CIS4301 Library Database Project

Overview

Most projects require a database, whether it's a .csv file, .json file, plain text, or even a data structure stored entirely in memory. You may be familiar with these options and know that they aren't always easy to work with, and don't scale well with a lot of data. Using an SQL database would be the ideal solution, but up until now we've mainly had you interact with databases through the terminal. This method is fine for a single user that doesn't have to run many queries with much data, but it isn't really applicable when writing an application that has to interact with the database. This assignment is intended to familiarize you with working with an SQL database in your code, and everything that comes with it.

We have given you a real world example of creating the backend for a library management application. This is a system that will be used by the librarians to manage and keep track of their books, users, and loans. The frontend is a CLI written in Python that is provided for you, so you only have to worry about writing the SQL queries and any logic that is needed to complete them in the backend. To streamline the process we have provided you with the `db_handler.py` file, that contains function definitions along with docstrings that discuss the parameters and return values. To complete this assignment, you must write the logic for all the functions provided, and any additional functions you may find useful. **All of your python code should be contained in the `db_handler.py` file, otherwise your work will not be graded properly.**

Additionally, the schema we provide is cursory and incomplete. To flesh out the DBMS, you are to write and submit a few SQL queries in a separate file named `project.pdf`. See the **SQL Queries** section for more information.

Setup

Steps 1 and 2 should've already been completed at the start of the semester during the MariaDB setup session. To set up your project workspace, open up Terminal on Mac or PowerShell on Windows and follow the steps below.

1. Ensure that MariaDB is set up and running on your computer
2. Ensure that you have the Python MariaDB package installed

3. Download the project repo from GitHub by running: `git clone https://github.com/Aelly-A/CIS4301-Project.git`
4. Now you can either open the `CIS4301-Project` directory in your IDE, or in the command line by running: `cd CIS4301-Project`
5. Open the `MARIADB_CREDS.py` file and update the value for the username and password fields
    - If you aren't using the default port (3306), update that as well. You can run `SHOW GLOBAL VARIABLES LIKE 'PORT';` in the MariaDB terminal to see what port you're running on
6. Open the `db_handler.py` file and update the UFID and FULLNAME variable to contain your information
7. Run the `load_db.py` file and follow the instructions on screen by running: `python3 load_db.py`
    - If `python3` doesn't work in your terminal you may need to use `python`
    - If you get a collation error, open up the `load_db.py` file and read the comment on line 5
8. Start the frontend by running `python3 main.py`

Once the data is loaded into your database, you should be able to run `main.py` and have a working frontend. This will allow you to test out your backend changes as you develop. If `main.py` was started successfully then you are good to start working on the project, otherwise reach out to a TA to receive help.

ER Diagram and Data Schema

In Figure 1 you will find the database schema represented with an ER diagram. Additionally, the tables below show the structure of the tables provided for you. The primary key of each table is bolded and underlined.

