

Project Report

Music Library Management System

Project submitted to the SRM University - AP, Andhra Pradesh for partial fulfillment of the requirements to award the degree of Bachelors of Technology in Computer Science and Engineering, School of Engineering and Sciences.



Project submitted by,

Manaswi.K	(AP22110010007)
Swathi.K	(AP22110010029)
Sudhamai.P	(AP22110010031)
Sreenedhi.V	(AP22110010039)
Madhavi.K	(AP22110010042)

Under the Guidance of
Mrs. Kavitha Rani Karnena.

Certificate

Date: 24-April-2024

This is to certify that the work present in this Project entitled “**Music Library Management System**” has been carried out by manaswi,swathi,sudhamai,sreenedhi,madhavi under my/our supervision.

The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Science**.

Supervisor

(Signature)

Mrs. Kavitha Rani Karnena

Project Background:

The Music Library Management System aims to provide a user-friendly platform for organizing and managing music collections. It involves creating a database system to efficiently store and manage user accounts, songs, artists, albums, and playlists.

Description of the Project:

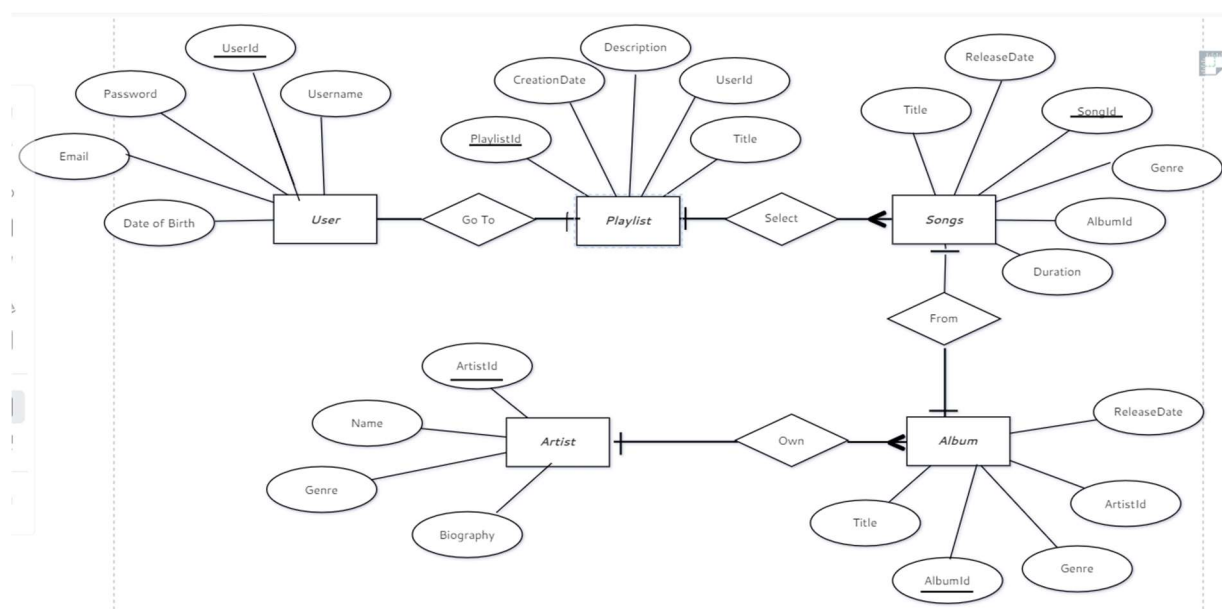
The project involves designing and implementing a database system for a music library. This includes creating an Entity-Relationship (ER) diagram, converting it into relational tables, normalizing the tables up to 3rd Normal Form (3-NF), populating the tables with sample data, writing SQL queries to retrieve information, and creating views for customized data perspectives.

Creating a database system to store, retrieve, update, and delete student records.

