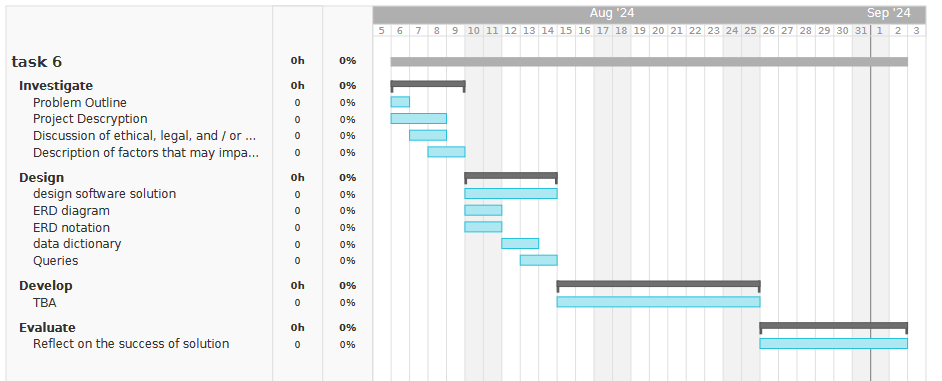
**Part 1:**

**Investigation:**

**Timeline:**



**Problem Outline:**

The problem outline is we need to create a music or video streaming service where users can somehow spend money for our products or services. This means we need a way to record all the music we have, the user’s information and their bank information, and all the purchases made on the platform.

**Problem Description:**

There will be scenarios where the user wants to view all the available music and buy the music with their money. The owner of the website will have scenarios where they will want to view today’s profits, overall profit, list of the number of each song sold, add music, remove music, and edit music.

The scenario where the user would want to view all the music available, we want to:

* Sort by alphabetical order, cost, length
* Show all music in a certain category such as genre, artist.
* In the database we will need an order, cost, duration, genre, and artist attribute for these features.
* In python we need a Get\_Song\_Records() function, Get\_Songs\_Records\_By\_Category(attribute, condition) for getting songs by category, Sort\_Records(attribute, descending (bool) ) for sorting the records.

The scenario where the user will buy the music, we want to:

* Get the users bank details if we don’t have it yet
* Check if the bank details are valid
* Create a receipt with details such as user, cost, song, and other details.
* add bought music to user’s library so they can download it.

The scenario where the owner will want to view profits:

* Create functions to list overall, yesterday’s, last week’s, last month’s, and last year’s profit.
* Create functions to show profit of a certain year.
* Show how much money a certain song or genre made.

The scenario where the owner will add and remove songs:

* Have a simple interface that’s makes it easy to add, remove, and modify music.
* Check if added or modified music follows domain integrity.
* We should also add the ability to add a genre and artist.
* Add\_Song(attr1, attr2, attr3…)
* Remove\_Song(songID)
* Modify\_Song(songID, attr, value)

**Ethical, legal, and/or security issues:**

Because we’ll be storing personal data, once we start making 3 million dollars profit, we will need to follow the Australia Privacy Act. We must be open about how we manage personal information. We must give users the ability to be anonymous or allowed to use a pseudonym. Only collect sensitive information when necessary. We must outline what personal information we need and why we need it. We can’t sell the information. The information must have currency, authenticity, relevance, and accuracy.

If we collect user bank details, unless allowed by the user, we will have to delete the data (but the details will still be on the receipt just in case we need to review it at a later date).

For security, data will be encrypted, the server storing the data will not be allowed to send data outside a network, only to the front-end server. The front-end will have implementations that prevents exploits that gives the user access to the database (SQL Injections, backdoors).

For ethics, we will donate some of our profit to charity. This will build up our reputation. We should give a relatively big percentage of revenue from selling a song to the original artist. We should also promise the user to never take away their ownership any of the music they bought and let him download it any time. We should also allow the user to refund any purchases (with limitations so they can’t exploit the system).

