Project Description and Database Design

22207654 Wang Xinyi 22207668 Zhao Yanqi

22207669 Liao Wenqi 22207672 Cai Minjia

Directory

1. System Description & System Architecture

1.1 System Description

1. Project Overview

Our platform, Crakkle music, is a music streaming and social networking website that aims to provide users with a high-quality music experience through the interaction between the front-end web interface and the back-end database. The system supports three user roles: Listener, Musician and Admin, which fulfill the different needs of music listening, creation and uploading, and platform management respectively. The core functions include song playing, commenting, playlist management, dynamic charts based on the number of plays and downloads, as well as advertisement support, and it is committed to building a network platform with comprehensive functions, synthesized song data from the whole network, and social attributes.

2. User Roles and Functions

2.0 User

- All users of the platform first use their email address and set a password to register as a USER, and are required to fill in personal information, such as date of birth, gender, and nationality. Login with account password, after login you can change personal information and password.

2.1 Listener

- Registration and Login: user chooses to become a listener, and the system automatically generates a Listener ID. The system also collects the listener's music preferences at this point.
- Music Experience:
  - Browse and play songs, view metadata such as lyrics, genre, number of downloads, etc...
  - Create and manage playlists, add or remove songs.
  - View dynamic charts, including top songs based on number of plays and downloads.
  - Get the latest music news by being pushed song ads.
  - Download songs locally for offline listening.
- Social Interaction:
  - Comment on songs to express your opinion or feelings.
  - Follow your favorite musicians.
  - View popular comments to get the public's attitude towards music.
- Paid function: Subscribe to become a member to remove ads.

2.2 Musician

- Account Management: Musicians register in the same way. By submitting an application to the administrator, they can become a musician account and receive a musician ID issued by the system.
- Content Management:

- Upload songs, specify song metadata (e.g., lyrics, genre, and song title) and creative roles (composer, vocalist, producer, lyricist).
- View song plays, downloads, and user comments.

- **Community features:** post comments to interact with listeners and get feedback from fans.

2.3 Administrator (Admin)

- **Account Management:** Log in using the Admin_ID and password. After logging in, the password can be changed.
- **Content Management:**
  - View information of all listeners and musicians.
  - Manage user accounts and information, review musician identities,
  - Review and edit songs, comments and advertisement content to ensure the quality of the platform.
- **Resource Management:**
  - Upload, update and delete advertisement information (e.g., title, cost, duration).
  - Manage song information (e.g. lyrics, genres, etc.).

3. Technical Architecture

- **Front-end**: The plan is to build an interactive web interface using **HTML** and **JavaScript (Vue.js)** to provide an intuitive display of music playback, reviews and charts.
- **Backend**: Developed using **Java**, connecting to **MySQL database** to store and process data, ensuring efficient data query and update.
- **Database**: **relational database (MySQL)**, the main tables include:
  - User: stores basic user information (user name, password, user type, etc.).
  - Song: stores song information (name, play count, download count, lyrics).
  - Comment: stores comment information (content, time, song ID).
  - Playlist: manages playlists and their songs.
  - Musician: records information about musicians and their creative roles.
  - Advertisement: Manage advertisement information.
  - Genre: supports song genre categorization.
- **Chart implementation:** Dynamically generate charts by querying total_views in Song table and download_count in download table, e.g.
  - Playlist ranking: SELECT song_id, name FROM Song ORDER BY total_views DESC LIMIT 10
  - Download count ranking: SELECT song_id, name FROM Download ORDER BY download_count DESC LIMIT 10

