The University of Texas at Dallas

WonderBeatz

Ezrela Amoako, Alexandra Swift, Cristina Kovacs, Kayla Tucker

Prof. Jalal Omer
Database Systems 4347.003
3 December 2023

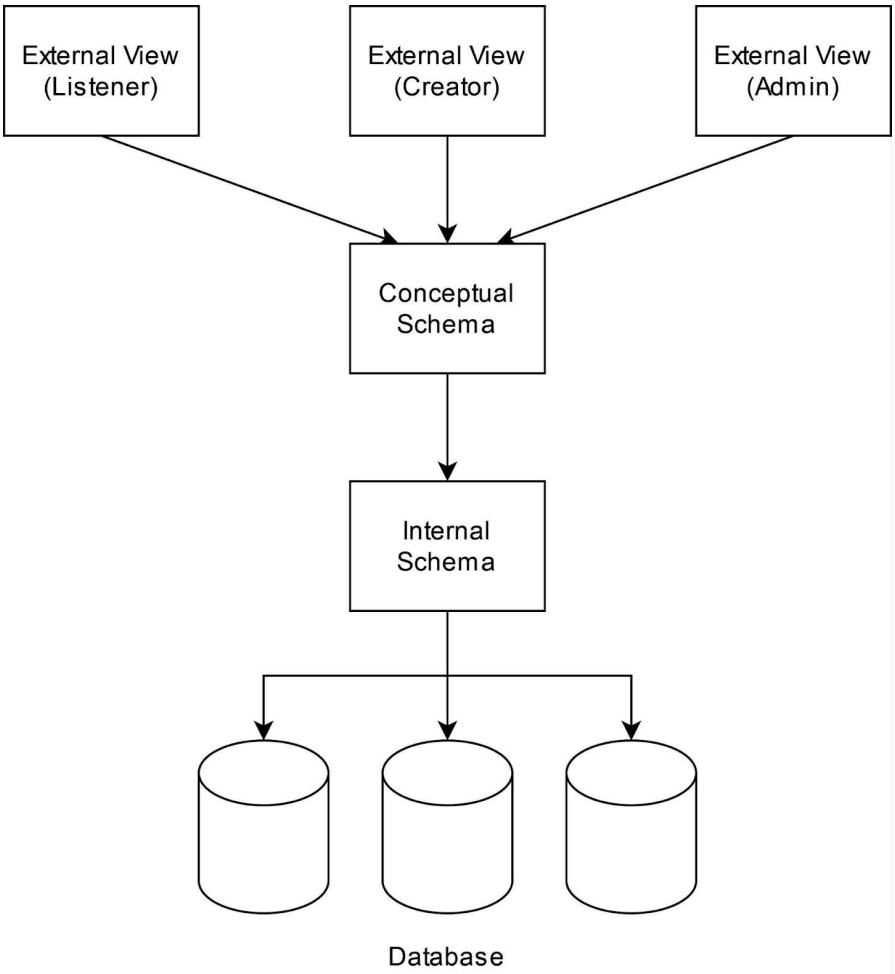
Table of Contents

Introduction

WonderBeatz was created as a way for sound editors and listeners to be able to search for songs and create playlists based on not just the creator of a song but details of the song as well. These features include the beats per minute (BPM) and the pitch of a song. Having these features allows the listeners to create high quality playlists more easily as the songs will flow together. Additionally, it also allows song creators to upload their songs so the public can enjoy them and have up to date and accurate information on their songs.

For this final report, we will discuss the system requirements such as what each interface is designed for as well as the functional dependencies we have in our database. Next we will discuss the conceptual design of our database which includes the entities and all of their relationships. This will be followed by the logical database scheme that takes the previous sections work and turns it into sql commands. After detailing our database, we will then discuss the normalization and how we made a more efficient database. The next two sections will give details on how to install our application and how to interact with it. Lastly, we will discuss any future upgrades we plan to have.