Steps:

1. **ER Diagram Creation:** Visualizing entities, relationships, and attributes.
2. **Table Conversion:** Translating the ER diagram into relational tables.
3. **Normalization:** Ensuring tables adhere to 3rd Normal Form.
4. **Data Population:** Adding sample data to at least 5 tables.
5. **SQL Queries:** Crafting queries for data retrieval and manipulation.
6. **View Creation:** Generating 5 views for customized data perspectives.

ER Diagram Creation:



Description of ER diagram:

The ER diagram will include entities such as User, Song, Artist, Album, and Playlist, along with their attributes and relationships. Each entity will be properly linked based on the relationships between them, such as the one-to-many relationship between User and Playlist.

Entities & Attributes:

1. **Song:** Represents a musical composition typically consisting of lyrics and melody.

- Song Id: Unique identifier for each song.
- Title: The title of the song.
- Duration: It is defined as the length of the song in seconds.
- ReleaseDate: It is a date when the song was or will released.
- Genre: The genre(s) of the song.
- Album ID: A foreign key that links the song to its corresponding album.

2. **Artist:** Represents a musician, band, or performer who creates music.

- Artist ID: Unique identifier for each artist.
- Name: It is an name of the artist or band.
- Biography: Information about the artist's background or history.
- Genre: The genre(s) associated with the artist's music.

3. **Album:** Represents a collection of songs released together by an artist or band.

- Album ID: Unique identifier for each album.
- Title: The title of the album.
- ReleaseDate: The date when the album was released.
- Genre: The genre(s) of the album.
- Artist ID: A foreign key that links the album to its corresponding artist.

4. **Playlist:** Represents a curated list of songs.

- Playlist ID: Unique identifier for each playlist.
- Title: The title or name of the playlist.
- Description: A brief description or summary of the playlist.
- CreationDate: The date when the playlist was created.
- User ID: A foreign key that links the playlist to its creator (user).

5. **Users:** Represents a person who uses the music platform.

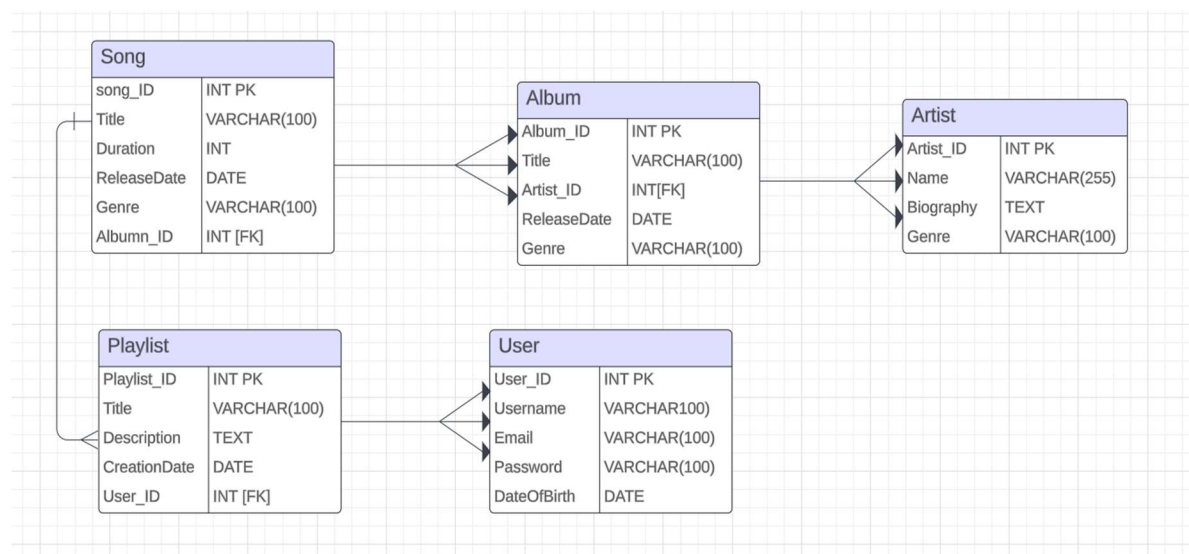
- User ID: Unique identifier for each user.