4. Assumption:

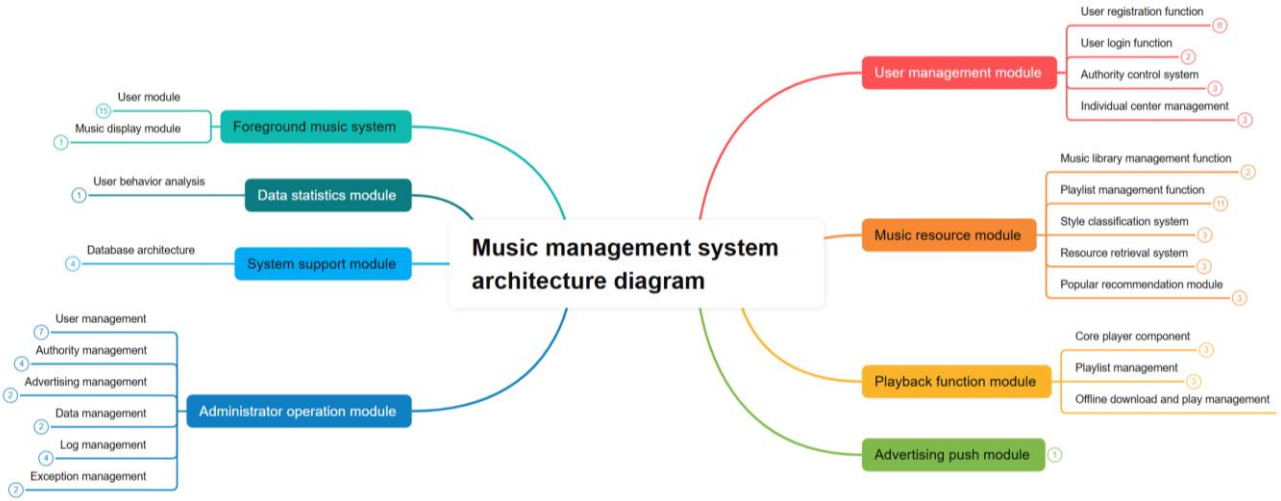
- The number of song plays and downloads are available on all platforms across the network.
- Each listener and musician in the system is registered and identified by a unique user ID, and the password information will be encrypted for safe storage.
- Each song's play count and download count are updated in real time, and the corresponding charts reflect the latest data instantly.

- Multiple users posting comments or creating playlists at the same time will not cause conflicts.
- There is no conflict between the administrator's editing of songs or advertisements and the operations of users and musicians.
- After a song is uploaded, genres and other metadata are manually specified by the musician who uploaded the song or by the administrator.
- All advertisements are fixed in position and scroll in order with the same cost; the cost of advertisements is calculated only according to the time of placement.

5. Future Plans

- **Personalized Recommendation**: Recommend relevant songs and musicians based on user preference, song list and history.
- **Comment Reply**: Support comment reply function.
- **Real-time notifications**: Users can receive instant push information such as new song releases and musician updates.
- **Data analysis**: Provide musicians with trend visualization of song play, download and comment data to assist content optimization.
- **Personalized Recommendation**: Building on the original personalized recommendations, the system suggests relevant songs and musicians based on the user's listening history and preference selections.

1.2 System Architecture Diagram



The overall architecture of the music management system consists of eight modules, each responsible for different functions and duties.

1. User Management Module

This module handles basic user account functions. First is the **user registration function**, where users need to register using an email address. During registration, users must select

their account type (Listener or Musician), and the system assigns different functional permissions based on the selection. For example, listeners need to fill in personal information and music preferences so that the system can later recommend suitable music based on their interests. After successful registration, the system notifies the user.

Next is the **user login function**, where users log in with their registered account information. If login fails, the system provides relevant prompts.

Moreover, this module includes an **authority control system**. The system automatically assigns permissions and access scopes based on the user's account type (Musician or Listener).

Finally, in **individual center management**, users can modify personal details, manage playlists, and check their music playback history.

2. Music Resource Module

This module manages music content collection and organization. First is the **music library management function**, which constructs the system's audio file storage structure by entering song metadata.

Then comes the **playlist management function**, allowing users to create new playlists, edit existing playlists, and search for songs by name within playlists.

Next is the **style classification system**, which systematically categorizes songs based on their musical style.

Additionally, the **resource retrieval system** enables users to search for music resources using keywords, with the system displaying content based on song ID.

The module also includes the **popular recommendation module**, which recommends trending music to users based on song play counts and downloads.

3. Playback Function Module

This module implements core music playback functions. It includes the **core player component**, which supports playback progress adjustment, volume control, and playback mode switching.

Users can also customize the playback order through **playlist management**, choosing from sequential play, shuffle play, single-loop play, and list-loop play.

Additionally, the system offers **offline download and play management**, allowing users to download songs for offline listening.

4. Advertising Push Module

This module manages advertising content, including **advertisement display methods**, **push timing**, and **placement settings**, aiming to improve ad exposure and user engagement.

5. Foreground Music System

This module forms the user-facing front-end, consisting of multiple submodules.
It includes the **user module**, which provides the user registration, login, personal center, and permission management interfaces.
Additionally, the **music display module** categorizes and showcases music content based on styles and genres, facilitating browsing and selection.

6. Data Statistics Module

This module **analyzes user behavior and system data**, including **user playback behavior analysis** (planned for future implementation) and **song play count statistics**, providing data support for recommendation systems and management decisions.

7. System Support Module

This module offers foundational support for the entire system, including **database architecture design**, such as primary-secondary read-write separation configuration, song information partitioning strategies, caching mechanisms, and data backup and recovery plans to ensure system stability and reliability.