System Requirements



External View (Listener)

Listener should be able to:

1) Create account
    1.1) Input username
    1.2) Input password
2) Log in
    2.1) Input username
    2.2) Input password
3) Create playlist
    3.1) Add song to playlist
    3.2) Remove song from playlist
4) Search for song
    4.1) Search by name
    4.2) Search by artist

4.3) Search by BPM
4.4) Search by pitch

5) View account
4.1) Search playlists
4.2) Add playlist
4.2) Request assistance

Non-Functional requirements:

1. Buttons should be present to return to the previous page
2. Songs should be displayed by date
3. Login should prevent SQL injections

External View (Creator)

Creator should be able to:

1) Create account
1.1) Input username
1.2) Input password
2) Log in
2.1) Input username
2.2) Input password
3) Add song
3.1) Input song name
3.2) Input song pitch
3.2) Input song BPM
4) Remove song
4.1) Input song name
5) Modify song
5.1) Change BPM
5.2) Change Pitch
6) View account
6.1) Search songs
6.2) Request assistance

Non-Functional requirements:

1. Buttons should be present to return to the previous page
2. Songs should be displayed by date
3. Songs to be modified should be searched by name
4. Login should prevent SQL injections

External View (Admin)

Admin should be able to:

1) Create account
1.1) Input username

    1.2) Input password
2) Log in
    2.1) Input username
    2.2) Input password
3) View assistance requests
4) View account

Non-Functional requirements:

1. Buttons should be present to return to the previous page
2. Assistance requests should be ordered by date sent
3. Login should prevent SQL injections

Conceptual Schema

Conceptual schema should be able to:

1) Correctly display data
    1.1) Read in data from internal schema

Non-Functional requirements:

1. Data should be consistent and correct with no redundancy issues
2. Schema should define proper constraints and keys

Internal Schema

1) Retrieve data
    1.1) Receive input for data path
    1.2) Search for path location

Non-Functional requirements:

1. Schema should define clear locations and paths for data retrieval

Database

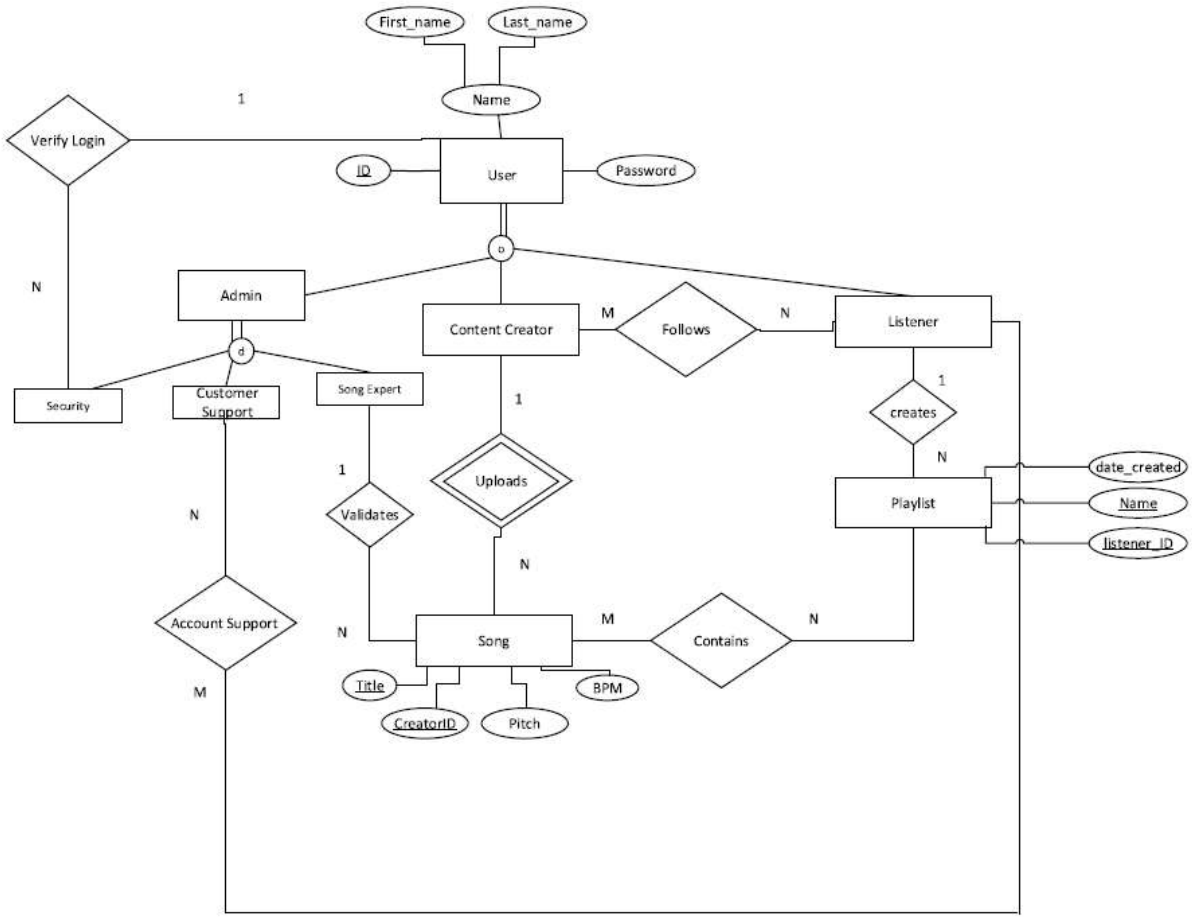
1) Contain data
    1.1) Account support data
    1.2) Admin account data
    1.3) Creator account data
    1.4) Customer support data
    1.5) Listener account data
    1.6) Playlist data
    1.7) Song data
    1.8) User account data