- Username: The username chosen by the user for their account.
- Email: It is an email address which is attached with the user's account
- The password used for user authentication.
- Date of Birth: The user's date of birth for age verification or personalization.

Relationship Between these Entites:-

- **Song – Album Relationship: One-to-many** relationship where each song belongs to one album, but an album can have multiple songs. Song table contains a foreign key (AlbumID) referencing the AlbumID in the Album table.
- **Album – Artist Relationship: One-to-many** relationship where each album is created by one artist, but an artist can have multiple albums. Album table contains a foreign key (ArtistID) referencing the ArtistID in the Artist table.
- **Playlist – Song Relationship: Many-to-many** relationship where each playlist can have multiple songs, and a song can be in multiple playlists. Implemented using an intermediate table (e.g., PlaylistSong) containing foreign keys (PlaylistID and SongID).
- **Users – Playlist Relationship: One-to-many** relationship where each user can create multiple playlists, but a playlist is created by one user. Playlist table contains a foreign key (UserID) referencing the UserID in the Users table.

Conversion of ER diagram into Tables :



Description of Tables :-

Song:

- **Song_ID:** Primary key, integer, uniquely identifies each song.
- **Title:** Variable character field, stores the title of the song.
- **Duration:** Integer, represents the duration of the song in seconds.
- **Release Date:** Date field, indicates the date when the song was released.
- **Genre:** Variable character field, specifies the genre of the song.
- **Album_ID:** Foreign key referencing the Album table, indicating which album the song belongs to.

```
CREATE TABLE Song (  
    Song_ID INT PRIMARY KEY,  
    Title VARCHAR(100),  
    Duration INT,  
    ReleaseDate DATE,  
    Genre VARCHAR(100),  
    Album_ID INT,  
    FOREIGN KEY (Album_ID) REFERENCES Album(Album_ID)  
);
```

Artist:

- **Artist_ID:** Primary key, integer, uniquely identifies each artist.
- **Name:** Variable character field, stores the name of the artist.
- **Biography:** Text field, contains a biography or description of the artist.
- **Genre:** Variable character field, specifies the genre(s) associated with the artist.

```
CREATE TABLE Artist (  
    Artist_ID INT,  
    Name VARCHAR(255),  
    Biography TEXT,  
    Genre VARCHAR(100)  
);
```

Album:

- **Album_ID:** Primary key, integer, uniquely identifies each album.
- **Title:** Variable character field, stores the title of the album.
- **Release Date:** Date field, indicates the date when the album was released.
- **Genre:** Variable character field, specifies the genre of the album.
- **Artist_ID:** Foreign key referencing the Artist table, indicates which artist or band released the album.

```
CREATE TABLE Album (  
  Album_ID INT PRIMARY KEY,  
  Title VARCHAR(100),  
  Artist_ID INT,  
  ReleaseDate DATE,  
  Genre VARCHAR(100),  
  FOREIGN KEY (Artist_ID) REFERENCES Artist(Artist_ID) );
```

Playlist:

- **Playlist_ID:** Primary key, integer, uniquely identifies each playlist.
- **Title:** Variable character field, stores the title of the playlist.
- **Description:** Text field, contains a description or summary of the playlist.
- **Creation Date:** Date field, indicates when the playlist was created.
- **User_ID:** Foreign key referencing the User table, indicates the user who created the playlist.

```
CREATE TABLE Playlist (  
  Playlist_ID INT PRIMARY KEY,  
  Title VARCHAR(100),  
  Description TEXT,  
  CreationDate DATE,  
  User_ID VARCHAR(100),  
  FOREIGN KEY (Title) REFERENCES Song(Title) );
```

Users:

- **User_ID:** Primary key, integer, uniquely identifies each user.
- **Username:** Variable character field, stores the username of the user.
- **Password:** Variable character field, stores the password of the user.
- **Email:** Variable character field, stores the email address of the user.