8. Administrator Operation Module

This module is designed for system administrators, enabling comprehensive management of the entire system, including:

- **User management**: Adding, editing, and deleting user information; auditing user permission usage and handling abnormal access behaviors.
- **Permission management**: Assigning specific permissions (such as extra privileges for paid users) and managing permission approval.
- **Advertising management**: Adding, editing, and deleting advertisement content.
- **Data management**: Performing data backup, recovery, and maintenance tasks.
- **Log management**: Recording user behavior and system changes, supporting error tracing and system recovery.
- **Exception management**: Capturing system errors and providing exception reports and resolution mechanisms.

1.3 Tasks

1. Requirement Analysis and Planning

- Task 1.1: Define Functional Requirements
  - **Description**: Based on the architecture diagram, confirm user roles and functional modules.
  - **Subtasks**:
    - **User Roles**:
      - **Listener**: Play songs, create playlists, comment, follow musicians, view charts,

download songs, personalized recommendations.

- **Musician**: Upload songs, view play counts and comments, apply for ad promotions, interact with other users.
- **Admin**: Manage users, review musicians and song information, upload and manage ads.
- **Core Functions (Architecture Diagram)**:
- **User Management Module**: Registration, login, permission control, personal center.
- **Music Resource Module**: Song upload, categorization, search, trending recommendations.
- **Playback Function Module**: Song playback, playlist management, offline download.
- **Ad Push Module**: Ad creation, scheduled push, click statistics.
- **Social Interaction**: Comments, likes, follows.
- **Non-Functional Requirements**: Responsive UI, security (password encryption, permission control), scalability (support for future recommendation algorithms).

- **Task 1.2: Define Technical Requirements**
  - **Tech Stack**: Frontend using Vue.js (Vue 3), backend using Spring Boot (Java), database using MySQL.
  - **Communication Method**: RESTful API.
  - **Third-Party Tools**: (To be specified).

2. Database Design

Task 2.1: Design ER Diagram Based on System Requirements

- **Description**: Design an Entity-Relationship (ER) diagram based on project requirements, clarifying entities, attributes, and relationships.
- **Tasks**:
  - **Identify Entities**: Admin, User, Listener, Musician, Song, Comments, Playlist, Advertisement, Genre, Language, Region, Era.
  - **Define Relationships**:
  - **1:1**: Admin and User (via admin_id).
  - **1:N**: Listener and User (via user_id).
  - **M:N**: Follow, Slike, Download, Contain, Listen_to, Prefer_genre, etc.
  - **Define Attributes**: e.g., User.user_id, Song.sname, Comments.content.
  - **Mark Super/Subtypes**: User as supertype; Listener, Musician as subtypes.

Task 2.2: Convert ER Diagram to Relational Model

- **Description**: Map the ER diagram to a relational model, defining table structures, keys, and relationships.
- **Tasks:**
  - Confirm entities and relationships based on the ER diagram.
  - Map regular entities to tables: e.g., User, Song, Playlist.
  - Map M:N relationships to intermediate tables: e.g., Follow(user_id, musician_id),
  - Map subtypes: Listener, Musician linked to User.user_id via foreign keys.

- Define primary keys: e.g., user_id, song_id.
- Define foreign keys: e.g., Comments.user_id references User.user_id.
- Ensure primary keys, foreign keys, and constraints support data integrity.

