

# iMusic: Music Record Collection Website

COMP1048, Databases and Interfaces: Coursework 2

Matthew Pike & Yuan Yao

## Table of contents

<b>Document Version History</b>	<b>2</b>
<b>Overview</b>	<b>2</b>
Database Schema . . . . .	3
<b>Tasks</b>	<b>5</b>
Task 1: Fix the Artist Table . . . . .	5
Task 2: Genre Statistics Page . . . . .	8
Task 3: Add New Genre and Assign Tracks . . . . .	11
<b>Submission</b>	<b>15</b>
Penalties . . . . .	15
Academic Misconduct . . . . .	15
<b>Acknowledgements</b>	<b>16</b>

## Document Version History

We will update the table below as the document is updated.

Version	Date	Author	Description
1.0	24 November 2023	Matthew Pike & Yuan Yao	Initial version for release.
1.1	25 November 2023	Matthew Pike & Yuan Yao	Updated Figure 2.
1.2	04 December 2023	Matthew Pike & Yuan Yao	Added hint for Task 3, under the “Track Assignment” subsection.

## Overview

### ! Fictional Scenario

Please be aware that this assignment is based on a fictitious scenario, and the company mentioned is not an actual entity. Additionally, there are no guarantees that the data provided is accurate or complete.

iMusic is a company that specialises in collecting and selling physical vinyl records. The company previously operated a physical store in the city center, but has recently decided to shift to an online-only business model. To reflect this change, they plan to update their website. They initially hired a professional developer to complete the work, but due to other commitments, the developer was unable to do so. Therefore, you have been hired to finish the project.

iMusic has provided you with a list of requirements for the website, which are outlined in the next sections. The website is partially implemented, and you are tasked with completing the implementation.

To complete the assignment, you only need to modify the `iMusic.py` file. The website makes use of the following technologies:

- [HTML](#) - Is used for structuring and presenting the content of the iMusic website. HTML 5 is used for the website.
- [CSS](#) - Is used for styling the content of the iMusic website. CSS 3 is used for the website.
- [Jinja](#) - Is a templating language for Python, which is used to generate HTML pages for the iMusic website. During the testing of your solution, the iMusic team will use the 3.0.3 version of Jinja.

- [Python 3](#) - Is a general-purpose programming language. During the testing of your solution, the iMusic team will use Python version **3.10 or later**. You are advised to use the latest version of Python to complete the assignment.
- [Flask](#) - A micro web application framework written in Python. During the testing of your solution, the iMusic team will use the 2.3.3 version of Flask.
- [SQLite](#) - A relational database management system. During the testing of your solution, the iMusic team will use the 3.44.0 version of SQLite.

**You are not permitted to use any other technologies or import any additional modules beyond those already imported in the `iMusic.py` file or those provided by the standard Python library.**

## Database Schema

The database for the project will store details of the vinyl records in the company's inventory. It consists of four tables:

- **Artist:** Stores the name of each artist. An artist is the person or group that performed the music on the record.
- **Album:** Stores the title of each album and links to the artist that created the album.
- **Genre:** A genre is a category that describes the type of music. This table stores the name of each available genre.
- **Track:** A track represents a single song on a record. Each track is linked to an album and a genre.

The Entity-Relationship diagram (ERD) in Figure 1 shows the relationships between the various tables in the database. The developer has used a slightly different notation than the one presented in the lectures. However, you should use your professional knowledge and experience to interpret the diagram and understand the relationships between the tables.