- **Date of Birth:** Date field, indicates the date of birth of the user.

```
CREATE TABLE Users (  
  
    User_ID INT PK,  
  
    Username VARCHAR(100),  
  
    Email VARCHAR(100),  
  
    Password VARCHAR(100),  
  
    DateOfBirth DATE  
  
);
```

Normalization of tables up to 3-NF:

To normalize the tables up to the third normal form (3-NF) in the Music Library Management System, we need to ensure that:

1. Each table has a unique primary key.
2. Each non-key attribute is fully functionally dependent on the primary key
3. There are no transitive dependencies.

Given the entities and their attributes in the system, let's go through the normalization process:

Song:

- **Song ID (pk)**
- **Title**
- **Duration**
- **ReleaseDate**
- **Genre**
- **Album ID (FK)**

Artist:

- **Artist ID (PK)**
- **Name**
- **Biography**
- **Genre**

Album:

- **Album ID (PK)**
- **Title**
- **ReleaseDate**
- **Genre**
- **Artist ID (FK)**

Playlist:

- **Playlist ID (PK)**
- **Title**
- **Description**
- **Creation Date**
- **User ID (FK)**

Users:

- **User ID (PK)**
- **Username**
- **Password**
- **Email**
- **DateOfBirth**

First Normal Form (1NF):

- All tables already satisfy 1NF since each attribute contains atomic values, and there are no repeating groups.

Second Normal Form (2NF):

- No partial dependencies exist.
- All non-key attributes are fully functionally dependent on the primary key.

Third Normal Form (3NF):

- No transitive dependencies exist.
- Now, let's analyze the music library database tables as a whole:

All tables in the music library database are in the third normal form (3NF). Here's why:

Atomicity: Each attribute in all tables contains atomic values. For example, attributes like Title, Duration, Release Date, Genre, Name, Biography, Username, Password, Email, Date of Birth, etc., are all atomic and cannot be further divided.

No Repeating Groups: There are no repeating groups within any table. Each record in a table represents a unique entity, and there are no repeating sets of attributes.

No Partial Dependencies: No partial dependencies exist in any table. All non-key attributes are fully functionally dependent on the primary key. For example, in the Song table, attributes like Title, Duration, Release Date, and Genre depend fully on the Song_ID primary key.

No Transitive Dependencies: No transitive dependencies exist in any table. There are no non-key attributes that depend on other non-key attributes, which in turn depend on the primary key.

Creation of Data in the tables :

-- Inserting data into the **Song table**

```
INSERT INTO Song (Song_ID, Title, Duration, ReleaseDate, Genre, Album_ID) VALUES
```

```
(1, 'Song 1', 180, '2023-01-01', 'Pop', 1),
```

```
(2, 'Song 2', 240, '2022-05-20', 'Rock', 2),
```

```
(3, 'Song 3', 200, '2024-03-10', 'Hip Hop', 3);
```

-- Inserting data into the **Artist table**

```
INSERT INTO Artist (Artist_ID, Name, Biography, Genre) VALUES
```

```
(1, 'Artist 1', 'Biography for Artist 1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed eget libero nec turpis posuere facilisis vel id quam. Nulla convallis, magna ut euismod pulvinar, purus dui suscipit lacus, nec volutpat lorem nisl nec est.', 'Pop'),
```

```
(2, 'Artist 2', 'Biography for Artist 2. Ut vel turpis id leo rhoncus fermentum. Proin volutpat velit non felis convallis, id lobortis libero tristique. Duis vitae posuere justo, id molestie nulla. Integer tempor est et risus semper, in aliquet risus volutpat.', 'Rock'),
```

```
(3, 'Artist 3', 'Biography for Artist 3. Fusce fringilla tortor quis felis placerat, nec tempus velit feugiat. Morbi nec mauris non magna aliquam placerat nec in elit. Nulla at augue vestibulum, tincidunt mi a, lacinia sem.', 'Hip Hop');
```