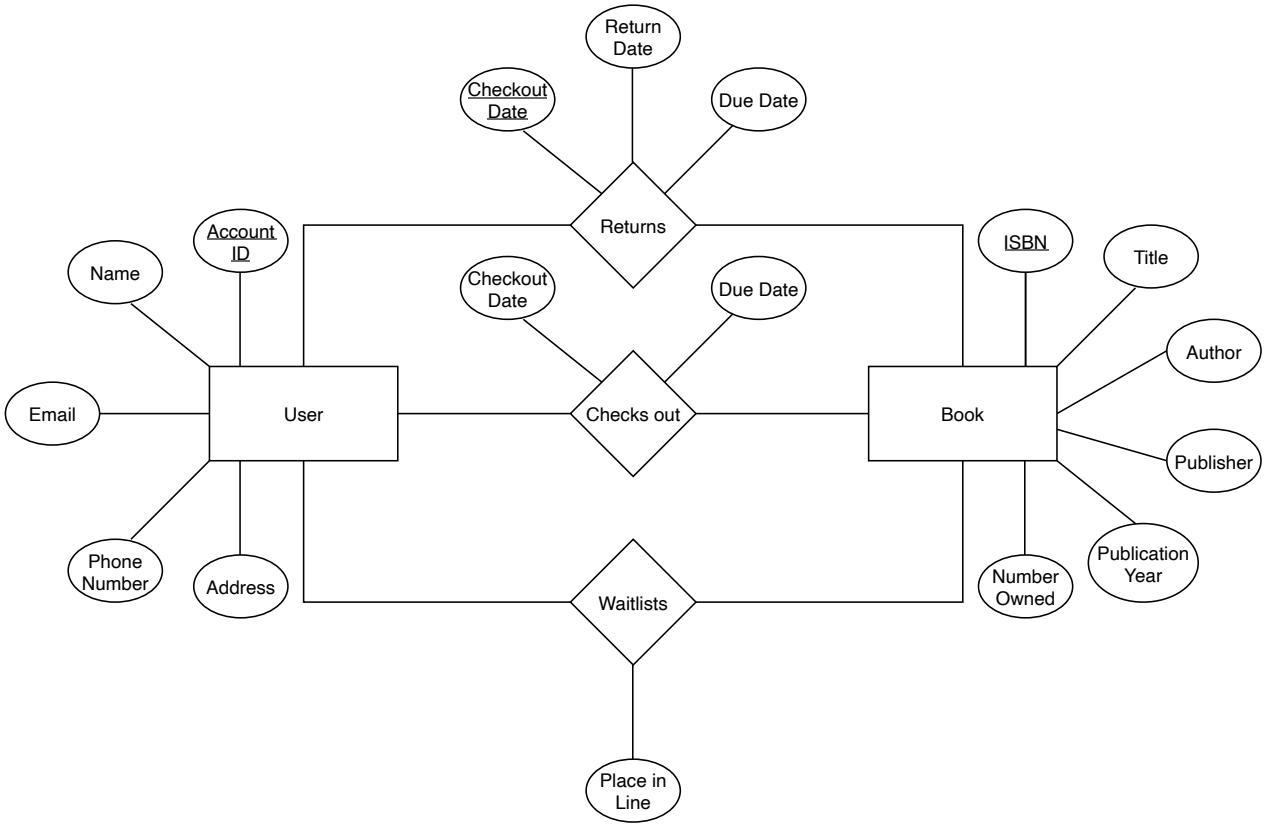

Figure 1. Library ER Diagram

Book

Attributes	Type	Description
isbn	VARCHAR(16)	The International Standard Book Number which uniquely identifies a book
title	VARCHAR(256)	The title of the book
author	VARCHAR(128)	The author(s) of the book
publisher	VARCHAR(128)	The publisher of the book
publication_year	INT(4)	The year the book was published
num_owned	INT(3)	The total number of copies of the book that the library owns

User

Attributes	Type	Description
account_id	VARCHAR(16)	A randomly generated unique ID that identifies a user
name	VARCHAR(32)	The full name of the user
address	VARCHAR(64)	The home address of the user
phone_number	VARCHAR(16)	The phone number of the user
email	VARCHAR(32)	The email of the user

Loan

Attributes	Type	Description
isbn	VARCHAR(16)	The ISBN of the book that is checked out
account_id	VARCHAR(16)	The ID of the user that has checked out the book
checkout_date	DATE	The date that the user checked out the book
due_date	DATE	The date the the user is expected to return the book

LoanHistory

Attributes	Type	Description
isbn	VARCHAR(16)	The ISBN of the book that was checked out
account_id	VARCHAR(16)	The ID of the user that checked out the book
checkout_date	DATE	The date that the user checked out the book
due_date	DATE	The date that the user was expected to return the book
return_date	DATE	The date that the user returned the book

Waitlist

Attributes	Type	Description
isbn	VARCHAR(16)	The ISBN of the book that the user wants to be waitlisted for
account_id	VARCHAR(16)	The user that wants to be waitlisted for the book
place_in_line	INT(3)	The user's place in line for checking out the book

Python Models

When getting data from a DB table, it is naturally returned as a 2d list. The positions of the columns within that list depend on the order of attributes from the `SELECT` clause. To standardize passing data between the frontend and backend, we have provided Python objects that correspond to a row in each table, i.e. the Book object represents a book in the Book table. These objects are available to you and can be found in the `models` directory.

