New Ismailia National University

Faculty of Engineering

Music Playlist Manager Project

Supervised by: Dr/Hadeer Heussien

Eng /Maryam Abdel Rahman

Team Members:

- Maryam Ellsayed Ellaban
- Mina Bassem samir
- Nada Essam Ahmed
- Mohamed Hesham Abdelsattar
- Mohamed Amir Ghanam
- Ahmed Ibrahiem Fahiem
- Farouk Mohamed Farouk
- Amr Mohamed zohier
- Ahmed Khairy
- Mohamed Zakria

1. *Introduction*

This project is a comprehensive music database management system designed to organize and store information about artists, albums, songs, users, playlists, favorite songs, recently played songs, and user interactions with songs such as play, share, and download actions.

The system aims to provide a structured way to track music data, linking songs to their respective artists and albums, as well as to playlists. It also offers complex analytics and queries that allow users to gain insights into important statistics, such as the most popular songs, average song duration per artist, songs not added to any playlists, and more.

2. Entities and Attributes

1. Artist

- *ArtistID (PK)*
- *ArtistName (Unique, Mandatory)*

2. Albums

- *AlbumID (PK)*
- *AlbumName (Mandatory)*
- *ArtistID (FK)*

3. Songs

- *SongID (PK)*
- *SongName, Genre (Mandatory)*
- *Duration (Optional)*
- *ArtistID, AlbumID (FKs)*

4. Playlists

- *PlaylistID (PK)*
- *PlaylistName (Mandatory)*

5. PlaylistSongs

- *PlaylistID, SongID (Composite PKs, FKs)*
- *song_order (Optional)*
- *Relationships:*
    - *Playlists 1:N PlaylistSongs*
    - *Songs 1:N PlaylistSongs*

C. Favorites

- *FavoriteID (PK)*
- *SongID (FK)*

7. Recent

- *RecentID (PK)*
- *SongID (FK)*
- *Timestamp (Derived, Optional)*

8. Users

- *UserID (PK)*
- *UserName (Unique, Mandatory)*

3. SongTransactions

- *transaction_id (PK)*
- *user_id, song_id (FKs)*
- *action_type (Mandatory)*
- *action_date (Derived, Optional)*

3. SQL commands

Used:

- ## 3.1 Data Definition Language (DDL)
  - *Creates and defines database structures such as tables.*
  - *Examples in the project: Creating tables like Artist, Albums, Songs, Users, Playlists, etc.*
  - *Includes commands like CREATE DATABASE, CREATE TABLE, DROP DATABASE.*

3.2 Data Manipulation Language (DML)

- *Inserts, updates, and deletes data within the tables.*
- *Examples in the project: Inserting artists, albums, songs, users, playlists data; updating song durations; deleting records if needed.*

3.3 Data Control Language (DCL)

- *Manages database security and controls user access permissions.*

- *Examples in the project: Creating users and roles; granting SELECT permissions; revoking INSERT permissions; denying DELETE permissions.*

3.4 Transaction Control Language (TCL)

. Controls transactions to ensure data integrity.

. Examples in the project: Beginning transactions to add new artists and albums; committing or rolling back transactions; using savepoints for partial rollbacks.

3.5 Data Query Language (DQL)

.Retrieves data from the database for analysis and reporting.

.Examples in the project: Selecting songs, artists, playlists; performing aggregation queries such as counts, averages; joining multiple tables for complex reports.

4. Cardinality Ratios

1. Artist to Albums

- *Cardinality Ratio: One-to-Many (1:N)*
- *Description: Each artist can have multiple albums.*

2. Artist to Songs

- *Cardinality Ratio: One-to-Many (1:N)*
- *Description: Each artist can have multiple songs.*

3. Albums to Songs

- *Cardinality Ratio: One-to-Many (1:N)*
- *Description: Each album can contain multiple songs.*

4. Users to Playlists

- *Cardinality Ratio: One-to-Many (1:N)*
- *Description: Each user can create multiple playlists.*

5. Songs to Favorites

- *Cardinality Ratio: One-to-Many (1:N)*
- *Description: Each song can be marked as a favorite by multiple users*

5. participation

1. Artist to Albums

- *Participation: Total*
- *Description: Every artist must have at least one album.*

2. Artist to Songs

- *Participation: Total*
- *Description: Every song must be associated with an artist.*

3. Albums to Songs

- *Participation: Total*
- *Description: Every song must belong to an album.*

4. Users to Playlists

- *Participation: Partial*
- *Description: Not every user is required to have playlists.*

5. Playlists to PlaylistSongs

- *Participation: Total*
- *Description: Every playlist must contain at least one song.*

C. Songs to Favorites

- *Participation: Total*
- *Description: Every favorite entry must refer to a song.*

7. Users to SongTransactions

- *Participation: Total*
- *Description: Every transaction must be associated with a user.*

6.Relationships and Their Types

1. Artist to Albums

- Type: One-to-Many (1:N)
- Description: An artist can have multiple albums, but each album is associated with only one artist.