-- Inserting data into **the Album table**

INSERT INTO Album (Album_ID, Title, ReleaseDate, Genre, Artist_ID) VALUES

(1, 'Album 1', '2023-01-01', 'Pop', 1),

(2, 'Album 2', '2022-05-20', 'Rock', 2),

(3, 'Album 3', '2024-03-10', 'Hip Hop', 3);

-- Inserting data into the **Playlist table**

INSERT INTO Playlist (Playlist_ID, Title, Description, CreationDate, User_ID) VALUES

(1, 'Playlist 1', 'Description for Playlist 1. Sed eu ex non ante convallis tincidunt. Sed auctor, orci eget mattis euismod, justo felis bibendum mauris, in tincidunt nisi lorem non ex. Sed eleifend ultricies nulla, a venenatis justo convallis vel. Praesent ultrices tellus at congue molestie. Duis fringilla lorem non libero bibendum, in consequat eros accumsan.', '2024-04-23', 1),

(2, 'Playlist 2', 'Description for Playlist 2. Vestibulum nec tempus lorem. Sed eleifend auctor ante a consequat. Sed sodales eleifend justo sit amet laoreet. Sed ultricies tortor et elit convallis rhoncus. Morbi nec urna at arcu posuere scelerisque sit amet id leo. Ut ut dictum justo. Integer efficitur, purus sit amet egestas fringilla, arcu nulla commodo odio, eget accumsan orci urna nec lacus.', '2024-04-23', 2),

(3, 'Playlist 3', 'Description for Playlist 3. Morbi fringilla justo non laoreet interdum. Nulla consectetur metus nisi, eget facilisis lectus luctus eget. Duis tempus, felis ut tempus feugiat, nisi risus consequat dui, at malesuada purus orci nec libero. Integer varius commodo arcu ac consequat. Sed ultricies dui vel risus efficitur varius. Suspendisse potenti.', '2024-04-23', 3);

-- Inserting data into the **Users table**

INSERT INTO Users (User_ID, Username, Password, Email, DateOfBirth) VALUES

(1, 'john', 'password1', 'john@example.com', '1990-05-15'),

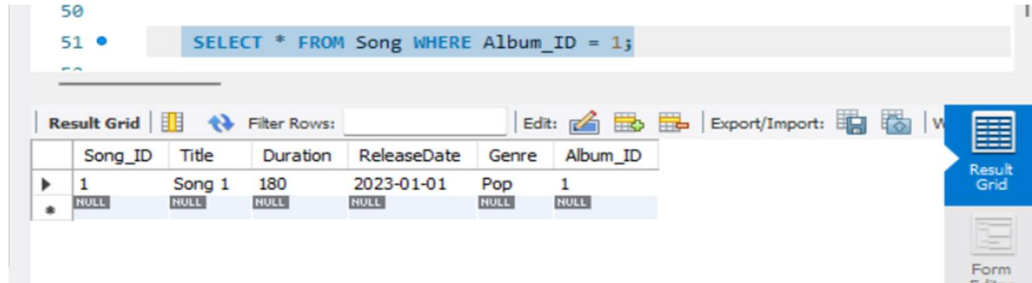
(2, 'dev', 'password2', 'dev@example.com', '1985-10-20'),

(3, 'siva', 'password3', 'siva@example.com', '2000-03-25');

Few sql queries on the created tables :

--Retrieve all songs from a specific album:

```
SELECT * FROM Song WHERE Album_ID = 1;
```

