Updating the iMusic Company Website

COMP1048, Databases and Interfaces: Coursework 2

Matthew Pike & Yuan Yao

Table of contents

Overview **2**
- Document Version History . . . 2
- Scenario . . . 2
- Database Schema . . . 3

Tasks **4**
- Task 1: Fix `Genre` Names . . . 4
  - Task Brief . . . 4
  - Marking Criteria . . . 4
- Task 2: Price Statistics Page . . . 5
  - Task Brief . . . 5
  - Marking Criteria . . . 7
- Task 3: Adding new Tracks . . . 8
  - Task Brief . . . 8
  - Marking Criteria . . . 11

Submission **12**
- Penalties . . . 12
- Academic Misconduct . . . 13

Acknowledgements **13**

Overview

Document Version History

Version	Date	Author	Description
1.0	05 December 2022	Matthew Pike & Yuan Yao	Initial version for release.
1.1	05 December 2022	Matthew Pike & Yuan Yao	For Task 3, Clarified `track_duration` should not accept numbers <=0.

Scenario

> **!** Fictional Scenario
>
> Please note that this assignment is based on a fictional scenario. The company is not real.

iMusic is a company that collects and sells physical vinyl records. Until recently, the company has been operating a physical shop in the city centre. However, due to the COVID-19 pandemic, the company has decided to close the shop and move to an online-only business model. The company wants to update their website accordingly. They hired a professional developer to do this, but unfortunately the developer has been unable to complete the work due to other commitments. You have been hired to complete the work.

The company has provided you with a list of requirements for the website, which are described in the next section. The website is partially implemented, and you will need to complete the implementation.

The website is implemented using the following technologies:

- HTML, CSS, and Jinja2 for the front-end
- Flask for the back-end
- SQLite for the database

You must use these technologies to complete the assignment. You must not use any other technologies.

Database Schema

The database is used to store the details of the vinyl records in the company's inventory. The database consists of four tables:

- `Artist` - Stores the name of each artist. An artist is the person or group that created 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 on the record. For example, the genre of a record might be "Rock" or "Pop". This table stores the name of each available genre.
- `Track` - Stores the details of each track on the record. Each track is linked to an album and a genre.

Figure 1 presents the Entity-Relationship (ER) diagram that shows the relationships between the tables. Note, the developer has used a slightly different notation to the one used in the lectures. You must use you professional knowledge and experience to interpret the diagram.

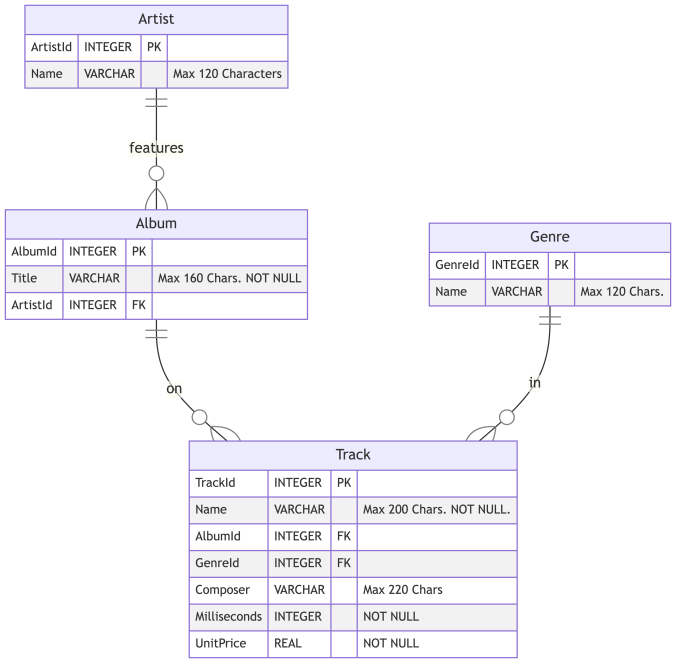

Figure 1: ER diagram of the database schema. Note: PK = Primary Key, FK = Foreign Key.

The database schema is implemented in the file `iMusic.db`. The database schema is implemented using the SQLite database engine. You must use the `iMusic.db` database schema to complete the assignment. You may not change the database schema.

Tasks

Task 1: Fix `Genre` Names

Task Brief

In the process of developing the website, the developer made a mistake when inserting the `Name`s of `Genre`s into the database. Instead of inserting the actual genre names, the developer instead inserted the word "Genre" for every genre in the database. You must fix this mistake by updating the database to contain the correct genre names.

The correct genre names are listed in `genres.csv`, a comma-separated values (CSV) file. The file contains a two columns of data:

- `genre_id` - The ID of the genre. This is the primary key of the `Genre` table.
- `genre_name` - The correct name of the genre for the given ID.