Non-Functional requirements:

1. Attributes in database should match listed attribute type

2. Database files should be named consistently for easy access

Conceptual Design of the Database



Data Dictionary

1. USER

- Attributes: ID (INT), First_name (CHARVAR(15)), Last_name (STRING), Password (CHARVAR(15))
- Primary Key: ID
- Foreign Key(s): None

2. LISTENER

- Attributes: ListenerID (INT)
- Primary Key: ListenerID
- Foreign Key(s): ListenerID references USER(ID) ON DELETE CASCADE, ON UPDATE CASCADE

3. CONTENT CREATOR

- Attributes: CreatorID (INT)

- Primary Key: CreatorID
- Foreign Key(s): CreatorID references USER(ID) ON DELETE CASCADE, ON UPDATE CASCADE

4. ADMIN
- Attributes: AdminID (INT)
- Primary Key: AdminID
- Foreign Key(s): AdminID ON DELETE SET NULL, ON UPDATE CASCADE

5. SECURITY
- Attributes: SecurityID (INT), Verify_login (BOOLEAN)
- Primary Key: SecurityID
- Foreign Key(s): SecurityID ON DELETE SET NULL, ON UPDATE CASCADE

6. CUSTOMER SUPPORT
- Attributes: SupportID (INT)
- Primary Key: SupportID
- Foreign Key(s): SupportID ON DELETE SET NULL, ON UPDATE CASCADE

7. ACCOUNT SUPPORT
- Attributes: SupportID (INT), ListenerID (INT)
- Primary Key: SupportID, ListenerID
- Foreign Key(s): SupportID, ListenerID ON DELETE SET NULL, ON UPDATE CASCADE

8. SONG EXPERT
- Attributes: ExpertID (INT)
- Primary Key: ExpertID

9. SONG
- Attributes: Title (CHARVAR(15)), Pitch (CHARVAR(2)), BPM (INT)
- Primary Key: Title
- Foreign Key(s): CreatorID ON DELETE CASCADE, ON UPDATE CASCADE

10. PLAYLIST
- Attributes: Name (CHARVAR(15)), Date_created (DATE)
- Primary Key: Name
- Foreign Key(s): ListenerID ON DELETE CASCADE, ON UPDATE CASCADE

11. CONTAINS
- Attributes: Title, CreatorID, PlaylistName
- Primary Key: Title, CreatorID, PlaylistName
- Foreign Key(s): Title ON DELETE CASCADE, ON UPDATE CASCADE; CreatorID ON DELETE CASCADE, ON UPDATE CASCADE; PlaylistName ON DELETE CASCADE, ON UPDATE CASCADE