Whenever the db_handler passes eligible data to the frontend, it is required to be stored in the corresponding object. Functions that require a return value of a certain type have it explicitly stated in the function definition, for example, `def example() -> Book` would return a Book object.

It is recommended (but not required) to create helper functions in the db_handler file that can handle the conversion of a 2d list from a table, into a list of objects. For example, you could create a function, `convert_db_book_to_book_object`, that would take in the list that is returned from `cur.fetchall()`, and returns a list of Book objects after converting each row into a Book object.

SQL Injections

According to the Cybersecurity and Infrastructure Security Agency, SQL Injection was one of the most exploited vulnerabilities in 2023. In short, An SQL Injection attack is when a third party is able to run an SQL query on your database, without your knowledge. For example, if you ask the user to input their name, and they enter `John Smith); DROP TABLE User;`. Your program could then generate an SQL insert statement using their input, and it would read: `INSERT INTO User VALUES (John Smith); DROP TABLE User;);`. In the worst case scenario this could delete all of your user data.

SQL Injection attacks do have a simple solution, called sanitizing the data. This is built into basically every SQL library/module that you would use in any coding language. In Python, when using `cur.execute()`, the function can take in two arguments: the query with placeholders, and the data to fill in the placeholders. For instance, using the example above, you have two options in writing the code.

The Bad Way: `cur.execute(f"INSERT INTO User VALUES ({name})")`

The Good Way: `cur.execute("INSERT INTO User VALUES (%s)", [name])`

The Bad Way is using an f-string, Python's in built method to insert a variable into a string, with

almost no sanitization. The Good Way uses the SQL connector's sanitization, with `%s` or `?` acting as a placeholder for the value to insert. Then, for the second argument, you would then pass in a list with the data that is meant to replace the placeholder(s).

Data sanitization has existed for a while now and is built in to basically every interface used to interact with a DB, but these vulnerabilities remain open due to lazy/irresponsible programming. Leaving your code open for attacks is bad, and we expect your final submission to sanitize any data passed in.

Dates in SQL and Python

An important function of a library system is to track when a user checks out a book, when they should return it by, and when they do return it. All of this is to say that you must perform date arithmetic in the project. Any date given to the frontend is expected to be formatted using the ISO-8601 standard, which is a fancy way of saying it should be formatted like "2000-01-31".

SQL

While it is possible to do the date arithmetic in python, we require you to use MariaDB's native Date functions. The following SQL functions may be useful for your purposes:

- DATEDIFF() - Get the number of days between two dates
- CURRENT_DATE() or CURDATE() - Get the current date
- DATE_SUB() - Subtract a fixed value from a date
- DATE_ADD() - Add a fixed value to a date
- DATE_FORMAT() - Format a date as a string

Python

While we do require you to use SQL for the date arithmetic, we do allow you to use Python to format a date into a string. When selecting an attribute with Date type in SQL, MariaDB returns that object to Python as a DateTime Date object. Since the ISO standard is very commonly used, Date objects have a class method, `.isoformat()`, which returns a string of that date in the ISO standard format. So if we had the date variable, `example_date`, all that needs to be done is `example_date_string = example_date.isoformat()`.