Task 2.3: Create Table Statements

Below are MySQL CREATE TABLE statements, precisely matching the relational diagram:

```
-- Create tables without dependencies first
CREATE TABLE Genre (
    genre_name VARCHAR(100) PRIMARY KEY
);

CREATE TABLE Language (
    language_name VARCHAR(50) PRIMARY KEY
);

CREATE TABLE Region (
    region_name VARCHAR(100) PRIMARY KEY
);

CREATE TABLE Era (
    era_name VARCHAR(50) PRIMARY KEY
);

-- Create user-related tables
CREATE TABLE User (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL, -- Encrypted password
    join_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    date_of_birth DATE,
    gender ENUM('Male', 'Female', 'Other'),
    user_type ENUM('Listener', 'Musician') NOT NULL,
    nationality VARCHAR(100),
    profile_picture VARCHAR(255)
);

CREATE TABLE Admin (
    admin_id INT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    aname VARCHAR(100) NOT NULL
);
```

```sql
CREATE TABLE Listener (
    listener_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT UNIQUE NOT NULL,
    listener_name VARCHAR(100) NOT NULL,
    is_subscribed BOOLEAN DEFAULT FALSE,
    subscription_end_date DATE,
    FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE CASCADE
);

CREATE TABLE Musician (
    musician_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT UNIQUE NOT NULL,
    mname VARCHAR(100) NOT NULL,
    FOREIGN KEY (user_id) REFERENCES User(user_id) ON DELETE CASCADE
);

-- Create tables dependent on Language, Region, etc.
CREATE TABLE Song (
    song_id INT AUTO_INCREMENT PRIMARY KEY,
    sname VARCHAR(255) NOT NULL,
    language_name VARCHAR(50),
    lyric TEXT,
    upload_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_views BIGINT DEFAULT 0,
    region_name VARCHAR(100),
    comment_count INT DEFAULT 0,
    FOREIGN KEY (language_name) REFERENCES Language(language_name),
    FOREIGN KEY (region_name) REFERENCES Region(region_name)
);

CREATE TABLE Playlist (
    playlist_id INT AUTO_INCREMENT PRIMARY KEY,
    listener_id INT NOT NULL,
    pname VARCHAR(255) NOT NULL,
    create_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    number_of_song INT DEFAULT 0,
    description TEXT,
    FOREIGN KEY (listener_id) REFERENCES Listener(listener_id) ON DELETE CASCADE
);

CREATE TABLE Advertisement (
    ad_id INT AUTO_INCREMENT PRIMARY KEY,
    admin_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
```

```
    content TEXT,
    cost DECIMAL(10, 2) NOT NULL,
    survive_time INT NOT NULL,
    create_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    song_id INT,
    FOREIGN KEY (admin_id) REFERENCES Admin(admin_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id)
);

-- Create M:N relationship tables
CREATE TABLE Comments (
    comment_id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT,
    admin_id INT,
    song_id INT NOT NULL,
    content TEXT NOT NULL,
    comment_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    comment_count INT DEFAULT 0,
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (admin_id) REFERENCES Admin(admin_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id) ON DELETE CASCADE,
    CONSTRAINT chk_commenter CHECK ((user_id IS NULL) != (admin_id IS NULL))
);

CREATE TABLE Musician_role (
    musician_id INT NOT NULL,
    mrole VARCHAR(50) NOT NULL,
    PRIMARY KEY (musician_id, mrole),
    FOREIGN KEY (musician_id) REFERENCES Musician(musician_id) ON DELETE CASCADE
);

CREATE TABLE Glike (
    listener_id INT NOT NULL,
    gname VARCHAR(100) NOT NULL,
    PRIMARY KEY (listener_id, gname),
    FOREIGN KEY (listener_id) REFERENCES Listener(listener_id),
    FOREIGN KEY (gname) REFERENCES Genre(genre_name)
);

CREATE TABLE Good_at (
    musician_id INT NOT NULL,
    gname VARCHAR(100) NOT NULL,
    PRIMARY KEY (musician_id, gname),
    FOREIGN KEY (musician_id) REFERENCES Musician(musician_id),
```

```
    FOREIGN KEY (gname) REFERENCES Genre(genre_name)
);

CREATE TABLE Listen_to (
    song_id INT NOT NULL,
    listener_id INT NOT NULL,
    listen_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (song_id, listener_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id),
    FOREIGN KEY (listener_id) REFERENCES Listener(listener_id)
);

CREATE TABLE Contain (
    song_id INT NOT NULL,
    playlist_id INT NOT NULL,
    added_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (song_id, playlist_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id),
    FOREIGN KEY (playlist_id) REFERENCES Playlist(playlist_id) ON DELETE CASCADE
);

CREATE TABLE Download (
    song_id INT NOT NULL,
    listener_id INT NOT NULL,
    download_count INT DEFAULT 1,
    download_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (song_id, listener_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id),
    FOREIGN KEY (listener_id) REFERENCES Listener(listener_id)
);

CREATE TABLE Slike (
    listener_id INT NOT NULL,
    song_id INT NOT NULL,
    like_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (listener_id, song_id),
    FOREIGN KEY (listener_id) REFERENCES Listener(listener_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id)
);

CREATE TABLE Contribute (
    song_id INT NOT NULL,
    musician_id INT NOT NULL,
    PRIMARY KEY (song_id, musician_id),
```

```
    FOREIGN KEY (song_id) REFERENCES Song(song_id),
    FOREIGN KEY (musician_id) REFERENCES Musician(musician_id)
);

CREATE TABLE Masterpiece (
    song_id INT NOT NULL,
    musician_id INT NOT NULL,
    PRIMARY KEY (song_id, musician_id),
    FOREIGN KEY (song_id) REFERENCES Song(song_id),
    FOREIGN KEY (musician_id) REFERENCES Musician(musician_id)
);

CREATE TABLE Have (
    song_id INT NOT NULL,
    gname VARCHAR(100) NOT NULL,
    PRIMARY KEY (song_id, gname),
    FOREIGN KEY (song_id) REFERENCES Song(song_id),
    FOREIGN KEY (gname) REFERENCES Genre(genre_name)
);

CREATE TABLE Follow (
    user_id INT NOT NULL,
    musician_id INT NOT NULL,
    follow_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, musician_id),
    FOREIGN KEY (user_id) REFERENCES User(user_id),
    FOREIGN KEY (musician_id) REFERENCES Musician(musician_id)
);

CREATE TABLE Prefer_genre (
    listener_id INT NOT NULL,
    genre_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (listener_id, genre_name),
    FOREIGN KEY (listener_id) REFERENCES Listener(listener_id),
    FOREIGN KEY (genre_name) REFERENCES Genre(genre_name)
);

CREATE TABLE Prefer_language (
    listener_id INT NOT NULL,
    language_name VARCHAR(50) NOT NULL,
    PRIMARY KEY (listener_id, language_name),
    FOREIGN KEY (listener_id) REFERENCES Listener(listener_id),
    FOREIGN KEY (language_name) REFERENCES Language(language_name)
);
```

```sql
CREATE TABLE Prefer_region (
    listener_id INT NOT NULL,
    region_name VARCHAR(100) NOT NULL,
    PRIMARY KEY (listener_id, region_name),
    FOREIGN KEY (listener_id) REFERENCES Listener(listener_id),
    FOREIGN KEY (region_name) REFERENCES Region(region_name)
);

CREATE TABLE Prefer_era (
    listener_id INT NOT NULL,
    era_name VARCHAR(50) NOT NULL,
    PRIMARY KEY (listener_id, era_name),
    FOREIGN KEY (listener_id) REFERENCES Listener(listener_id),
    FOREIGN KEY (era_name) REFERENCES Era(era_name)
);
```