Each row in the file contains the correct genre name for the given genre ID. You must use this file to update the database. `genres.csv` will only ever include the genre IDs that are currently in the database. For this reason, you do not need to worry about inserting new genres into the database when writing your solution.

iMusic have asked you to write a Python script that will update the database. The script must be called `fix_genres.py`. There are specific requirements that your script must meet:

- The script must be able to be run from the command line using the command `python fix_genres.py` (or `python3 fix_genres.py`).
- The script must read the genre names from a file named `genres.csv`.
- The script should use the `sqlite3` module to connect to the database and update the genre names.
- The script should use the `csv` module to read the genre ids and names from the CSV file. You are expected to read the documentation for the `csv` module to learn how to use it.
- The script must not use any other modules.

Marking Criteria

Task 1 is worth 5 of the 25 marks available for the assignment. The following marking criteria will be used to assess your work:

Table 2: Marking Criteria for Task 1.

ID	Requirement	Details	Marks
RQ1_1	Your solution makes correct use of the `csv` module to parse the `genres.csv` file.	Your script must use the `csv` module to read the genre ids and names from the CSV file. You should not write code to parse the CSV file yourself, you should use the `csv` module to do this for you. You are expected to read the documentation for the `csv` module to learn how to use it.	2
RQ1_2	Your solution makes correct use of the `sqlite3` module to connect to the database and update all the genre names.	Your solution should not insert any additional or incorrect data, or marks will be deducted for this criteria.	2
RQ1_3	The script can be run from the command line without error.	Your script must be able to run without additional input or modification and handle any valid input files named `genres.csv`.	1

Task 2: Price Statistics Page

Task Brief

iMusic are keen to get an overview of the prices of the tracks in their collection. They have asked you to finish the implementation of the "Price Statistics" page. The page should display a table containing the following information:

- `Price` - Each unique `UnitPrice` value in the database. Each unique `UnitPrice` should only be listed once in the table.
- `# Tracks` - The number of tracks in the database that have the given `UnitPrice`.
- `# Albums` - The number of albums in the database that have at least one track with the given `UnitPrice`.
- `# Artists` - The number of artists in the database that have at least one track with the given `UnitPrice`.
- `Duration` - The total duration of all tracks in the database that have the given `UnitPrice`. The duration should be displayed in seconds format, rounded to the nearest integer.
- `Total Value` - The total value of all `Tracks` in the database that have the given `UnitPrice`. The value should be rounded two decimal places.

The table should be sorted by `Price` in ascending order. Additionally, your table should include a row at the bottom of the table that contains the following information:

- `Total` - The word "Total" should be displayed in this cell.
- `# Tracks` - An integer value representing the total number of tracks in the database.
- `# Albums` - An integer value representing the total number of albums in the database, with at least one `Track`.
- `# Artists` - An integer value representing the total number of artists in the database, with at least one `Album`.
- `Duration` - The total duration of all tracks in the database. The duration should be displayed in seconds format and rounded to the nearest second.
- `Total Value` - The total value of all tracks in the database. The value should be rounded to two decimal places.

The final row of the table should be calculated from all the data available in the database. You should not sum the values calculated for each `UnitPrice` (above), as this will introduce errors into the final result. An example of the output is shown in Figure 2.

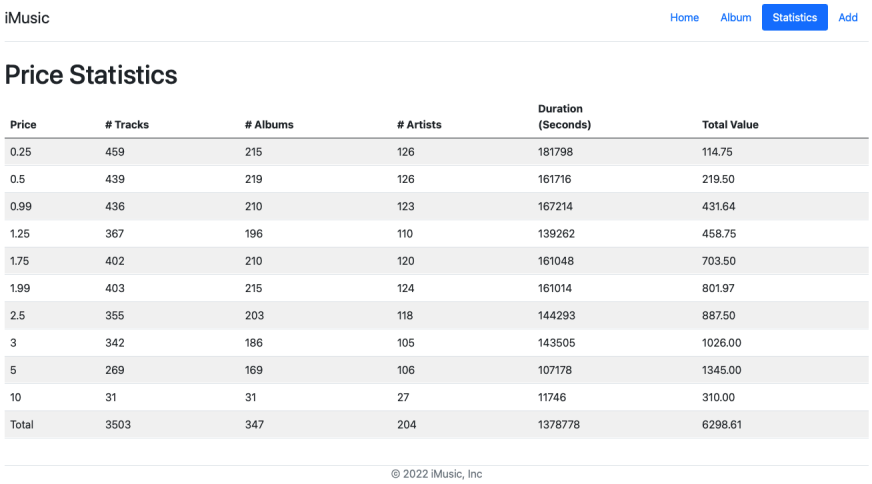

Figure 2: A screenshot of an example output.

