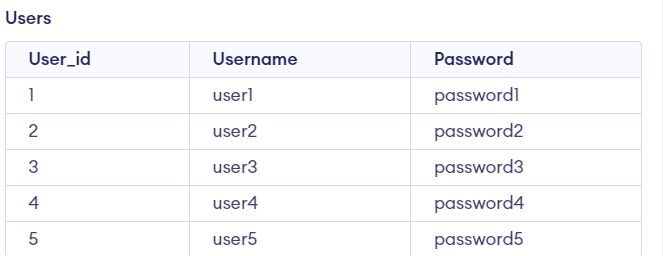
**3)Albums Table:-**



**4)Genre Table:-**



**5)Users Table:-**



**SQL Queries (subqueries, aggregate functions, joins) on the created tables:**

**1) SUB QUERIES:**

**Query 1: List all songs with their genres :**

**SELECT Title,**

**(SELECT Name FROM Genre WHERE Genre\_id = Songs.Genre\_id) AS Genre**

**FROM Songs;**

**Output:**

****

**2) Aggregate Functions:**

**Query 1:** Count the number of songs in each Genre

**SELECT g.Name AS Genre,**

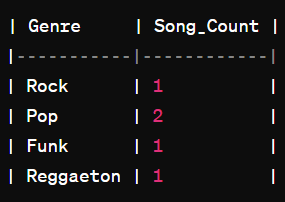
**COUNT(\*) AS Song\_Count**

**FROM Songs s**

**JOIN Genre g ON s.Genre\_id = g.Genre\_id**

**GROUP BY g.Name;**

**Output:-**



**3) JOINS**

**Query 1:** List all albums with their corresponding artists.

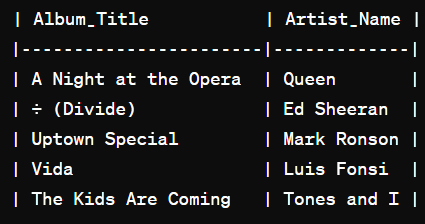
**SELECT a.Title AS Album\_Title,**

**ar.Name AS Artist\_Name**

**FROM Albums a**

**JOIN Artists ar ON a.Artist\_id = ar.Artist\_id;**

**Output:-**

****

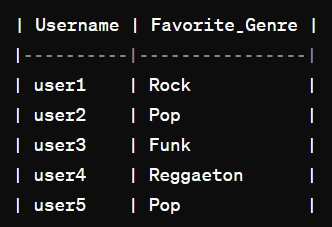
**Query 2:** List all users and their favourite genres:

**SELECT u.Username,**

**(SELECT g.Name FROM Genre g WHERE g.Genre\_id = u.Favorite\_Genre\_id) AS Favorite\_Genre**

**FROM Users u;**

**Output:-**

****

**NORMALIZATION:**

Normalization is the process of organizing data in a database efficiently by reducing redundancy and dependency. It involves breaking down large tables into smaller, more manageable ones and establishing relationships between them to minimize data duplication and ensure data integrity.

THERE are three normal forms

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

Ensure each cell in a table holds a single, atomic value.

**Second Normal Form (2NF):**

Remove partial dependencies by ensuring non-key attributes are fully dependent on the entire primary key.

**Third Normal Form (3NF):**

Eliminate transitive dependencies by ensuring non-key attributes are not dependent on other non-key attributes.

**Normalizing the Created Tables:-**

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

All tables are in 1NF. This means:

* + Each cell contains a single atomic value.
  + There are no repeating groups within a table.
  + All columns have unique names.

Here's the revised schema ensuring 1NF:

Albums Table:

Album\_id (Primary Key)

Title

Release\_Year

Artists Table:

Artist\_id (Primary Key)

Name

Albums\_Artists Table:

Album\_Artist\_id (Primary Key)

Album\_id (Foreign Key)

Artist\_id (Foreign Key)

Genre Table:

Genre\_id (Primary Key)

Name

Language Table:

Language\_id (Primary Key)

Name

Users Table:

User\_id (Primary Key)

Username

Password

Favorite\_Genre\_id (Foreign Key)

Songs Table:

Song\_id (Primary Key)

Title

Album\_id (Foreign Key)

Duration

Language\_id (Foreign Key)

Album\_Songs Table:

Album\_Song\_id (Primary Key)

Album\_id (Foreign Key)

Song\_id (Foreign Key)

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

To satisfy 2NF, the database must first be in 1NF, and all attributes must depend on the whole primary key.