**Factors that may impact quality:**

User input can impact the quality of our database. We can always add constraints like having field must be not null, be a certain type of format, and be a certain length. But it doesn’t stop people from inputting misinformation. Of course, people can put in pseudonyms for personal information, but problems arise when users input fake emails and bank details as it would break our program and impact authenticity and accuracy. User input also includes the owner adding music to their platform. We could always automate song data with metadata but even then, metadata can also be inaccurate. Theres the chance the user may change bank details affecting the currency of our data, also artists might want to take down their music affecting the currency as well. We also must consider the relevance data has. There is no reason for us to get the date of birth of our users unless we want to analyze what type of music certain age groups listen to.

**Design:**

**ERD Diagram:**

**A diagram of a computer

Description automatically generated**

My process of making it is in Appendix 1.

**Relational notation:**

Song = <SongID, Title, ReleaseDate, Description, Cost, ArtistID, AlbumID, GenreID, MusicFileID>

Artist = <ArtistID, Name>

Album = <AlbumID, Title>

Genre = <GenreID, Title>

MusicFile = <MusicFileID, FileName, Duration, FileSize, SampleRate>

User = <UserID, UserName, Password, Email, BankDetailsID>

BankDetails = <BankDetailsID, BankNumber>

Reciept = <RecieptID, OwnedMusicID, PurchaseDate, Discount, TotalCost>

OwnedMusic = <OwnedMusicID, UserID, SongID>

**Data dictionary:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Size | Constraints | Description | Relationships |
| SongID | Integer | Inf | Primary Key,  Auto Increment,  Not null,  Unique | The unique identifier of the Song entity | Foreign key of Owned Music entity |
| Song Title | Text | 50 characters | Not Null | The song name |  |
| Song ReleaseDate | Date | YYYY/MM/DD | Not Null | The date the song released |  |
| Song Description | Text | 200 characters |  | Description of song |  |
| Song Cost | Float | 2 decimal places only | Not Null | How much the song cost to buy |  |
| ArtistID | Integer | Inf | Primary Key,  Auto Increment,  Not null,  Unique | The unique identifier of Artist entity | Foreign key of Song entity |
| Artist Name | Text | 50 characters | Not Null | The name of the artist |  |
| AlbumID | Integer | Inf | Primary Key,  Auto Increment,  Not null,  Unique | The unique identifier of Album entity | Foreign key of Song entity |
| Album Title | Text | 50 characters | Not Null | Name of the album |  |
| GenreID | Integer | Inf | Primary Key,  Auto Increment,  Not null,  Unique | The unique identifier of Genre entity | Foreign key of Song entity |
| Genre Title | Text | 50 characters | Not Null | Name of the genre |  |
| MusicFileID | Integer | Inf | Primary Key,  Auto Increment,  Not null,  Unique | The unique identifier of MusicFile Entity | Foreign key of Song entity |
| MusicFile FileName | Text | 55 characters | Not Null,  Unique | The name of the file stored on the file server (including the extension) |  |
| MusicFile  Duration | Time | Max 40 minutes | Not Null | How long the song is |  |
| MusicFile  FileSize | Integer | Max 300mb | Not Null | The size of the file in bits |  |
| MusicFile  SampleRate | Integer | Max 44.1kHz | Not Null | The sample rate of the song in hertz. Sample rate determine the sound quality of a song. |  |
| UserID | Integer | Inf | Primary Key,  Auto Increment,  Not null,  Unique | The unique identifier of the user’s entity | Foreign key of owned music. |
| User  Username | Text | Max 50 characters | Not Null  Unique | The name of the user account |  |
| User  Password | Encrypted Text | Minimum 8 characters, 2 numbers and a symbol | Not Null | The user’s password |  |
| User  Email | Text | Must have an @ and a valid website at the end | Not Null  Unique | The user’s email |  |
| BankDetailID | Integer | Inf | Primary Key,  Auto Increment,  Not null,  Unique | The unique identifier of the BankDetails entity | Foreign key of the User entity. Can be NULL in User entity. |
| BankDetails  Bank number | Integer | However long a bank number is | Not Null,  Unique | The bank number |  |
| ReceiptID | Integer | Inf | Primary Key,  Auto Increment,  Not null,  Unique | The unique identifier of the Receipt entity |  |
| Receipt  PurchaseDate | Date | YYYY/MM/DD | Not Null | the date of purchase |  |
| Receipt  Discount | Integer | 0-100 Null=0 |  | the discount on the song. |  |
| Receipt  Total Cost | Float | 2 decimal places | Not Null | The total cost of the song after discount |  |
| OwnedMusicID | Integer | Inf | Primary Key,  Auto Increment,  Not null,  Unique | The unique identifier of the OwnedMusic entity. | Foreign Key of the Receipt entity. |