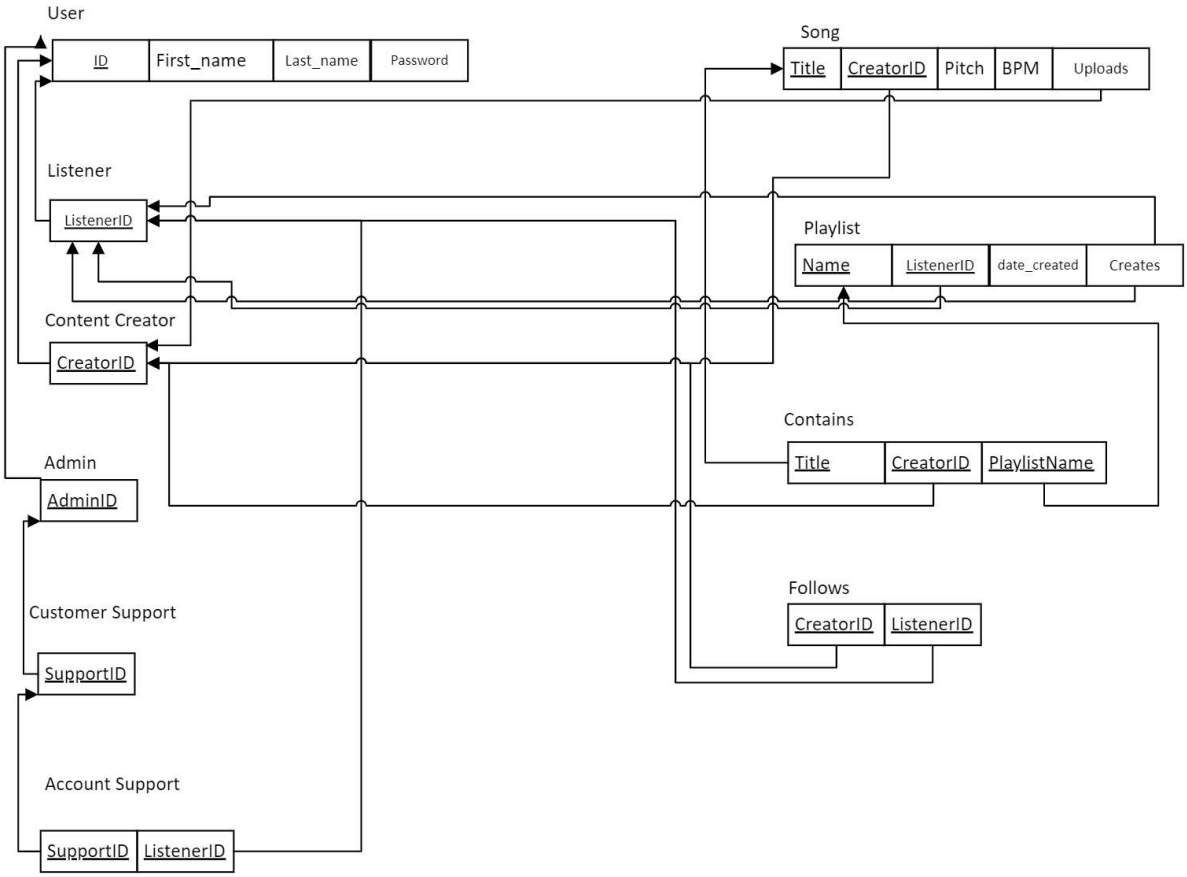
12. FOLLOWS

- Attributes: CreatorID, ListenerID
- Primary Key: CreatorID, ListenerID
- Foreign Key(s): CreatorID ON DELETE CASCADE, ON UPDATE CASCADE; ListenerID ON DELETE CASCADE, ON UPDATE CASCADE

Business Rules (Constraints)

1. Each user has a unique ID for their account.
2. Passwords cannot be NULL and are associated with user IDs for login.
3. Listener and Content Creator entities are linked to their respective playlists and songs using unique IDs.
4. Admin, Security, Customer Support, Account Support, and Song Expert entities have unique IDs for tracking purposes.
5. Verify_login in the Security entity is a boolean attribute to check user credentials' correctness.
6. Playlists and Songs are identified by their names and titles, respectively.
7. Songs have additional attributes such as pitch and BPM.
8. Foreign key constraints are set to cascade actions on delete and update for referential integrity.