Users should be able to access the page by navigating to the `/statistics/` URL. 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`. You must complete the implementation of the page by writing the Python code that will generate the table. You must not modify the template file.

Marking Criteria

Task 2 is worth 10 of the 25 marks available for the assignment. The following marking criteria will be used to assess your work:

Table 3: Marking Criteria for Task 2.

ID	Requirement	Details	Marks
RQ2_1	The page is accessible at the URL `/statistics/`	The page should be accessible at the URL `/statistics/`. If the page is not accessible at this URL, you will lose all marks for this question.	1
RQ2_2	The table contains all unique `UnitPrice` values in the database.	The table should contain a row for each unique `UnitPrice` value in the database.	1
RQ2_3	The table is sorted by `Price` in ascending order.	The table should be sorted by `Price` in ascending order.	1
RQ2_4	The table contains the correct number of `Track`s, `Album`s and `Artist`s for each `UnitPrice`.	The table should count the number of `Track`s, `Album`s and `Artist`s for each `UnitPrice`. This should be done using SQL queries.	3
RQ2_5	The table contains the correct `Duration` and `Total Value` for `Track`s at a given `UnitPrice`.	Each row in the table should contain the correct `Duration` (rounded to the nearest second) and `Total Value` (rounded to two decimal places) for `Track`s at a given `UnitPrice`. This should be done using SQL queries.	2
RQ2_6	The table contains the correct final row.	The final row of the table should contain the correct values for `# Tracks`, `# Albums`, `# Artists`, `Duration` (rounded to the nearest second) and `Total Value` (rounded to two decimal places). This should be done using SQL queries.	2

Task 3: Adding new Tracks

Task Brief

iMusic wants staff to be able to add new tracks to the database. A HTML form (`templates/add_track.html`) has already been implemented to allow users to enter the details of the new track.

The form has been designed to collect the following data:

- `track_name` - The name of the track to be added. This is a **required** field.
- `track_album` - The ID of the album that the track belongs to. This is a drop-down list that contains all the album **Title**s in the database. The user will select the title, but the ID of the album should be submitted with the form. This is a **required** field.
- `track_genre` - The ID of the genre that the track belongs to. This is a drop-down list that contains all the genre **Name**s in the database. The user will select the name, but the ID of the genre should be submitted with the form. This is a **required** field.
- `track_composer` - The name of the composer of the track. This is an **optional** field.
- `track_duration` - The duration of the track in seconds. This is a **required** field. Duration must be a positive integer value. If a floating point value is entered, you do not need to store the mantissa (the digits after the decimal dot) part of the value, just the exponent (the digits before the decimal dot).
- `track_price` - The price of the track. This is a **required** field. Prices must be a number greater than zero (0) and less than or equal to 10. Your system only needs to support prices to two decimal places. Your system will not be tested with prices containing more than two decimal places.

Figure 3: A screenshot of an example output. **Note** Your data may vary from that shown in this figure.

The form has been implemented using HTML and CSS, but is not yet connected to the Flask application. The form should be displayed when the user visits the URL `/add/` and should render the `templates/add_track.html` template, as shown in Figure 3. The form values will be submitted to `/add/track`. You will need to provide values for the `track_album` and `track_genre` drop-down lists. The values for the drop-down lists should be generated using SQL queries, with both being ordered by the `Title` and `Name` fields respectively, in ascending order. The form has been implemented and must not be changed.

In the `iMusic.py` file, you will need to implement the code that will process the form data and add the new track to the database. When processing the form data, you must perform the following checks:

Table 4: Form data validation for Task 3.

Field	Check	Error Message to Display
`track_name`	The field is empty.	`Track name is empty`
`track_name`	The field contains more than 200 characters.	`Track name is too long`
`track_album`	The field is empty or is not an integer number.	`Album ID is invalid`
`track_album`	The field contains a value that is not a valid album ID. That is: the album ID does not exist in the database table `Album`.	`Album does not exist`
`track_genre`	The field is empty or is not an integer number.	`Genre ID is invalid`
`track_genre`	The field contains a value that is not a valid genre ID. That is: the genre ID does not exist in the database table `Genre`.	`Genre does not exist`
`track_composer`	The field contains more than 220 characters.	`Composer name is too long`
`track_duration`	The field is empty or is not a number.	`Duration is invalid`
`track_duration`	The field contains a number less than or equal to zero (0).	`Duration must be a positive number`
`track_price`	The field is empty or is not a number.	`Price is invalid`

Field	Check	Error Message to Display
`track_price`	The field contains a value that is less than or equal to zero (0) or greater than 10. Your system only needs to support prices to two decimal places. Your system will not be tested with prices containing more than two decimal places.	`Price must be more than zero and less than or equal to 10`

Hints and Tips

1. Python provides a `len()` function that can be used to determine the length of a string. For example, `len("Hello")` will return the value `5`.
2. The previous developer has included a `is_number`, `is_integer`, `string_to_int` and `string_length` functions in the `iMusic.py` file. These functions may be useful when implementing the form data validation checks.