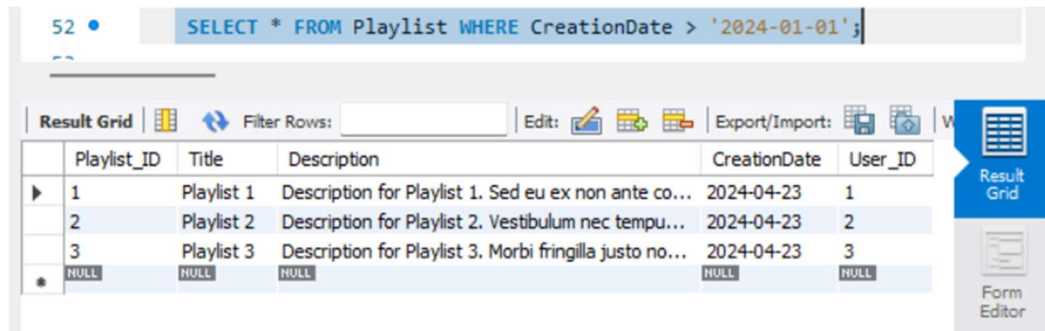


The screenshot shows a SQL query editor with the query `SELECT * FROM Song WHERE Album_ID = 1;` entered. Below the query, the 'Result Grid' is displayed, showing the results of the query. The grid has columns: Song_ID, Title, Duration, ReleaseDate, Genre, and Album_ID. The first row shows Song_ID 1, Title 'Song 1', Duration 180, ReleaseDate '2023-01-01', Genre 'Pop', and Album_ID 1. The second row shows NULL values for all columns.

Song_ID	Title	Duration	ReleaseDate	Genre	Album_ID
1	Song 1	180	2023-01-01	Pop	1
NULL	NULL	NULL	NULL	NULL	NULL

--Retrieve all playlists created after a specific date:

```
SELECT * FROM Playlist WHERE CreationDate > '2024-01-01';
```



The screenshot shows a SQL query editor with the query `SELECT * FROM Playlist WHERE CreationDate > '2024-01-01';` entered. Below the query, the 'Result Grid' is displayed, showing the results of the query. The grid has columns: Playlist_ID, Title, Description, CreationDate, and User_ID. The first three rows show Playlist_ID 1, 2, and 3, each with a title, description, creation date of '2024-04-23', and user ID 1, 2, and 3 respectively. The fourth row shows NULL values for all columns.

Playlist_ID	Title	Description	CreationDate	User_ID
1	Playlist 1	Description for Playlist 1. Sed eu ex non ante co...	2024-04-23	1
2	Playlist 2	Description for Playlist 2. Vestibulum nec tempu...	2024-04-23	2
3	Playlist 3	Description for Playlist 3. Morbi fringilla justo no...	2024-04-23	3
NULL	NULL	NULL	NULL	NULL

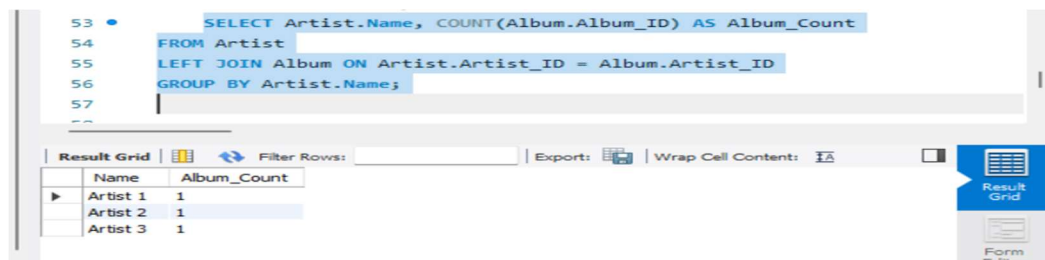
--Retrieve the number of albums released by each artist:

```
SELECT Artist.Name, COUNT(Album.Album_ID) AS Album_Count
```

```
FROM Artist
```

```
LEFT JOIN Album ON Artist.Artist_ID = Album.Artist_ID
```

```
GROUP BY Artist.Name;
```



The screenshot shows a SQL query editor with the query `SELECT Artist.Name, COUNT(Album.Album_ID) AS Album_Count FROM Artist LEFT JOIN Album ON Artist.Artist_ID = Album.Artist_ID GROUP BY Artist.Name;` entered. Below the query, the 'Result Grid' is displayed, showing the results of the query. The grid has columns: Name and Album_Count. The first three rows show Artist 1, Artist 2, and Artist 3, each with an Album_Count of 1.

