0.1 Banking System GUI

We have provided a skeleton GUI application for a banking system. Your task is to implement the backend logic to connect the application to a MySQL database and perform transactional operations.

0.1.1 Tools

MySQL

Functionality: A powerful, open-source Relational Database Management System (RDBMS) that stores and manages structured data using SQL.
Variances: MariaDB (a community-developed fork of MySQL, effectively a drop-in replacement).

phpMyAdmin

Functionality: A free software tool written in PHP, intended to handle the administration of MySQL over the Web. It supports operations on databases, tables, columns, relations, indexes, users, permissions, etc.

Tkinter

Functionality: The standard Python interface to the Tk GUI toolkit. It provides widgets such as buttons, labels, and text boxes to build desktop applications.

PyMySQL

Functionality: A pure-Python MySQL client library. It allows Python programs to connect to a MySQL server and execute SQL queries.

0.1.2 Database Schema

You must create the following tables in your MySQL database to support the application:

1. **Customers:** Stores user identity.
   - `customer_id` (int, PK), `name` (text), `tax_id` (text, unique).
2. **Accounts:** Stores individual bank accounts.
   - `account_id` (int, PK), `customer_id` (FK), `balance` (decimal).
3. **Transactions:** Stores an audit log of all operations.
   - `transaction_id` (int, PK), `account_id` (FK), `transaction_type` (text), `amount` (decimal), `created_at` (timestamp).
4. **BankReserves:** Stores the total cash holding of the bank branch.
   - `branch_id` (int, PK), `total_reserve` (decimal).
5. **AllCustomerTransactions (View):** A joined view for reporting.
   - Logic: Join `Customers` $\bowtie$ `Accounts` $\bowtie$ `Transactions`.
   - Columns: `CustomerName`, `AccountID`, `Type`, `Amount`, `Date`.

0.1.3 Lab Tasks

Open the provided `banking_gui.py` file and complete the following TODO sections:

1. **Server Connection:** Implement logic to `connect()` and `disconnect()` from the MySQL database server. Handle connection errors gracefully.
2. **Banking Operations:**

(a) **Open Account:** Implement SQL to insert a new customer and account.

(b) **Deposit (Transaction):**

- Update the specific `Account` balance ($+Amount$).
- Update the `BankReserves` total reserve ($+Amount$).
- **Requirement:** These updates must be wrapped in a transaction block. If one fails, both must roll back.

(c) **Withdraw (Transaction):**

- Check if the `Account` has sufficient funds ($Balance \geq Amount$).
- Update the `Account` balance ($-Amount$).
- Update the `BankReserves` total reserve ($-Amount$).
- **Requirement:** All steps must be atomic.

(d) **Transfer (Transaction):** The core ACID test. Move money from Account A to Account B.

- Wrap operations in a `BEGIN ... COMMIT` transaction block.
- If any step fails (e.g., insufficient funds), `ROLLBACK` the entire transaction.

(e) **Check Balance:** Query and display the current balance.

(f) **Bank Statement:** Query the `AllCustomerTransactions` view to display the user's transaction history.