If the form data is not valid, you should not insert the new track into the database. Instead you should return a rendered version of the `error.html` template with all relevant error message, as specified in Table 4. All fields with errors should have their associated error messages be displayed using the `error.html` template, not just the first error message, as shown in Figure 4. Your error messages must appear exactly as shown in Table 4. The `error.html` template is already implemented. You do not need to make any changes to it. You should not implement any additional error checking. The URL of the error page does not matter. There is no requirement to preserve ("remember") the form data that the user entered if an error occurs.

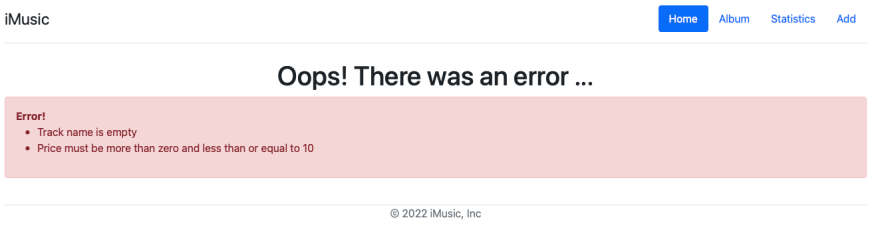

Figure 4: An example of an error page, displaying multiple error messages.

If the form data is valid, you should add the new track to the database. The new track must be added to the `Track` table. The `TrackId` of the new track must be automatically generated by the database. Once the new track has been added to the database, you must redirect the user to the root URL `/` (i.e. the home page).

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:

Table 5: Marking Criteria for Task 3.

ID	Requirement	Details	Marks
RQ3_1	The form is displayed when the user visits the URL `/add/` with values for the `track_album` and `track_genre` drop-down lists.	The form should be displayed when the user visits the URL `/add/` and should render the `templates/add_track.html` template.	1
RQ3_2	Valid data is added to the database.	If the form data is valid, it must be added to the database.	2
RQ3_3	Form data is validated according to the criteria in Table 4.	You must implement the validation procedure on the server-side. You must not rely on the client-side validation.	2
RQ3_4	Submissions whose fields do not meet the validation rules should return an error message.	All fields with invalid data should have their associated error messages be displayed using the `error.html` template.	2
RQ3_5	After inserting the new track, the user is redirected to the root URL.	The root url is the URL `/` and should render the `templates/index.html` template.	1
RQ3_6	Your solution should make use of `try`/`except` blocks to handle errors.	Your solution should be reliable and robust. It should not crash if the user enters invalid data or if the database is not available. You should consider a range of possible errors and implement appropriate error handling.	2

Submission

You should submit a single zip file containing the following files:

- `fix_genres.py`
- `iMusic.py`

No other files should be included in the zip file. Your ZIP archive should be named `<your student id>.zip`, where `<your student id>` is your student ID number. For example, if your student ID number is `2012345`, you should name your submission: `2012345.zip`. Your student ID is (typically) an 8-digit number that begins with the number `20`.

Your submission should be archived using the zip file format. You should not use any other file format (such as .rar, .7z, .tar.gz, etc.). You should not use any compression options when creating your zip file. You should not include any directories in your zip file. All files should be at the top level of the zip file.

You should submit your solution via the appropriate Moodle assignment. The deadline for submitting is specified on the coursework issue sheet.

Penalties

Table 6 shows the penalties that apply to this assignment.

Table 6: Penalties applicable to submissions.

Penalties	Details	Deduction
Late Submission	If you submit your assignment after the deadline, you will be penalized according to the standard university penalty	5% absolute deduction, per day
Incorrect Filename	If you submit your assignment with an incorrect filename.	10% absolute deduction
Incorrect File Format	If you submit your assignment with an incorrect file format.	10% absolute deduction
Use of Other Technologies	If you use technologies other than those specified in the assignment brief.	100% absolute deduction

Academic Misconduct

You are reminded that, by submitting your work for assessment, you are declaring that the work is your own. Please read the Academic Misconduct policy for more information. Remember:

- You are not allowed to share your code or your specific approach for solving the task.
- You should include references to any code or resources you have used in completing the assignment. You should include the references directly in your code. For example:

```
# The following code was adapted from https://stackoverflow.com/a/12345678
if x == 1:
    print("x is 1")
```

- It is your responsibility to protect your work. Do not allow others to access your computer.

Acknowledgements

The data used in this assignment is based on the Chinook Database. We'd like to thank Jane Zhao and Renjie Wu for their help reviewing this assignment.