The database schema for the project is implemented in the `iMusic.db` file using the SQLite database engine. To complete the assignment, **you must use this pre-existing schema and may not alter it in any way.**

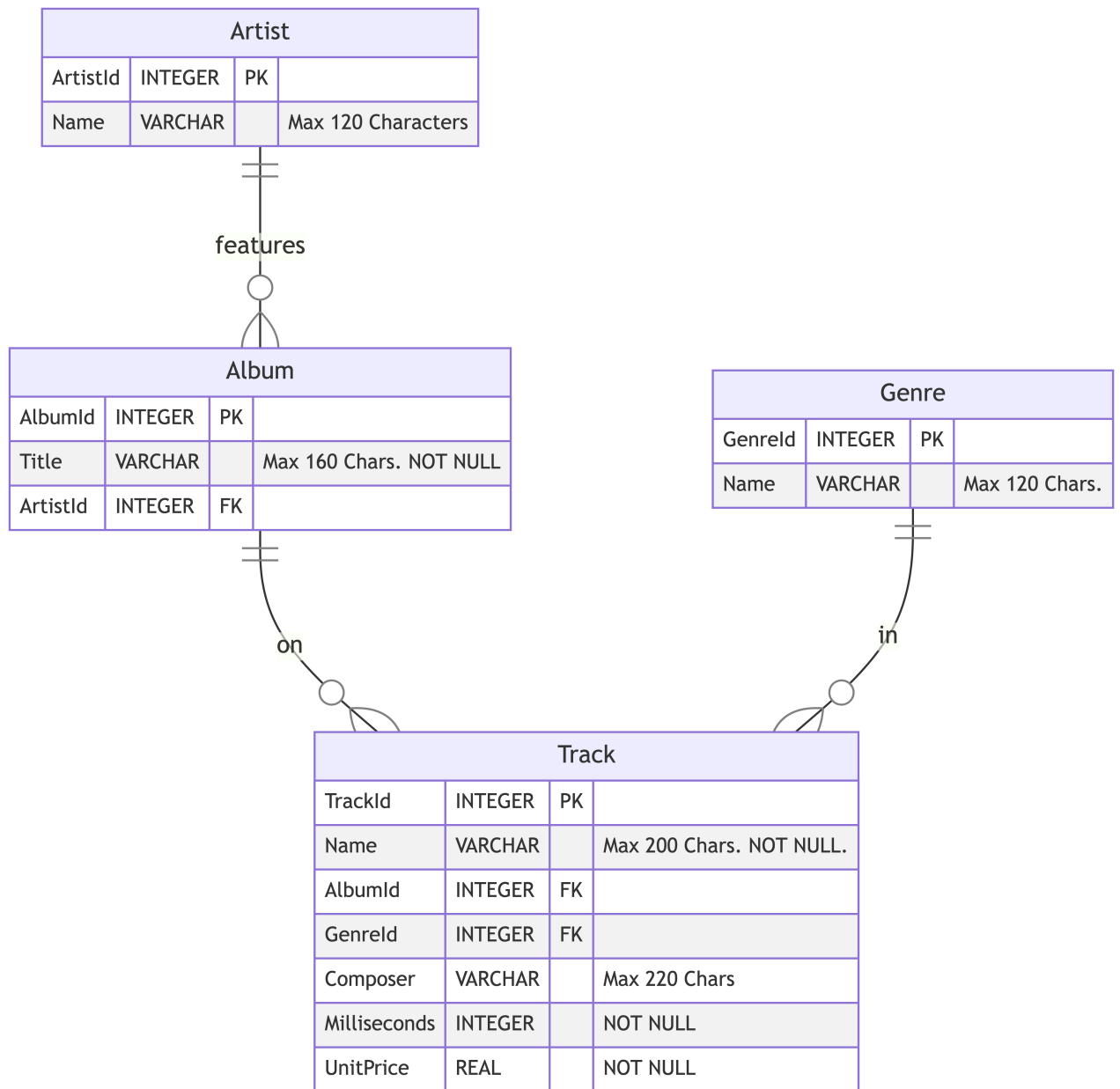


Figure 1: The iMusic record collection database schema

# Tasks

## Task 1: Fix the Artist Table

### Task Brief

In the process of developing the website, the developer made a mistake when inserting the name's of Artists into the database. There was an unusual bug in the code that caused some (but not all) names to be entered incorrectly. You have been asked to fix the mistake by updating the database with the correct names.

The previous developer managed to create the HTML template (`templates/upload.html`) for uploading a file, which contains the correct names of the artists, and added the URL `/upload/` to the Flask application (`iMusic.py`) to handle the upload. However, they were unable to complete the implementation of the update functionality.