Logical Database Schema



The SQL statements used to create the database are as follows:

```
use project4347;

CREATE TABLE user(
ID char(6) primary key not null,
firstName varchar(15) ,
lastName varchar(15),
password varchar(15)
);

CREATE TABLE listener (
  listenerID char(6)  NOT NULL,
  PRIMARY KEY (listenerID),
  CONSTRAINT `ListenerId`
      FOREIGN KEY (`listenerID`)
      REFERENCES user (`ID`)
      ON DELETE CASCADE
```

```
      ON UPDATE CASCADE
      );

CREATE TABLE creator (
  creatorID char(6) NOT NULL,
  PRIMARY KEY (`creatorID`),
  CONSTRAINT `CreatorIDFK`
      FOREIGN KEY (`creatorID`)
      REFERENCES `user` (`id`)
      ON DELETE CASCADE
      ON UPDATE CASCADE
      );

CREATE TABLE admin (
  adminID char(6) NOT NULL,
  PRIMARY KEY (`adminID`),
   CONSTRAINT `AdminIDFK`
      FOREIGN KEY (`adminID`)
      REFERENCES `user` (`id`)
      ON DELETE CASCADE
      ON UPDATE CASCADE
      );

CREATE TABLE customerSupport (
  supportID char(6) NOT NULL,
  PRIMARY KEY (SupportID),
  CONSTRAINT `CustSupportIDFK`
      FOREIGN KEY (`supportID`)
      REFERENCES `admin` (`adminID`)
      ON DELETE CASCADE
      ON UPDATE CASCADE
      );

CREATE TABLE accountSupport (
  supportID char(6) NOT NULL,
  listenerID char(6) NOT NULL,
  PRIMARY KEY (`supportID`),
  CONSTRAINT `accSupportIDFK`
      FOREIGN KEY (`supportID`)
      REFERENCES `customerSupport` (`supportID`)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  CONSTRAINT `accSuppListenerIDFK`
      FOREIGN KEY (`listenerID`)
      REFERENCES `listener` (`listenerID`)
```

```
      ON DELETE CASCADE
      ON UPDATE CASCADE
      );




CREATE TABLE song (
    title         VARCHAR(30)    NOT NULL,
    creatorID    char(6)    NOT NULL,
    pitch         INT        NOT NULL,
    BPM     INT        NOT NULL,
    CONSTRAINT SONGPK
       PRIMARY KEY (`title`, `creatorID`),
    CONSTRAINT SONGTFK
       FOREIGN KEY (`creatorID`) REFERENCES `creator`(`creatorID`)
            ON DELETE CASCADE
           ON UPDATE CASCADE
           );

CREATE TABLE playlist (
    `name`        VARCHAR(30)    NOT NULL,
    listenerID    char(6)    NOT NULL,
    dateCreated    VARCHAR(15)    NOT NULL,
    CONSTRAINT PLAYLISTSONGPK
       PRIMARY KEY (`name`, `listenerID`),
    CONSTRAINT PLAYLISTFK
       FOREIGN KEY (`listenerID`) REFERENCES `listener`(`listenerID`)
       ON DELETE CASCADE
      ON UPDATE CASCADE,
    CONSTRAINT PLAYNAMEFK
       FOREIGN KEY (`name`) REFERENCES `song`(`title`)
            ON DELETE CASCADE
    );


CREATE TABLE contains (
 title VARCHAR(30) NOT NULL,
 creatorID char(6) NOT NULL,
 playlistName VARCHAR(30) NOT NULL,
  CONSTRAINT CONTAINSPK
    PRIMARY KEY (`title`, `creatorID`, `playlistName`),
  CONSTRAINT TITLEFK
    FOREIGN KEY (`title`) REFERENCES `song`(`title`)
```

```
    ON DELETE CASCADE,
  CONSTRAINT CONTAINSCREATORIDFK
    FOREIGN KEY (`creatorID`) REFERENCES `creator`(`creatorID`)
    ON DELETE CASCADE
      ON UPDATE CASCADE,
  CONSTRAINT PLAYLISTNAMEFK
    FOREIGN KEY (`playListName`) REFERENCES `playlist`(`name`)
    ON DELETE CASCADE
      ON UPDATE CASCADE
   );

CREATE TABLE follows (
creatorID char(6) NOT NULL,
listenerID char(6) NOT NULL,
CONSTRAINT FOLLOWSPK
    PRIMARY KEY (`creatorID`, `listenerID`),
CONSTRAINT FOLLOWSCREATIDFK
    FOREIGN KEY (`creatorID`) REFERENCES `creator`(`creatorID`)
    ON DELETE CASCADE
      ON UPDATE CASCADE,
CONSTRAINT FOLLOWSLISTIDFK
    FOREIGN KEY (`listenerID`) REFERENCES `listener`(`listenerID`)
      ON DELETE CASCADE
      ON UPDATE CASCADE

);
```

