Nikita Belii
COP3540
11/07/2024

Project Report: Cart Management GUI and Database Operations

Task Overview

This project involved creating a Cart Management System with a GUI interface and backend database operations using MySQL and Python. The primary tasks included setting up the database, implementing stored procedures, and building a GUI for interacting with the cart. Below is a detailed breakdown of each task, along with the steps, code, and results.

For Mysql inputs and outputs I have attached TerminalSavedOutput.txt

Task 1: Database and GUI Setup

1) Description

The first step was to set up the database tables in MySQL to represent the essential entities for a cart system. I designed the following tables:

- **Product**: Stores product information (like name, category, and price).
- **Customer**: Contains customer details.
- **Cart**: Keeps track of products added to each customer's cart.

After setting up the tables, I developed a GUI in Python using `tkinter`. The GUI includes:

- Fields to input `Customer ID`, `Product ID`, and `Quantity` for adding items to the cart.
- A button to "Add to Cart" based on the input values.
- Another section to calculate the total cost for a customer's cart by entering the `Customer ID`.

2) Implementation

Here is the code used to create the `Product`, `Customer`, and `Cart` tables in MySQL:

```
CREATE TABLE Product (
    product_id INT PRIMARY KEY,
    name VARCHAR(255),
```

```
    category_id INT,
    price DECIMAL(10, 2)
);

CREATE TABLE Customer (
    customer_id INT PRIMARY KEY,
    name VARCHAR(255)
);

CREATE TABLE Cart (
    cart_id INT AUTO_INCREMENT PRIMARY KEY,
    customer_id INT,
    product_id INT,
    quantity INT,
    FOREIGN KEY (customer_id) REFERENCES Customer(customer_id),
    FOREIGN KEY (product_id) REFERENCES Product(product_id)
);
```

For the GUI, I used `tkinter`. However, due to some issues with visibility in JupyterLab and Terminal, the GUI was only partially tested in the environment.

3) Results

The tables were created successfully in MySQL. I was able to add records to each table and execute basic queries to verify the structure. The GUI opened and displayed the elements, but unfortunately, the input fields were not visible in my testing environment.

Task 2: Adding Items to the Cart and Calculating Total

1) Description

This task focused on implementing functionality to:

1. Add items to a customer's cart.
2. Calculate the total cost of items in the cart.

I created two main functions in Python:

- **`add_to_cart`**: This function takes in `customer_id`, `product_id`, and `quantity` as inputs and adds the specified quantity of the product to the cart for the given customer.

- **`calculate_cart_total`**: This function calculates the total price of all items in the cart for a specific customer.

2) Implementation

The `add_to_cart` and `calculate_cart_total` functions were implemented in Python using `mysql.connector` to interact with the MySQL database.

Here's a simplified version of the code:

```
def add_to_cart(customer_id, product_id, quantity):
    connection = create_connection()
    cursor = connection.cursor()
    cursor.callproc('AddToCart', [customer_id, product_id, quantity])
    connection.commit()
    cursor.close()
    connection.close()

def calculate_cart_total(customer_id):
    connection = create_connection()
    cursor = connection.cursor()
    cursor.execute("SELECT CalculateCartTotal(%s)", (customer_id,))
    total = cursor.fetchone()[0]
    cursor.close()
    connection.close()
    return total
```

3) Results

Testing was done by calling these functions with specific customer and product IDs. For example:

- **Adding Items**: We added a product with `product_id = 1` and quantity 3 for `customer_id = 1`.
- **Calculating Total**: After adding items, we calculated the cart total for `customer_id = 1`.

The functions worked correctly, with SQL operations showing that items were added and the total was calculated accurately.

Outputs:

```
Adding items to the cart...
Database connection established.
Added product 101 to cart with quantity 2.
Connection closed after adding to cart.
Database connection established.
Added product 102 to cart with quantity 1.
Connection closed after adding to cart.

Calculating cart total...
Database connection established.
Total cart cost for customer 1: $1399.98
Connection closed after calculating cart total.
```

```
Database connection established.
Added product 1 to cart with quantity 2.
Connection closed after adding to cart.
Database connection established.
Added product 2 to cart with quantity 1.
Connection closed after adding to cart.
```

```
Database connection established.
Total cart cost for customer 1: $1399.98
Connection closed after calculating cart total.
```

```
[10]:
```

Task 3: Stored Procedures and Functions

1) Description

I implemented two SQL procedures/functions to handle cart management:

- **AddToCart**: A stored procedure to add or update items in the cart.
- **CalculateCartTotal**: A function to calculate the total value of items in the cart for a given customer.