Task 2.4: Initialize Test Data

Insert sample data:

```sql
-- Insert Genre data
INSERT INTO Genre (genre_name) VALUES ('Pop'), ('Rock'), ('Jazz');

-- Insert Language data
INSERT INTO Language (language_name) VALUES ('Chinese'), ('English');

-- Insert Region data
INSERT INTO Region (region_name) VALUES ('China'), ('USA');

-- Insert Era data
INSERT INTO Era (era_name) VALUES ('2000s'), ('2010s');

-- Insert Admin data
INSERT INTO Admin (email, password, aname)
VALUES ('admin@music.com', 'hashed_admin_pass', 'Admin1');

-- Insert User data (only Listener and Musician)
INSERT INTO User (email, password, user_type, join_date, date_of_birth, gender,
nationality)
VALUES
    ('listener1@music.com', 'hashed_listener_pass', 'Listener', CURRENT_TIMESTAMP, '1995-
05-10', 'Female', 'USA'),
    ('musician1@music.com', 'hashed_musician_pass', 'Musician', CURRENT_TIMESTAMP, '1990-
03-15', 'Male', 'China');
```

```sql
-- Insert Listener data
INSERT INTO Listener (user_id, listener_name, is_subscribed, subscription_end_date)
SELECT user_id, 'ListenerOne', TRUE, '2025-12-31'
FROM User
WHERE email = 'listener1@music.com';

-- Insert Musician data
INSERT INTO Musician (user_id, mname)
SELECT user_id, 'MusicianOne'
FROM User
WHERE email = 'musician1@music.com';

-- Insert Song data
INSERT INTO Song (sname, language_name, lyric, upload_time, total_views, region_name,
comment_count)
VALUES
    ('SongA', 'Chinese', 'Lyrics for SongA...', CURRENT_TIMESTAMP, 1000, 'China', 0),
    ('SongB', 'English', 'Lyrics for SongB...', CURRENT_TIMESTAMP, 500, 'USA', 0);

-- Insert Playlist data
INSERT INTO Playlist (listener_id, pname, create_date, number_of_song, description)
SELECT listener_id, 'My Favorites', CURRENT_TIMESTAMP, 2, 'My favorite songs'
FROM Listener
WHERE listener_name = 'ListenerOne';

-- Insert Contain data (add songs to the playlist)
INSERT INTO Contain (song_id, playlist_id, added_time)
VALUES
    (1, 1, CURRENT_TIMESTAMP),
    (2, 1, CURRENT_TIMESTAMP);

-- Insert Comments data (User and Admin comments)
INSERT INTO Comments (user_id, admin_id, song_id, content, comment_time)
VALUES
    ((SELECT user_id FROM User WHERE email = 'listener1@music.com'), NULL, 1, 'Great
song!', CURRENT_TIMESTAMP),
    ((SELECT user_id FROM User WHERE email = 'listener1@music.com'), NULL, 2, 'Love it!',
CURRENT_TIMESTAMP);


-- Insert Download data (test download count)
INSERT INTO Download (song_id, listener_id, download_count, download_time)
VALUES
    (1, (SELECT listener_id FROM Listener WHERE listener_name = 'ListenerOne'), 2,
CURRENT_TIMESTAMP),
```

```
    (2, (SELECT listener_id FROM Listener WHERE listener_name = 'ListenerOne'), 1,
CURRENT_TIMESTAMP);
```