The form on the `/upload/` page allows the user to upload a Tab-Separated Values (TSV) file containing the correct IDs and names of the artists. A TSV is a simple, plain text file that contains tab characters (`\t`) to separate columns of data. When the user uploads the TSV file, the existing code will save the file to the `uploads/` directory with the name `Artist.tsv`. The TSV file contains the following columns:

- **ArtistId** - The ID of the Artist. This is the primary key of the **Artist** table.
- **Name** - The correct name of the artist for the given ArtistId.

The TSV file contains one row for each artist to be inserted or updated in the database. Once the file has been uploaded, the existing code will call the `update_artist_table` function, passing the path to the uploaded file as an argument. Inside the `update_artist_table` function, you will need to read the file and update the database with the correct names. To parse the TSV file, you should use the `csv` module from the Python standard library. To learn how to use the `csv` module, you should read the [documentation](https://docs.python.org/3/library/csv.html) (<https://docs.python.org/3/library/csv.html>) - this is an expected part of the assignment.

Your task is to complete the implementation of the `update_artist_table` function. You must write the code that will read the file and update the database with the correct names. When implementing the `update_artist_table` function, you will need to consider the following:

- If the **ArtistID** already exists in the database, you must **update** the **Name** of the artist with the given **ArtistId** to the value provided in the TSV file.
- If the **ArtistID** does not exist in the database, you must **insert** a new artist with the given **ArtistId** and **Name** into the database.

Care is needed here - if you simply try to **INSERT** a new artist with the given **ArtistId** and **Name**, you will get an error because the **ArtistId** is the primary key of the **Artist** table (Uniqueness Constraint). There are a variety of ways to solve this problem, but the `iMusic`

team aren't sure which approach is best. You will need to decide how to solve this problem yourself. You may use any functionality supported by the SQLite version specified above.

There is no requirement to give feedback to the user about the success or failure of the operation, but you can if you'd like to. Further there is no requirement to handle errors or invalid data, but again, you can if you'd like to - it is good practice to do so. After the form is submitted and processed, you must redirect the user to the index page (/), regardless of whether the operation was successful or not.

When testing your implementation, the iMusic team will:

1. Visit the `/upload/` page.
2. Upload a TSV file containing the correct artist names.
3. Submit the form.
4. Check that the `Artist` table has been updated correctly, according to the contents of the TSV file.

The iMusic team will not test your solution with invalid data. A sample TSV file containing the correct artist names have been provided in the `data/` directory. You do not need to validate data in your solution or implement additional error checking or handling.

Your implementation must meet the following requirements:

- **Your implementation must be wholly contained within the `update_artist_table` function. You must not create any additional functions, or modify the template file in any way.**
- Your implementation must run as a Flask application, which will be run using the command `python iMusic.py` (or `python3 iMusic.py`).
- When interacting with the database, you must use the `sqlite3` module. You cannot use any other modules or approaches (e.g. SQLAlchemy).
- When reading the TSV file, you must use the `csv` module. You are expected to read the [documentation](#) for the `csv` module to learn how to use it. You cannot use any other modules or approaches (e.g. Pandas).

## Marking Criteria

Task 1 is worth 7 of the 25 marks available for the assignment. The following marking criteria will be used to assess your work:

ID	Requirement	Details	Marks
RQ1_1	Correct File Reading	Successfully reads the TSV file using the <code>csv</code> module, correctly parsing the <code>ArtistId</code> and <code>Name</code> .	1

ID	Requirement	Details	Marks
RQ1_2	Database Connection	Establishes a connection to the SQLite database using the <code>sqlite3</code> module.	1
RQ1_3	Correct Data Handling	Correctly identifies whether an <code>ArtistId</code> exists in the database and decides to update or insert data.	1
RQ1_4	Accurate Database Update	Accurately updates existing records in the database with the correct artist names.	1.5
RQ1_5	Accurate Database Insertion	Correctly inserts new records into the database where <code>ArtistId</code> does not exist.	1.5
RQ1_6	Correct Redirection	Correctly redirects the user to the index page (‘/’) after the form is submitted.	1