2) Implementation

Here's the SQL code for creating the stored procedure and function:

```
DELIMITER //
CREATE PROCEDURE AddToCart(IN customer_id INT, IN product_id INT, IN
quantity INT)
BEGIN
    DECLARE cart_count INT;
    SELECT COUNT(*) INTO cart_count FROM Cart WHERE customer_id =
customer_id AND product_id = product_id;
    IF cart_count > 0 THEN
        UPDATE Cart SET quantity = quantity + quantity WHERE
customer_id = customer_id AND product_id = product_id;
    ELSE
        INSERT INTO Cart (customer_id, product_id, quantity) VALUES
(customer_id, product_id, quantity);
    END IF;
END //
DELIMITER ;

DELIMITER //
CREATE FUNCTION CalculateCartTotal(customer_id INT) RETURNS
DECIMAL(10,2)
BEGIN
    DECLARE total DECIMAL(10,2);
    SELECT SUM(Product.price * Cart.quantity) INTO total
    FROM Cart JOIN Product ON Cart.product_id = Product.product_id
    WHERE Cart.customer_id = customer_id;
    RETURN total;
END //
DELIMITER ;
```

3) Results

I tested the stored procedure by calling it through Python and MySQL queries. The procedure successfully added items to the cart, either updating quantities if the item already existed or inserting a new row otherwise. The function returned the correct total cart cost for the customer.

Challenges

This project had some tricky parts, and I faced quite a few challenges along the way. Below is a list of the main problems I ran into and what we did to try to solve them.

1. `tkinter` GUI Display Issues

One of the biggest issues was making the `tkinter` GUI display correctly. In JupyterLab, the input fields for entering customer and product info were often invisible, even though the code seemed to be correct. Here's what we tried:

- Changing the background colors and text colors of the input fields and labels.
- Running the code in different places, including JupyterLab, Terminal, and standalone Python files (like `cart_management_gui.py`).
- Using different programs to run the code, like VSCode, PyCharm, and IDLE.

Despite all these attempts, I couldn't get the GUI to display perfectly in JupyterLab. In the future, I might consider using a different tool like `ipywidgets` or even building a small web app with `Flask` to make the interface more compatible with different environments.

2. Database Connection Errors

I sometimes had trouble connecting to the MySQL database, especially when using Python's `mysql.connector`. An error I saw often was:

- `Error: 2055 - Cursor is not connected`

This error would happen at random times, often when I was running stored procedures or fetching data. To troubleshoot:

- I checked the connection settings in the code to make sure they were correct (host, user, password, database).
- Added extra error messages and print statements to track when the connection was failing.

While I fixed most connection issues, this specific error still happened occasionally, possibly due to configuration issues on macOS or something in `mysql.connector` itself.

3. Foreign Key Constraint Errors

When trying to add items to the `Cart` table, I got the following error:

- `Error adding to cart: 1452 (23000): Cannot add or update a child row: a foreign key constraint fails`

This error meant that the `product_id` I was using didn't exist in the `Product` table. To solve this:

- I checked the `Product` table to find valid `product_id`s.
- I inserted sample products with specific `product_id`s (like 1 and 2) so I could test with existing data.

This error taught me the importance of having valid data in related tables, especially when using foreign keys.

4. Syntax Errors in SQL Procedures and Functions

Writing SQL procedures and functions led to a lot of syntax errors. I ran into problems with commands like `DELIMITER` and issues with variable declarations. Some errors included:

- `ERROR 1064 (42000)`: This general error meant there was a syntax mistake in the SQL code. Sometimes, it happened because the `DELIMITER` command wasn't placed correctly.
- Issues with variables that were undeclared or not handled correctly in the procedure.

To fix these, I had to carefully go through each line, double-checking MySQL syntax and sometimes rewriting parts of the SQL code. This was time-consuming but helped us learn the importance of SQL syntax details.

5. Testing Issues in JupyterLab

Testing code in JupyterLab added some extra challenges:

- The `tkinter` GUI didn't display well in JupyterLab, which made it hard to check the functionality.
- JupyterLab doesn't always show real-time output and errors as clearly as a regular Python environment.

To work around this, I tried running the code directly in Terminal and other editors to get better feedback. However, the display problems with the GUI remained, showing me that JupyterLab might not be the best place to test `tkinter`.

6. Syntax Errors in MySQL

Writing SQL code in MySQL also caused some syntax errors, especially when creating procedures and functions. Here are a few examples:

- `ERROR 1064 (42000)`: This common error happened when the syntax wasn't correct, often due to missing `DELIMITER` commands or incorrect placements of SQL commands.
- Errors with nested SQL statements and variable declarations that required careful attention to syntax rules.