Task 2.5: Create Indexes

Optimize query performance:

```
CREATE INDEX idx_song_views ON Song(total_views);
CREATE INDEX idx_comments_song ON Comments(song_id);
CREATE INDEX idx_download_song ON Download(song_id);
CREATE INDEX idx_listen_to_song ON Listen_to(song_id);
```

Task 2.6: Implement Leaderboard Logic

SQL queries to dynamically generate leaderboards:

```
-- Top 10 songs by play count
SELECT song_id, sname, total_views
FROM Song
ORDER BY total_views DESC
LIMIT 10;

-- Top 10 songs by download count
SELECT s.song_id, s.sname, SUM(d.download_count) as total_downloads
FROM Song s
JOIN Download d ON s.song_id = d.song_id
GROUP BY s.song_id, s.sname
ORDER BY total_downloads DESC
LIMIT 10;
```

- Screenshots of some of the table

music.User

user_id	email	password	join_date	date_of_birth	gender	user_type	nationality	profile_picture
1	listener1@music.com	hashed_listener_pass	2025-04-12 23:09:35	1995-05-10	Female	Listener	USA	<null>
2	musician1@music.com	hashed_musician_pass	2025-04-12 23:09:35	1990-03-15	Male	Musician	China	<null>

song_id	sname	language_name	lyric	upload_time	total_views	region_name	comment_count
1	SongA	Chinese	Lyrics for SongA...	2025-04-12 23:09:35	1000	China	0
2	SongB	English	Lyrics for SongB...	2025-04-12 23:09:35	500	USA	0

musician_id	user_id	mname
1	2	MusicianOne

listener_id	user_id	listener_name	is_subscribed
1	2	ListenerOne	1

3. Backend Development

- Task 3.1: Set Up Spring Boot Project
- Task 3.2: Create Core Functions
  - Implement entity classes and repositories, mapping to database tables.
  - Develop RESTful APIs to support:

        - User registration, login, role management.
        - Listener functions: song playback, playlist management, commenting, downloading, liking.
        - Musician functions: song upload, statistics viewing.
        - Admin functions: user management, musician review, ad management.
- **Task 3.3: Handle File Upload**
    - Implement song upload, storing files locally or in cloud services.
    - Update file paths in the database.

4. Frontend Development (Based on Vue.js, Preliminary Plan)

The first phase focuses on core functions, quickly building basic interfaces and interactions, with details refined in later iterations.