**SQL Queries:**

Some SQL queries we will need are:

* Creating the database.
* Create the Song, Artist, Album, Genre, MusicFile, User, BankDetails, OwnedMusic, and Reciept tables with their attributes and constraints.
* A query that inserts a new record in any of the tables.
* A query that gets all the songs, filter by artist or genre, and sort them alphabetically, by release date, or cost.
* Get user from username and password.
* Get all owned music with userID.
* Get sum of all totalcost from receipts filtered by date.
* Get sum of all totalcost grouped by songs.

**Part 2:**

**Develop:**

Look at python scripts

**Evaluate:**

**Success:**

I haven’t completely check off all the requirements of the program since some of them are not SQL based and requires actual records instead of sample ones (e.g. I need the actual music file to allow the user to download it, I need the actual bank details to check if the user can buy the song) but it a good demonstration of how you’d create a digital music store where you can buy songs, add new songs, and view profits.

* Sort by alphabetical order, cost, length.
  + The program succeeds doing all of these
* Show all music in a certain category such as genre, artist.
  + The program can filter out songs based on their attributes.
* In the database we will need an order, cost, duration, genre, and artist attribute for these features.
  + The song has all but the order attribute which is its own table called Receipt
* In python we need a Get\_Song\_Records() function, Get\_Songs\_Records\_By\_Category(attribute, condition) for getting songs by category, Sort\_Records(attribute, descending (bool) ) for sorting the records.
  + The program has all of those.
* Get the users bank details if we don’t have it yet.
  + The program doesn’t do this. I should only reference BankDetails instead of Users in the Receipt table but if I implement it know I’ll have to modify the BankDetailsID attribute in Users that is in thee OwnedMusic table.
* Check if the bank details are valid.
  + Not really possible unless I actually use real bank details.
* Create a receipt with details such as user, cost, song, and other details.
  + The program does this. But I should reference bank details instead of users.
* add bought music to user’s library so they can download it.
  + The program does this through the OwnedMusic table.
* Have a simple interface that’s makes it easy to add, remove, and modify music.
  + The program has this.
* Check if added or modified music follows domain integrity.
  + Added music function does this but not modified music.
* We should also add the ability to add a genre and artist.
  + The program does this.
* Add\_Song(attr1, attr2, attr3…)
  + This is a function.
* Remove\_Song(songID)
  + This is a function.
* Modify\_Song(songID, attr, value)
  + This is a function.

**ERD Changes:**

For the bank details table. I’ve changed it so the bank number can be null. This is to uphold the privacy principle of anonymity and pseudonymity since the user can do actions that doesn’t need his bank details.

Additionally, duration does not use the time datatype but instead the integer datatype since it’s more simpler the python program to read.

**Extra Features:**

Some extra features are a interface to execute all the functions. Another is user input validation before using the data in SQL code. This is so I don’t get an error in the program, but instead, a console print statements outline what specifically the error (e.g. if the username is too long the program will tell me the username is not valid rather than an ambiguous error from the execute() function.)

**Known Bugs/limitations:**

You can’t filter and sort songs at the same time.

If you delete a User record which is referenced in the Receipt table will have a delete anomaly which is not good since we need the Receipt record even if the user is deleted.