Functions

Function	Description	Corresponding Module
Create, Update, and Delete		
`add_book`	Given a Book object, insert it into the Book table.	SQL I
`add_user`	Given a User object, insert them into the User table.	SQL I
`checkout_book`	Given an ISBN and account_id, find the checkout date (current date) and due date (2 weeks from the checkout date), and insert a new record into the Loan table.	SQL I
`waitlist_user`	Given an ISBN and account_id, waitlist the user for the corresponding book. You must use SQL to determine the new user's place in the waitlist.	SQL II
`return_book`	Given an ISBN and account_id, find the corresponding checkout date and due date (using SQL), and insert a new record into the LoanHistory table. Additionally, the corresponding row in the Loan table should be deleted since the book has now been returned.	SQL II

`edit_user`	Given an account_id and attributes to update, update an existing user's information with the new values.	SQL III
`grant_extension`	Given an ISBN and account_id, find the corresponding loan record and update it's due date to be 2 weeks from the existing due date.	SQL III
`update_waitlist`	Given an ISBN, find every user on it's waitlist and move them up a place in line, while removing the person at the front of the line.	SQL III
Filter and Search		
`get_filtered_users`	Given various filter parameters, return a list of User objects containing every user that meets the criteria.	SQL I
`get_filtered_books`	The same as above but with Books	SQL II
`get_filtered_loans`	The same as above but with Loans	SQL II
`get_filtered_loan_histories`	The same as above but with LoanHistory entries	SQL II
`get_filtered_waitlist`	The same as above but with Waitlist entries	SQL II
Helper Functions		
	Saves the changes made	

`save_changes`	to the DB	SQL I
`close_connection`	Safely close the cursor and connection to the DB	SQL I
`place_in_line`	Given an ISBN and account_id, find the user's place in line in the waitlist for the book.	SQL I
`number_in_stock`	Given an ISBN, return the number of copies that the library currently has in stock, i.e. the number owned minus the number checked out.	SQL II
`line_length`	Given an ISBN, return the total number of people in line for the waitlist for the book.	SQL II

Guided Timeline

In the table above, each function is listed with a corresponding module. This is to let you know that you should be able to complete a function after finishing that module. We'd recommend that you follow along with this structure, as it allows you to incrementally complete the project as the semester goes on, so all the work is not left for the last few days before it's due. To give you a more explicit timeline, you should complete the module's corresponding functions the following weeks into the semester.

Function's Module	Suggested Deadline
SQL I	Week 8
SQL II	Week 11
SQL III	Week 12
Additional SQL Queries	Week 15

Testing

We have provided you with a handful of unit tests to see if your code works. You can run these tests either through your IDE, or by using the terminal and running `python3 public_tests.py`, the output should be something like "FAILED (failures=8, errors=6)". These tests are by no means comprehensive and don't cover every bit of functionality. We suggest that you use the frontend or write additional tests to verify that your code works completely.

SQL Queries

In a separate file, write the following SQL queries and submit it as a PDF file named `project.pdf`. These queries cover modules SQL II and above, so we recommend that you answer these after completing all SQL modules.

1. Create a View containing users with overdue loans, and how many days they are overdue by
2. Create a View containing users that have gotten an extension on a loan, and include how many extensions they've received
3. Create a Trigger that automatically updates a user's place in line in a book's waitlist whenever that book is returned

4. Create a Trigger that automatically inserts a new LoanHistory record when a row in Loan is deleted
5. Using the given CREATE TABLE commands in the SQL files (found in the data/ directory) as a starting point, add Foreign Key constraints wherever reasonable. Your solution to this problem should contain multiple CREATE TABLE commands. You *can* add `ON UPDATE` and `ON DELETE` clauses but they will not affect grading.
6. Find the soonest due date for all books that are currently waitlisted
7. Find all the books that have been checked out, are currently checked out, and currently waitlisted. For each of these cases find how many times that has happened
8. Find users who share an address with other users

What to Submit

When submitting your project, you are expected to turn in the `db_handler.py` and `project.pdf` files on Canvas. Your DB Handler should include all of your python code, as well as your UFID and name in the respective variables at the top of the file. The PDF file should essentially look like your homework, with your work clearly numbered/labeled, and your name, assignment name, and due date at the top.