2. Artist to Songs

- Type: One-to-Many (1:N)
- Description: An artist can have multiple songs, but each song is associated with only one artist.

3. Albums to Songs

- Type: One-to-Many (1:N)
- Description: An album can contain multiple songs, but each song belongs to only one album.

4. Users to Playlists

- Type: One-to-Many (1:N)
- Description: A user can create multiple playlists, but each playlist is associated with only one user.

5. Playlists to PlaylistSongs

- Type: One-to-Many (1:N)
- Description: A playlist can contain multiple songs, but each entry in the PlaylistSongs table refers to only one playlist.

C.Songs to PlaylistSongs

- Type: One-to-Many (1:N)
- Description: A song can appear in multiple playlists, but each entry in the PlaylistSongs table refers to only one song.

7.Users to Favorites

- Type: One-to-Many (1:N)
- Description: A user can have multiple favorite songs, but each favorite entry refers to only one song.

8.Users to Recent

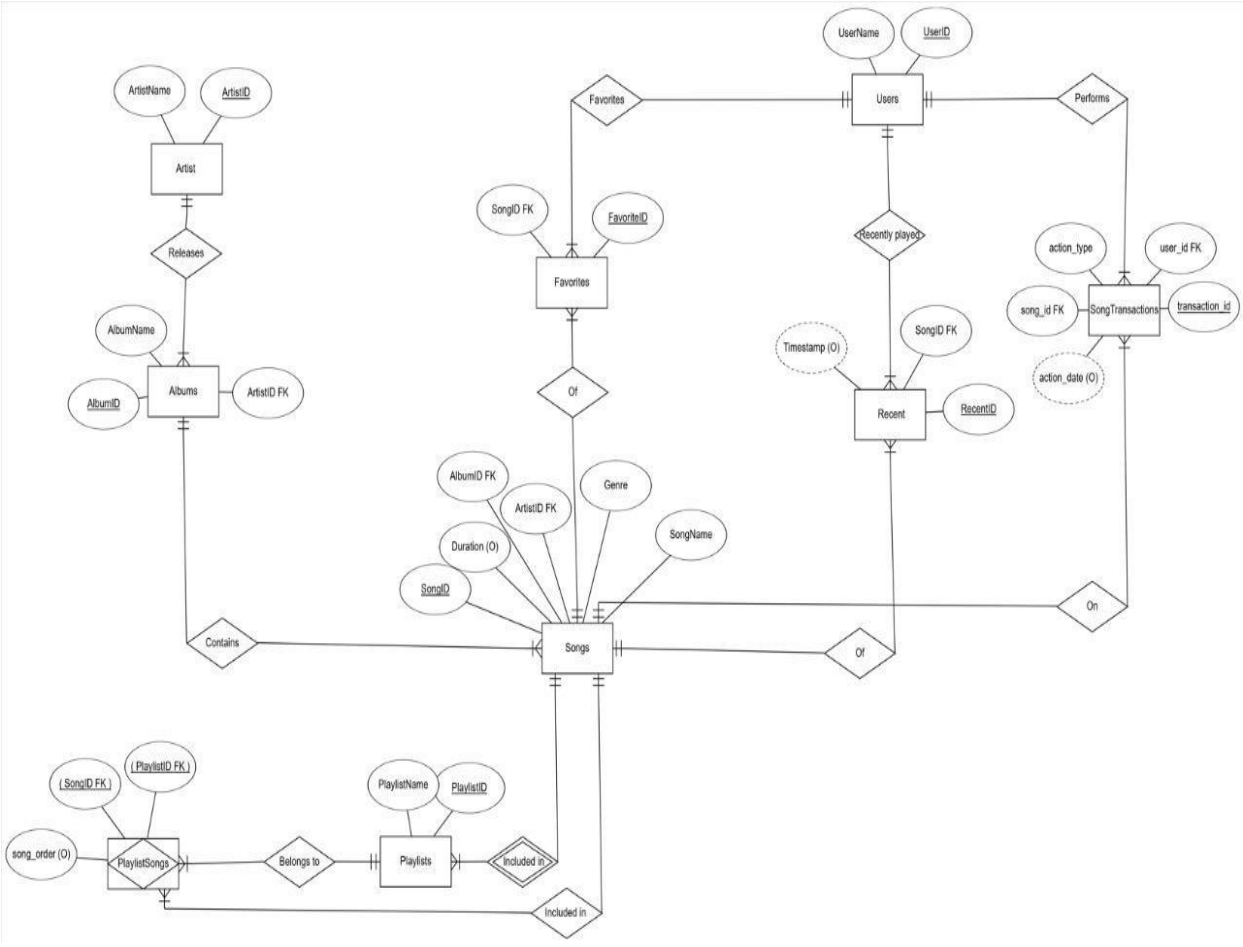
- Type: One-to-Many (1:N)
- Description: A user can have multiple recently played songs, but each recent entry refers to only one user.