Functional Dependencies & Database Normalization

The Functional dependencies in our database are as follows:

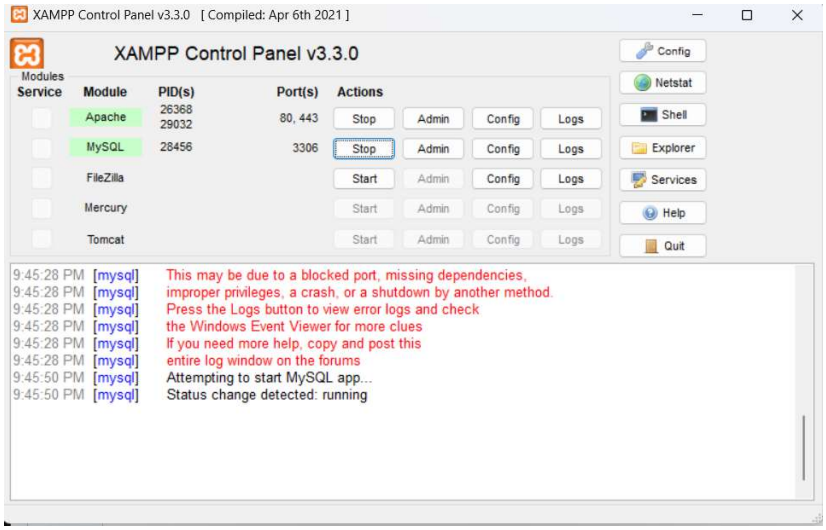
```
User Table:
ID -> FirstName, LastName
ID-> password

Playlist Table:
Name, listnerID -> dateCreated

Song Table:
Title, CreatorID -> pitch, BPM
```

A user's ID will determine their first and last name. Each user has a unique ID, so it is not possible for there to be a case where one ID is linked to two different names. The same goes for the password.
In the Playlist table, each listener must create a playlist with a unique name (only amongst their Playlist, not a name unique to all Playlist names in the database). Therefore, there is no possibility of there being a Playlist with the same name belonging to the same listener that was created on more than one date.

Finally, for the song table, each creator must upload a song with a unique name (amongst their song names, not a song name unique to the entire database). Therefore, there is no possibility of a song with the same song name and creator ID having more than one pitch and bpm.

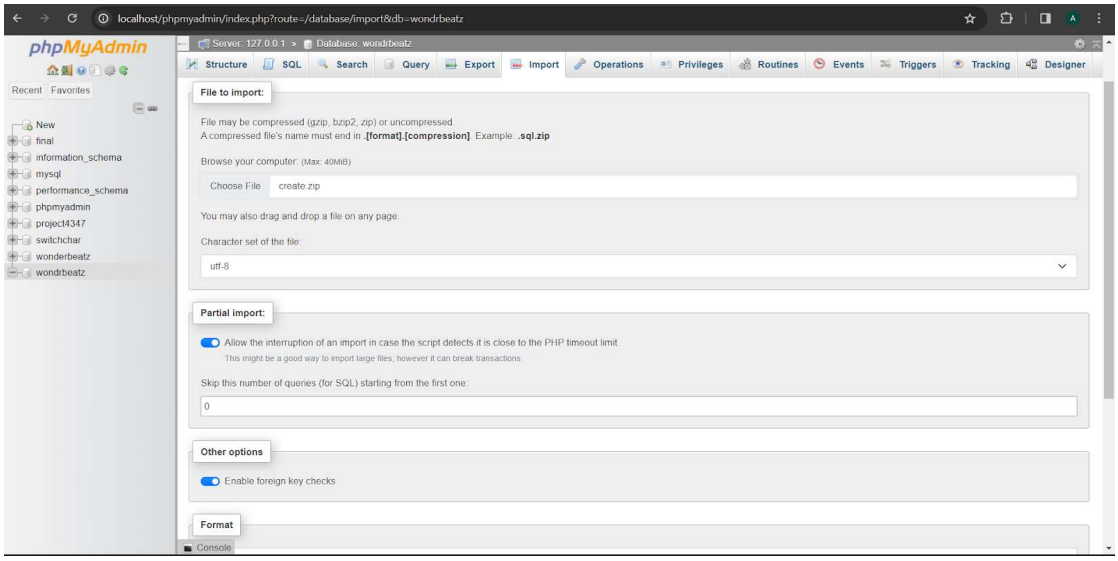
When we went into this phase of the project, we found that all of our tables were already in BCNF. Each of the functional dependencies in our table are directly linked to a candidate key, and they follow 1NF, 2NF, and 3NF as well. So it was not necessary for us to modify our tables or justify why we chose to keep a table in 3NF form.