Albums Table:

Album\_id (Primary Key)

Title

Release\_Year

Artists Table:

Artist\_id (Primary Key)

Name

Albums\_Artists Table:

Album\_Artist\_id (Primary Key)

Album\_id (Foreign Key)

Artist\_id (Foreign Key)

Genre Table:

Genre\_id (Primary Key)

Name

Language Table:

Language\_id (Primary Key)

Name

Users Table:

User\_id (Primary Key)

Username

Password

Songs Table:

Song\_id (Primary Key)

Title

Duration

Album\_Songs Table:

Album\_Song\_id (Primary Key)

Album\_id (Foreign Key)

Song\_id (Foreign Key)

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

We need to analyze potential partial and transitive dependencies to determine if the tables are in 3NF.

To achieve 3NF, the database must first be in 2NF, and all attributes must be dependent only on the primary key or other candidate keys, and not on non-key attributes.

Users Table:

User\_id (Primary Key)

Username

Password

Genre Table:

Genre\_id (Primary Key)

Name

Language Table:

Language\_id (Primary Key)

Name

Albums Table:

Album\_id (Primary Key)

Title

Release\_Year

Artists Table:

Artist\_id (Primary Key)

Name

Albums\_Artists Table:

Album\_Artist\_id (Primary Key)

Album\_id (Foreign Key)

Artist\_id (Foreign Key)

Songs Table:

Song\_id (Primary Key)

Title

Duration

Album\_Songs Table:

Album\_Song\_id (Primary Key)

Album\_id (Foreign Key)

Song\_id (Foreign Key)

User\_Favorite\_Genre Table:

User\_Favorite\_Genre\_id (Primary Key)

User\_id (Foreign Key)

Genre\_id (Foreign Key)

**Creation Of Views On Tables:-**

**1)Songs with Album and Artist Details:**

* This view will provide details about each song along with its associated album and artist information.

CREATE VIEW Song\_Details AS

SELECT s.Song\_id, s.Title AS Song\_Title, s.Duration,

a.Name AS Artist\_Name,

al.Title AS Album\_Title, al.Release\_Year

FROM Songs s

JOIN Albums al ON s.Album\_id = al.Album\_id

JOIN Album\_Artists aa ON al.Album\_id = aa.Album\_id

JOIN Artists a ON aa.Artist\_id = a.Artist\_id;

**2)Albums with Artist details:-**

* This view will provide details about each album along with its associated artist information.

CREATE VIEW Album\_Details AS

SELECT al.Album\_id, al.Title, al.Release\_Year,

a.Name AS Artist\_Name

FROM Albums al

JOIN Album\_Artists aa ON al.Album\_id = aa.Album\_id

JOIN Artists a ON aa.Artist\_id = a.Artist\_id;

1. **User’s favourite Genre with Genre details:**

* This view will display the favorite genre of each user along with its details.

CREATE VIEW User\_Favorite\_Genre\_Details AS

SELECT u.User\_id, u.Username,

g.Name AS Favorite\_Genre

FROM Users u

JOIN Genre g ON u.Favorite\_Genre\_id = g.Genre\_id;

1. **Albums with Genre details:**
   * This view presents a summary of farming activities performed by farmers.
   * Users can easily track farming activities without needing to delve into the detailed FarmingActivities table.

CREATE VIEW Album\_Genre\_Details AS

SELECT al.Album\_id, al.Title, al.Release\_Year,

g.Name AS Genre

FROM Albums al

JOIN Genre g ON al.Genre\_id = g.Genre\_id;

### **5)** **Songs with Language Details**

* + This view provides insights into harvested crops along with associated farmer and land details.

CREATE VIEW HarvestView AS

SELECT h.FarmerID, ld.LandID, f.Name AS FarmerName, c.Name AS CropName, Quantity, DATE\_FORMAT(h.Date, '%Y-%m-%d') AS Date, Notes

FROM Harvest h

INNER JOIN LandDetails ld ON h.LandID = ld.LandID

INNER JOIN Farmers f ON h.FarmerID = f.FarmerID

INNER JOIN Crops c ON ld.CropID = c.CropID;

# Conclusion: -

Our Music management system leverages the power of a robust database management system (DBMS) to streamline and optimize farming operations. With a focus on efficient data storage, retrieval, and analysis, our system empowers farmers to make informed decisions, maximize productivity, and ensure sustainability. By integrating DBMS concepts such as entity-relationship modeling, normalization, views, subqueries, aggregate functions, and joins, we provide a user-friendly platform that facilitates effective management and decision-making in Music System.