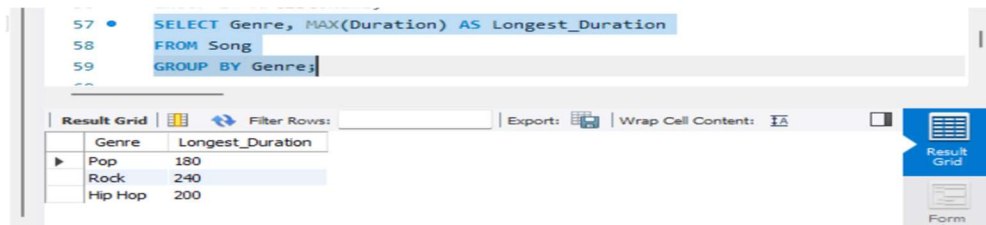
Name	Album_Count
Artist 1	1
Artist 2	1
Artist 3	1

--Retrieve the longest song in each genre:

```
SELECT Genre, MAX(Duration) AS Longest_Duration
```

```
FROM Song
```

```
GROUP BY Genre;
```



The screenshot shows a SQL query editor with the following query:

```
57 • SELECT Genre, MAX(Duration) AS Longest_Duration
58 FROM Song
59 GROUP BY Genre;
```

Below the query editor, the 'Result Grid' is displayed with the following data:

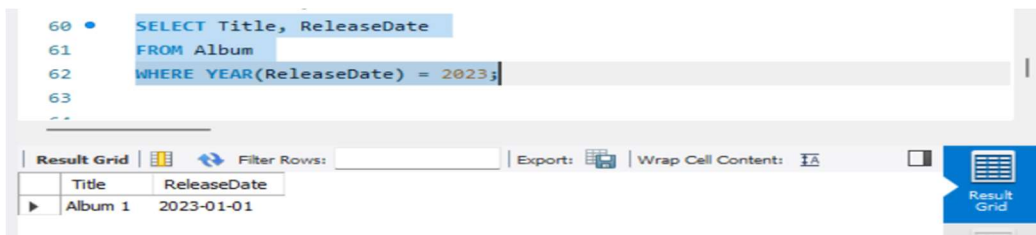
Genre	Longest_Duration
Pop	180
Rock	240
Hip Hop	200

--Retrieve the title and release date of albums released in 2023:

```
SELECT Title, ReleaseDate
```

```
FROM Album
```

```
WHERE YEAR(ReleaseDate) = 2023;
```



The screenshot shows a SQL query editor with the following query:

```
60 • SELECT Title, ReleaseDate
61 FROM Album
62 WHERE YEAR(ReleaseDate) = 2023;
63
```

Below the query editor, the 'Result Grid' is displayed with the following data:

Title	ReleaseDate
Album 1	2023-01-01

Creation of 5 views using the tables:-

View 1: Songs with Album Titles

```
CREATE VIEW SongWithAlbum AS
```

```
SELECT s.Title AS Song_Title, a.Title AS Album_Title FROM Song s
```

```
INNER JOIN Album a ON s.Album_ID = a.Album_ID;
```

View 2: Playlist Details

```
CREATE VIEW PlaylistDetails AS
SELECT p.Title AS Playlist_Title, p.CreationDate, u.Username AS User_Username
FROM Playlist p
INNER JOIN Users u ON p.User_ID = u.User_ID;
```

View 3: Album Releases by Genre

```
CREATE VIEW AlbumReleasesByGenre AS
SELECT Genre, COUNT(*) AS Album_Count
FROM Album
GROUP BY Genre;
```

View 4: Long Songs

```
CREATE VIEW LongSongs AS
SELECT *
FROM Song
WHERE Duration > 300;
```

View 5: Artists and Their Albums

```
CREATE VIEW ArtistsWithAlbums AS
SELECT ar.Name AS Artist_Name, al.Title AS Album_Title
FROM Artist ar
INNER JOIN Album al ON ar.Artist_ID = al.Artist_ID;
```

Thank You