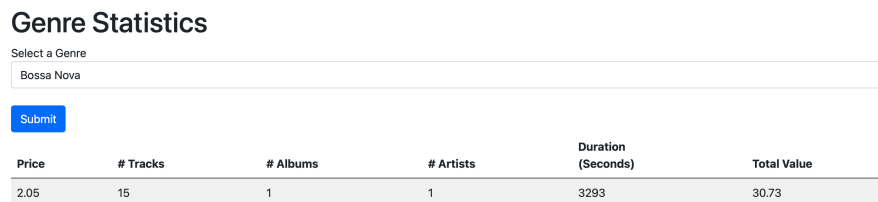
Table 2: Marking Criteria for Task 1.

## Task 2: Genre Statistics Page

### Task Brief

iMusic want an overview of the genres of music in their collection and the statistics associated with each genre. They have asked you to finish the implementation of the “Genre Statistics” page, for which the previous developer has already created a template (`templates/statistics.html`). A screenshot of the page is shown in Figure 2. The page contains a dropdown menu and a table. The dropdown menu is populated with the names of all genres in the database, in ascending order. The table should display the following, for the selected genre:

- **Price** - The average price of all tracks that have the selected Genre. The price should be rounded to two decimal places.
- **# Tracks** - The number of tracks that have the selected Genre.
- **# Albums** - The number of albums with at least one track in the selected Genre.
- **# Artists** - The number of artists that have at least one track with the selected Genre.
- **Duration** - The total duration of all tracks that have the selected Genre. The duration should be displayed in **seconds**, rounded to the nearest integer.
- **Total Value** - The total value of all tracks that have the selected Genre. The value should be rounded two decimal places.



The screenshot shows a web form titled "Genre Statistics". It has a dropdown menu labeled "Select a Genre" with "Bossa Nova" selected. Below the dropdown is a blue "Submit" button. Underneath the button is a table with six columns: Price, # Tracks, # Albums, # Artists, Duration (Seconds), and Total Value. The table contains one row of data for the selected genre.

Price	# Tracks	# Albums	# Artists	Duration (Seconds)	Total Value
2.05	15	1	1	3293	30.73

Figure 2: A screenshot of an example output for Task 2.

Users should be able to access the page by navigating to the `/statistics/` URL, followed by the `id` of the selected genre. For example, to view the statistics for the genre with `GenreId` 1, the user would visit the URL `/statistics/1/`. Note, that there is a special case where the user can visit the URL `/statistics/all/` to view the statistics for all genres in the database. The `all` case is not a genre in the database, but it should be handled correctly by your code. When `all` is selected, the table should display the statistics for all genres in the database.

When displaying the statistics for the specified genre, your code should render the `statistics.html` template. The template specifies a table, which should be populated with the statistics for the selected genre, as described above. The table will only ever contain one row.



The statistics displayed in the table will depend on the **GenreId** provided in the URL. If the specified **GenreId** does not exist in the database, then you should redirect the user to the statistics page (**/statistics/**) and display the error message: “The specified genre does not exist”. This error message should be displayed using the **flash** function and should be displayed in the **warning** style specified. An example of setting the display style (**success**, **danger**, **warning** among others) is shown below in the existing **iMusic.py** source code. Remember, when implementing this functionality, you also need to handle the special **all** case.

The functionality should be implemented in the file **iMusic.py**. The previous developer has already implemented the template for the page in the file **templates/statistics.html**, which you must not modify. When implementing the functionality, **you will only need to modify the following functions** in **iMusic.py**:

- **get\_genres()** - Retrieves a list of all genres from the database, and the special **all** genre, sorted in ascending order by name.
- **get\_genre\_statistics(genre\_id)** - Returns a list containing the statistics for the selected genre. The function will need to return values as specified in the template file. Must handle the special **all** case.
- **statistics()**, **statistics\_genre(genre\_id)** and **statistics\_process()** routing functions - These functions handle the routing for the **/statistics/** URL. It’s likely that you’ll only need to modify **statistics\_process()**.