3.Users to SongTransactions

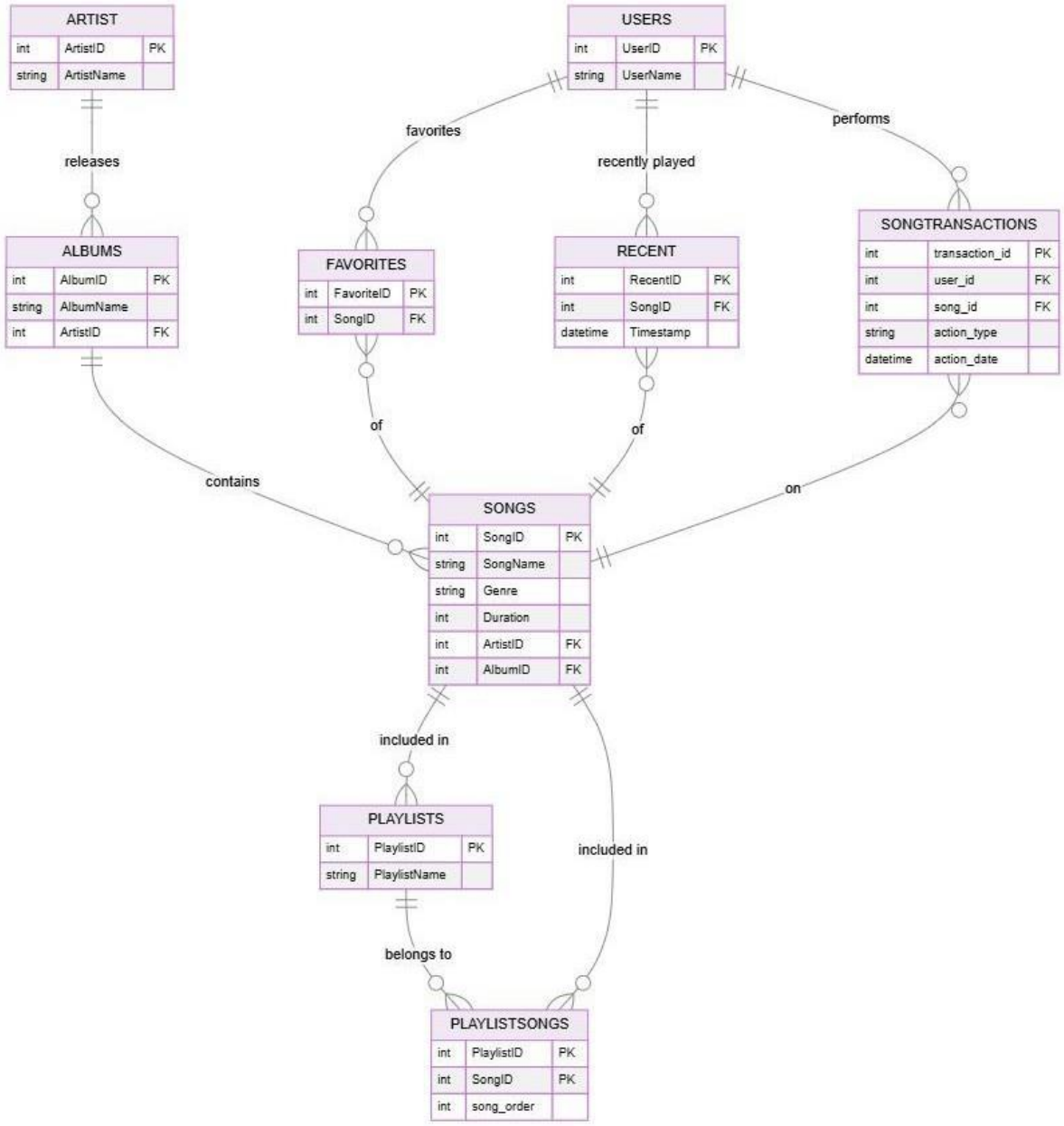
- Type: One-to-Many (1:N)

- Description: A user can have multiple song transactions, but each transaction is associated with only one user

7.Entity Relationship Diagram(ERD)_Relational Schema



8.Entity Relationship Diagram(ERD)_Conceptual ERD



9.Conclusion

In this project, we created a music database system to manage and organize music data, including artists, albums, songs, users, playlists, and transactions. The main goals were to build a structured database that allows easy data access and ensures data accuracy.

Key points of the project include:

1. Database Design: We defined important entities like Artists, Albums, Songs, Users, and Playlists, and established their relationships. For example, one artist can have many albums, and playlists can contain many songs.
2. Code Development: We wrote code to add, retrieve, update, and delete records in the database. This includes functions for adding new artists, albums, and songs, as well as managing user playlists and favorites.
3. Data Integrity: We set rules to ensure the database maintains accurate data. For instance, every artist must have at least one album, and each song must be linked to an artist.
4. User Experience: The system was designed to be user-friendly, allowing users to create playlists, mark songs as favorites, and track their listening history.

Overall, this project successfully demonstrates how to create a functional music database system. The design and code provide a solid foundation for future improvements, such as adding music recommendations and advanced search features. The project meets the initial requirements and sets the stage for further development in music management.