**Retrospective:**

What worked well was all of the tables are pretty optimized and there is little to no redundancies. Some functions such as GetOrAddRecord(), RemoveRecordWithAttribute(), ModifyRecordWithAttribute() are very versatile and could’ve been used in many concepts.

What didn’t work well was that executing SQLite code felt very archaic, maybe if I had more general SQL functions or even constants of SQL commands. I wanted to keep all of the SQL queries in one script which ended up with functions such as AddUser() and SignUp() that could be one function but I instead have them in separate scripts for the sake of readability. I should’ve used BankDetails and Songs as foreign keys rather than OwnedMusic for the Receipt table as if the user wants to delete their account the Receipt table will break.

**Appendix:**

**Appendix 1:**

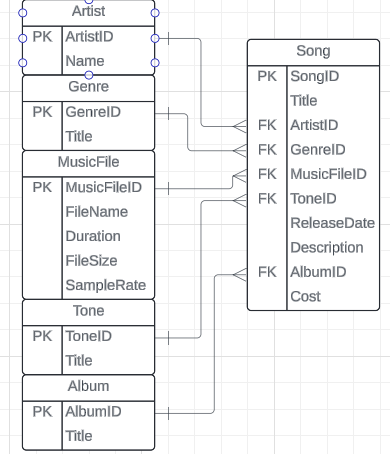
When making the song entity I realize Artist, Genre, Album, and Tone, FileName and FileSize are not dependent of the Primary Key so they all go in their separate tables in 2nd normal form. And Duration, while is dependent on SongID, can be found through, or is transitive to the MusicFile entity. So, duration goes in the MusicFile entity in 3rd normal form.

0NF:

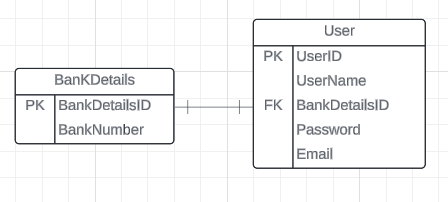
A screenshot of a music album

Description automatically generated

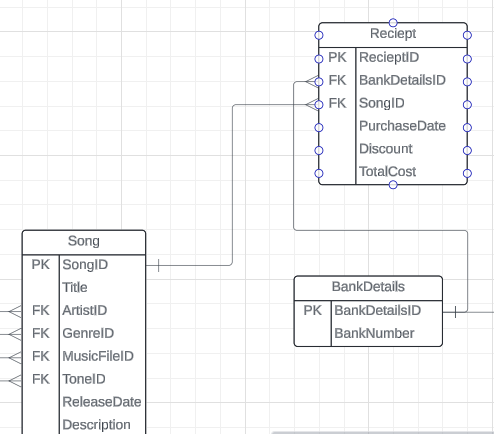
3NF:



For the User entity. We need a username, password, email, and bank details. Bank details are not dependent on the user, so we make a separate table for bank details. Only problem is that there’s a case that many users could have many bank details. But for this project, I’ll limit the field to only one bank account detail per user. And I’ll assume a bank account is used for only one user.



For Reciept entity. I’m just going to make each individual song purchased has a receipt to prevent many to many relationships.



To prevent many to many relationships for users and songs, a OwnedMusic composite key is created. One song can have many owned music, one user can have many owned music.

A screenshot of a computer

Description automatically generated

I changed some bits afterwards such as removing tone since it didn’t have that much purpose. And I replaced SongID to OwnedMusicID in the Receipt entity since I also wanted to have UserID in the Receipt entity.

**A diagram of a computer

Description automatically generated**

References:

<https://soundcharts.com/blog/music-metadata>

<https://i.imgur.com/qvMJzsP.jpeg>

<https://www.w3schools.com/sql/>

<https://www.digitalocean.com/community/tutorials/how-to-use-the-sqlite3-module-in-python-3>

<https://stackoverflow.com/>

<https://everynoise.com/>