## Marking Criteria

Task 2 is worth 8 of the 25 marks available for the assignment. The following marking criteria will be used to assess your work:

ID	Requirement	Details	Marks
RQ2_1	Genre Data Retrieval	Correct implementation of <b>get_genres()</b> to retrieve and return a list of all genres in ascending order. Includes the special ‘all’ case.	1
RQ2_2	Genre Statistics Calculation	Accurate implementation of <b>get_genre_statistics(genre_id)</b> to calculate and return the correct statistics for a selected genre.	3
RQ2_3	Handling of Special ‘All’ Case	Correct handling of the <b>/statistics/all/</b> URL to display statistics for all genres.	2
RQ2_4	Error Handling for Non-existent Genres	Proper error handling and user redirection with an appropriate message when a non-existent genre ID is requested.	1

ID	Requirement	Details	Marks
RQ2_5	Error-Free Code Execution	The code runs without errors, handling web requests and database operations correctly.	1

Table 3: Marking Criteria for Task 2.

### Task 3: Add New Genre and Assign Tracks

iMusic is looking to diversify its collection by adding new musical genres and assigning tracks to these genres. Your task is to create a feature that allows the insertion of new genres into the Genre table and the assignment of tracks to these genres. This feature will be accessible through the `/add/` URL.

The `/add/` page presents a form for entering the name of a new genre and selecting tracks to be associated with this genre. The form has already been implemented in the `templates/add.html` template. You must complete the implementation of this page by writing the Python code that will handle the insertion of the new genre into the database and the assignment of selected tracks to this genre.

Your implementation should include the following functionalities:

#### Populate the Tracks

When the `/add/` page is accessed, the select element in the form should be populated with the IDs and names of all tracks in the database without a genre, with items ordered by track name in ascending order. The track IDs and names should be retrieved from the database using the `Track` table. Tracks that already have a genre should not be included in the form. **This functionality should be implemented in the `get_tracks_with_no_genre()` function**, which should return a list of tuples containing the track IDs and names of all tracks without a genre.

#### Adding the New Genre

When a new genre name is submitted, your implementation must validate the name before inserting it into the database. You should only insert the new genre, iff:

- The genre name does not already exist in the database (uniqueness constraint).
  - Note, that your solution should be case sensitive, that is, “Rock” and “rock” are considered different genres.
- The genre name contains **at least three characters and less than or equal to 120 characters**. There are no limitations to the types of characters that can be used in the genre name.

If any of these conditions are not met, your code should not insert the genre into the database. Instead, it should redirect the user to the `/add/` page with an error message informing the user that: “Problem with the provided genre name.”. This error message should be displayed using the `flash` function and should be displayed using the `warning` style. **You should modify**

the `add_process()` and `add_genre_and_tracks(genre_name, track_ids)` functions to implement this functionality.

## Track Assignment

### Hint

So far, you've only retrieved single values from the form using the `request.form.get()` function. However, when the user submits the form, they will submit multiple values for the selected tracks. In this situation, you will need to use the `request.form.getlist()` function to retrieve all of the selected track IDs. Remember, using the documentation and researching how to address problems you encounter is an expected part of the assignment.

Along with the genre name, a list of track IDs **may** be submitted. **There is no requirement for any tracks to be selected.** That is, the user can submit the form without selecting any tracks. Your code must update these tracks in the `Track` table to associate them with the newly added genre. Track assignment should only proceed if:

- The genre name is valid, as per the conditions above.
- All submitted track IDs exist in the `Track` table and do not already have a genre.