The Database System



Place source file inside of C:\xampp\htdocs\project
Place data file inside C:\ProgramData\MySQL\MySQL Server 8.0\Uploads\Data

In web browser search for: localhost/dashboard
Select phpMyAdmin. Next, create a new database called WonderBeatz.
Once created, select database and on the top menu select import. Import the create.zip followed by the load.zip.



The WonderBeatz application is now installed and ready to run.

To access the login page, open a web browser and enter localhost/project/login.html

User Application Interface

The user interface is designed using html, connected to the WonderBeatz database via php to handle user queries, and some javascript to transfer smoothly between pages.
User accounts are either Creator or Listener type accounts, and each has different html pages to allow users to either upload songs as a creator, or add songs to a playlist as a listener. Users can submit information by filling in fields and pressing buttons.

Conclusions & Future Work

Conclusion

We achieved the majority of our goals for the project. The Listener and creator pages are fully functional and interact with each other. Meaning, if you update a song as a creator, a listener who has that song added to a Playlist will see that update. The login page is fully functional and will direct users who have accounts to their respective listener, creator, and admin accounts. The search page is operational, allowing listeners to filter through the full database of songs. And finally, every place that input is required, we used prepared statements to prevent SQL injections.

Feedback

This project was very helpful with furthering our understanding of sql and how to create an efficient database. Learning database systems through creating a database from scratch has allowed me to retain much more information than I would have if we had only done lectures, exams and quizzes. However, I felt our group's biggest hurdle was understanding and writing HTML code. All of us had minimal experience prior to the course. In the end, we were able to fix all of our bugs and errors, so I believe the project is doable as is, but I think it would be helpful to future students if some HTML resources were provided.

Future Work

With our Admin page, the goal was to have WonderBeatz administrators be available in some way to aid creator and listener users with any issues they may be having. We did fully draft out how exactly an admin would aid a user issue, and so the final page we created didn't have a lot to go off of. Right now, a user or creator ticket will appear on an Admin's account. But the ticket is blank and does not specify what the user's issue is. There is also no way for the admin to get back to the user.

To improve upon this, we could create a ticket table. The ticket table could have attributes such as userID, problem type, and problem description, and Admin ID. The admin ID could be initialized to null when the ticket is first created. Then, each admin will have access to the full ticket table. If a ticket has an admin ID that is null, that will mean an admin has not taken that ticket yet. Once an admin takes a ticket, the admin ID can be updated to that admins ID. Then, the admin can fix the issue and delete the ticket when resolved.

References List

To complete our project, we referenced:

1. Fundamentals of Database Systems, 7th Edition by Ramez Elmasri and Shamkant Navathe.
2. CS 4347.003 Course Lecture slides
3. Video Tutorial “How to Install a Local Server for PHP | 2023 | Learn PHP Full Course for Beginners” by Dani Krossing: https://www.youtube.com/watch?v=GRqw0pBwewY
4. HTML Tutorial from W3Schools: https://www.w3schools.com/html/
5. Stack Overflow: https://stackoverflow.com/

Appendix

See attached file.