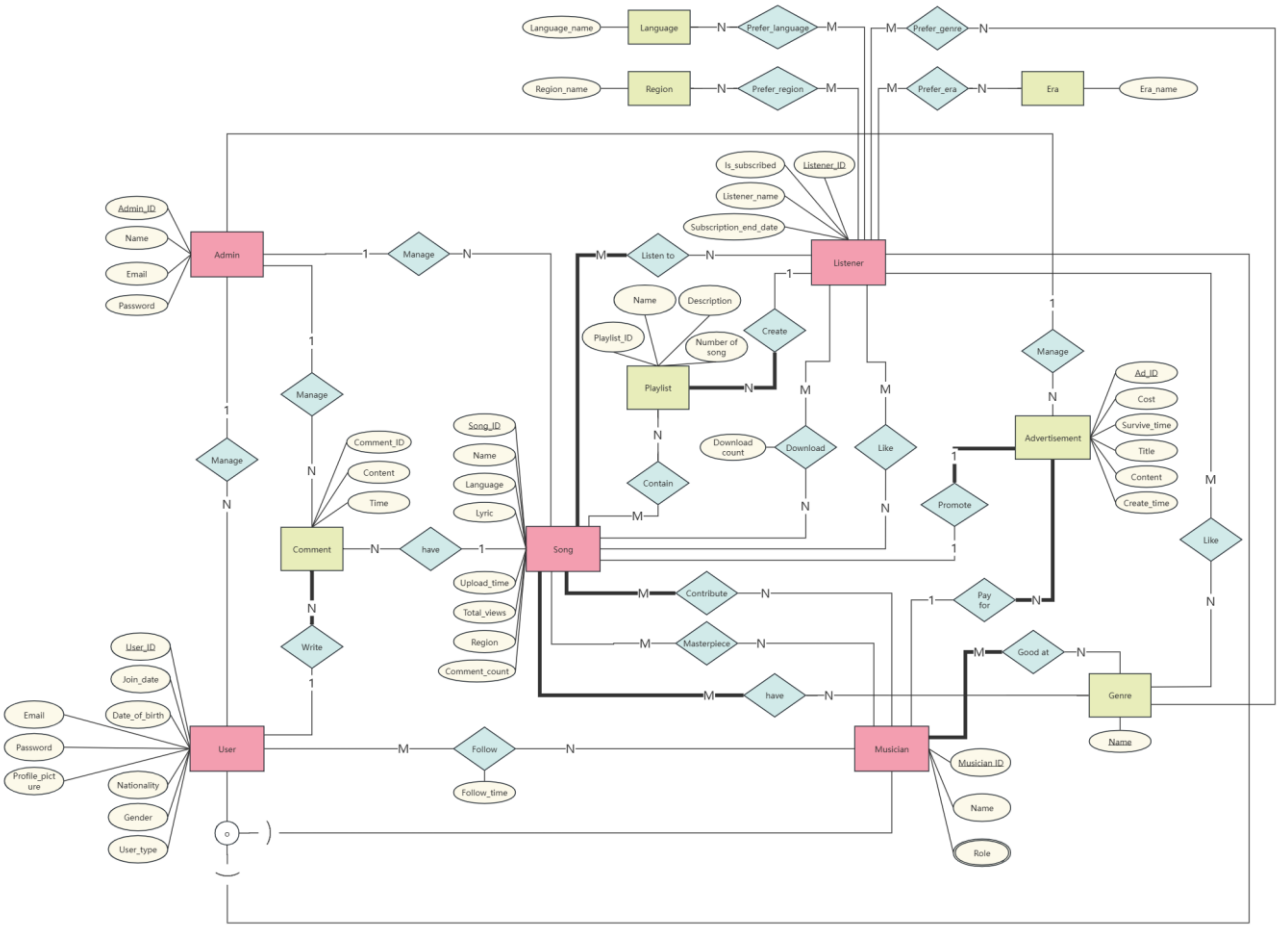
- **Task 4.1: Plan Core Pages**
    - Design main pages:
        - **Homepage**: Display song lists, leaderboards, ads.
        - **Login/Register Page**: User authentication.
        - **Playlist Page**: Manage playlists.
        - **Song Details Page**: Show lyrics, comments, support playback and liking.
        - **Musician Page**: Upload songs, view statistics.
        - **Admin Page**: Manage users and content.
- **Task 4.2: Implement Basic Interactions**
- **Task 4.3: Optimize User Experience**
    - Add loading states and error prompts.
    - Ensure responsive interface, supporting both mobile and desktop.

5. Test Data Interaction and User Workflows

- **Description**: Verify the correctness of database interactions and user workflows.
- **Tasks**:
    - **Backend Testing**: Use JUnit to test APIs (e.g., song upload, comment creation).
    - **Data Interaction Testing**: Insert test data, verify queries (e.g., leaderboards) and updates (e.g., play counts).
    - **User Workflow Testing**:
        - **Listener**: Register, log in, play songs, add comments.
        - **Musician**: Upload songs, view statistics.
        - **Admin**: Review musicians, add ads.
    - Generate a test report documenting passed/failed test cases.

2. Database Design

2.1 ER diagram



The ER diagram describes a comprehensive model of a music platform: users of the platform are categorized into two roles—**Listener** and **Musician**—with an **overlapping inheritance structure**. The base entity **User** includes fundamental attributes such as user ID, join date, birthdate, email, password, profile picture, nationality, and gender.

Listeners have additional attributes including subscription information, listener name, and listener ID. They engage in platform activities through many-to-many relationships involving **downloading**, **liking**, and **listening to songs**, as well as **creating playlists** that include songs. Their preferences regarding region, language, genre, and era are also recorded.

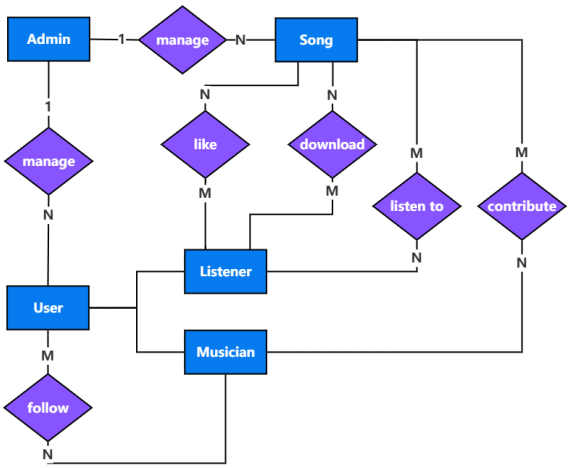
Musicians, on the other hand, have attributes such as musician ID, stage name, and role. They participate in many-to-many relationships through **contributing songs**, **owning representative works**, and **paying for advertisements** (a one-to-many relationship). Each **advertisement** is exclusively linked to one song in a one-to-one promotional relationship, meaning a single advertisement promotes only one specific song.