If any of these conditions are not met, your code should not update the tracks in the database. Instead, it should redirect the user to the `/add/` page with an error message informing the user that: “The provided track IDs are invalid”. This error message should be displayed using the `flash` function and should be displayed in the `warning` style specified. **Again, you will need to modify the `add_process()` and `add_genre_and_tracks(genre_name, track_ids)` functions to implement this functionality.**

## All or Nothing Approach

Your code should use an “all or nothing” approach when inserting the new genre and updating the tracks. That is, if any of the conditions above are not met, your code should not insert the new genre or update any of the tracks. For example, if the user submits a genre name that already exists in the database, your code should not update any of the tracks (or insert the new genre). Similarly, if the user submits a valid genre name, but an invalid track ID(s), your code should not insert the new genre or update any of the tracks.

## Error Handling

Your solution should handle likely errors that may occur during the submission and processing of the form. Your code should be resilient to invalid inputs and operations. For all other

errors, except those specified above, your code should redirect the user to the `/add/` page with an error message informing the user that: “An error occurred while processing your request”. This error message should be displayed using the `flash` function and should be displayed with the `warning` style.

Here you will want to make use of the `try` and `except` statements to handle errors. You can read more about these statements in the [Python documentation](#).

### Redirecting the User After Successful Submission

After successful submission and processing, redirect the user to the index page (`/`) with a success message - “The genre has been added successfully”. This message should be displayed using the `flash` function, with the ‘success’ style.

### Marking Criteria

Task 3 is worth 10 of the 25 marks available for the assignment. The following marking criteria will be used to assess your work:

ID	Requirement	Details	Marks
RQ3_1	Form Population with Tracks	Correctly populating the form with track IDs and names from the database ordered by name, excluding tracks already assigned to a genre.	1
RQ3_2	Genre Name Validation	Properly validating the new genre name (uniqueness and character length constraints).	2
RQ3_3	Database Insertion of New Genre	Successfully inserting the new genre into the database if all validation checks pass.	1
RQ3_4	Track Assignment Validation	Correctly validating the track IDs submitted (existence in the database and no prior genre assignment).	1
RQ3_5	Database Update with Track Assignments	Accurately updating the database to associate selected tracks with the new genre.	2
RQ3_6	All or Nothing Transaction Implementation	Implementing the all or nothing approach correctly, ensuring either all changes are committed or none if an error occurs.	2
RQ3_7	Error Handling and User Feedback	Proper implementation of error handling and providing appropriate user feedback using the <code>flash</code> function.	1

ID	Requirement	Details	Marks
Table 4: Marking Criteria for Task 3.			

## Submission

To complete the assignment, you must submit an updated `iMusic.py` file containing your solutions to the tasks. No other files should be included in the submission. Ensure that your code is well commented and that you have included your name and student ID in the file. Additionally, ensure that you modify the functions specified in the assignment brief. You will need to submit your solution via Moodle by the deadline specified on the coursework issue sheet.

## Penalties

Table 5 shows the penalties that apply to this assignment.

Penalties	Details	Deduction
Late Submission	Standard university penalty policy.	5% absolute deduction, per working day.
Incorrect Filename	The submitted filename must be 'iMusic.py'.	10% absolute deduction
Use of Other Technologies	Only the technologies specified in the assignment brief and the python standard library are allowed.	100% absolute deduction

Table 5: Penalties applicable to submissions.

## Academic Misconduct

By submitting your work for assessment, you declare that the work is your own. Familiarise yourself with the [Academic Misconduct policy](#). Remember:

- **Code Sharing:** Do not share your code or specific approach for solving the task with others.
- **Code References:** Include references for any external code or resources used. Understand and be able to explain any referenced code.
- **Protect Your Work:** Ensure your work is secure and regularly backed up. Do not allow others access to your work or computer. Take care if you share a dormitory with other students.
- **Purpose of Assignment:** This assignment assesses your understanding and application of course material. Contact the module convenor if you have questions about what constitutes academic misconduct.

- **No LLMs:** Do not use Large Language Models (LLMs) like ChatGPT or CoPilot for code generation. This is considered academic misconduct.

```
# Example of a code reference:  
# Adapted from https://stackoverflow.com/a/12345678  
if x == 1:  
    print("x is 1")
```

## Acknowledgements

The data used in this assignment is based on the [Chinook Database](#). We'd like to thank Jane Zhao for their help reviewing this assignment.