On the **social level**, the platform supports user interaction via **many-to-many follow relationships** between users and musicians, as well as **one-to-many relationships** where songs receive comments and users write comments.

Platform management is handled by **administrators (Admin)**. Admins log into admin accounts and are connected to **songs, users, comments, and advertisements** through **one-to-many management relationships**, ensuring all content and interactions are monitored and reviewed.

This design not only meets the needs for administrative oversight but also supports complex user behaviors. By separating roles (Listener/Musician) and utilizing attribute inheritance (with User as the parent entity), it allows for flexible expansion of the user system—ultimately constructing a robust and complete music platform.

- **Core entity-relationship diagram:**



To present the core relationships between entities more intuitively, we extracted the main entities from the full ER diagram—including **song**, **admin**, **user**, **listener**, and **musician**—and created the entity-relationship diagram. This diagram highlights that listener and musician are two subclasses of the user entity, each inheriting the basic attributes and behavioral relationships of users.

At the user level, a user can follow a musician, and the user's account is managed by a system administrator (admin).

As subclasses of user, listeners engage with songs through three types of interactions: like, download, and listen to, demonstrating the diverse ways listeners participate in music consumption. On the other hand, musicians have the ability to contribute songs, emphasizing their role as content creators within the system.

The system's administrative capabilities are also reflected: admins are responsible not only for managing users, but also for managing songs, ensuring the quality and compliance of content across the platform.

Overall, this architecture is designed to clearly illustrate the interactions between users and music, as well as the functional roles of different participants within the system.

2.2 Mapping the ER diagram to a Relational Model

Black word is the mapping of regular entity type

Blue word is the mapping of 1:1 relationships

Pink word is the mapping of 1:N relationships

Red word is the mapping of M:N relationships

Orange word is the mapping of Multivalued Attribute

Purple word is the mapping of Supertypes/Subtypes

User	User_ID	Join_date	User_type	Email	Password	Profile_picture	Gender	Date_of_birth	Nationality	Admin_ID

Admin	Admin_ID	Aname	Email	Password

Listener	Listener_ID	Listener_name	Subscription_end_date	Is_subscribed	User_ID

Musician	Musician_ID	Mname	User_ID

Song	Song_ID	Sname	Upload_ time	Total_views	Language	Lyric	Region	Comment_count	Ad_ID

Comments	Comment_ID	Content_time	Ad_ID	User_ID	Song_ID

Playlist	Playlist_ID	Pname	Number_of_song	Description	Listener_ID

Advertisement	Ad_ID	Cost	Survive_time	Title	Content	Create_time	Song_id	Admin_ID	Musician_ID

Genre	Genre_name

Language	Language_name

Region	Region_name

Era	Era_name

Musician_role	Mrole	Musician_ID

Glike	Listener_ID	Genre_name

Good_at	Musician_ID	Genre_name

Listen_to	Song_ID	Listener_ID

Contain	Song_ID	Playlist_ID

Download	Song_ID	Listener_ID	Download_count

Slike	Listener_ID	Song_ID

Contribute	Song_ID	Musician_ID

Masterpiece	Song_ID	Musician_ID

Have	Song_ID	Genre_name

Follow	User_ID	Musician_ID	Follow_time

Prefer_genre	Listener_ID	Genre_name

Prefer_language	Listener_ID	Language_name

Prefer_region	Listener_ID	Region_name

Prefer_era	Listener_ID	Era_name

2.4 Team Member Contributions

Wang Xinyi: Partial ER diagram drawing + overall beautification; Drawing of entity relationship diagrams + textual description; Overall report formatting and integration; Reviewing others' content

Liao Wenqi: Partial ER diagram drawing; Textual description of the system architecture diagram; Mapping to relational model + later revisions; Reviewing others' content

Zhao Yanqi: Partial ER diagram drawing; Textual description of the ER diagram; Mapping to relational model; Reviewing others' content

Cai Minjia: Partial ER diagram drawing; Writing the system description; Drawing the system architecture diagram; Describing the design and implementation steps; Writing TABLE statements; Reviewing others' content