

Database SET225

**ATM MACHINE**

Team Members

|  |  |
| --- | --- |
| Name | ID |
| Mahmoud Ayman | 192200089 |
| Seif Adham | 192200119 |
| Jad Sherif | 192200134 |
| Youstina Adel | 192200148 |
| Ruba Tarek | 192200243 |
| Sohaila Tamer | 192200251 |

**Supervised By:**

Dr. Ahmed Osman

TA. Eng. Doaa Amin

# Table of Contents

Introduction ………………………………………………………………………….. Page 3

Database Design Process…..…………………………………………………....... Page 4

Implementation Details ……………………………………………………………… Page 8

Application Development ………………………………………………………..….. Page 9

Challenges and Solutions…………………………………………………………....Page 11

Conclusion ………………………………………………………………………….....Page 13

**Introduction**

**ATM Machine**

1. Domain:

Financial Technology (FinTech)

This project falls under the FinTech domain, focusing on the design and implementation of a digital banking platform that enables users to manage their accounts, perform transactions, and track their financial activities.

2. Initial Scope:

The initial version of the system will include the following features:

• User registration with unique usernames and secure PIN codes.

• Viewing account balance and creation date.

• Performing transactions including deposits, withdrawals, and transfers.

• Recording and tracking all transactions between users.

• Timestamping all actions for transparency and tracking.

The system will ensure data consistency, maintain a transaction history, and enforce unique user identities.

3. Preliminary List of Entities:

1. User (Account Holder)

• account\_id

• username

• pin\_code

• balance

• created\_at

2. Transaction

• transaction\_id

• sender\_id

• receiver\_id

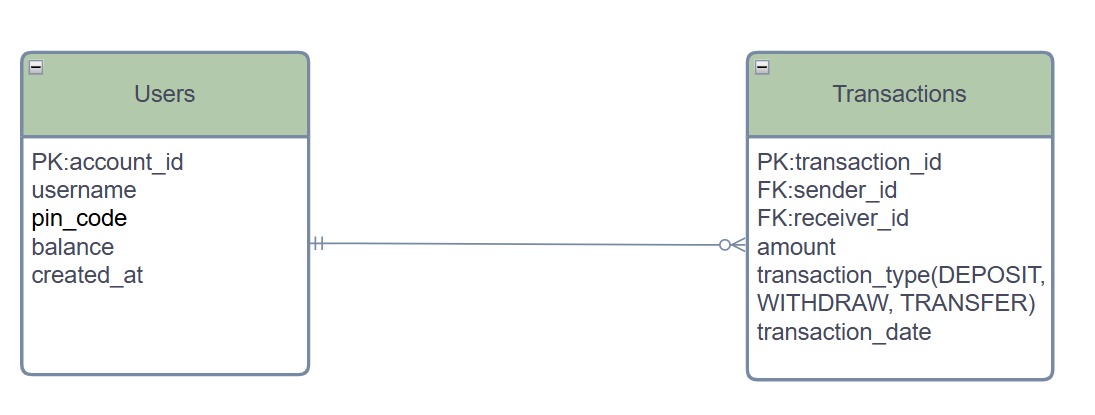
• amount

• transaction\_type

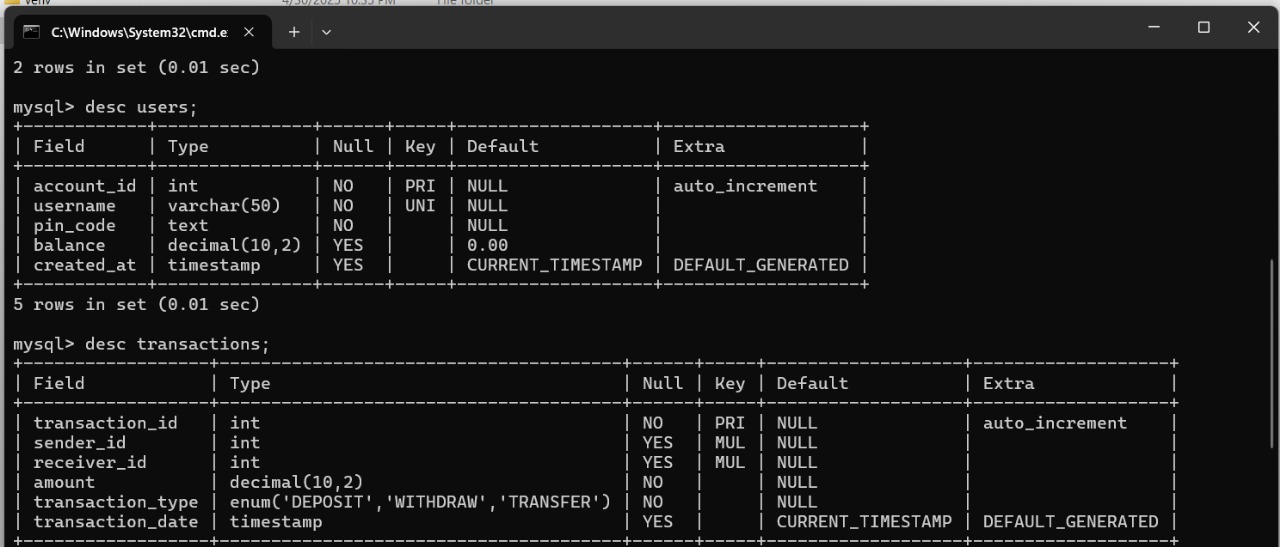
• transaction\_date

**Database Design Process**

**ER-Diagram :**

****

**Relational Schema Design :**

****

**Normalization :**

**1NF:**

**User:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **account\_id** | **username** | **pin\_code** | **balance** | **created\_at** |

**Transaction:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **transaction\_id** | **sender\_id** | **receiver\_id** | **amount** | **transaction\_type** | **transaction\_date** |

**2NF:**

**User:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **account\_id** | **username** | **pin\_code** | **balance** | **created\_at** |

**Transaction:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **transaction\_id** | **sender\_id** | **receiver\_id** | **amount** | **transaction\_type** | **transaction\_date** |

**3NF:**

**User:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **account\_id** | **username** | **pin\_code** | **balance** | **created\_at** |

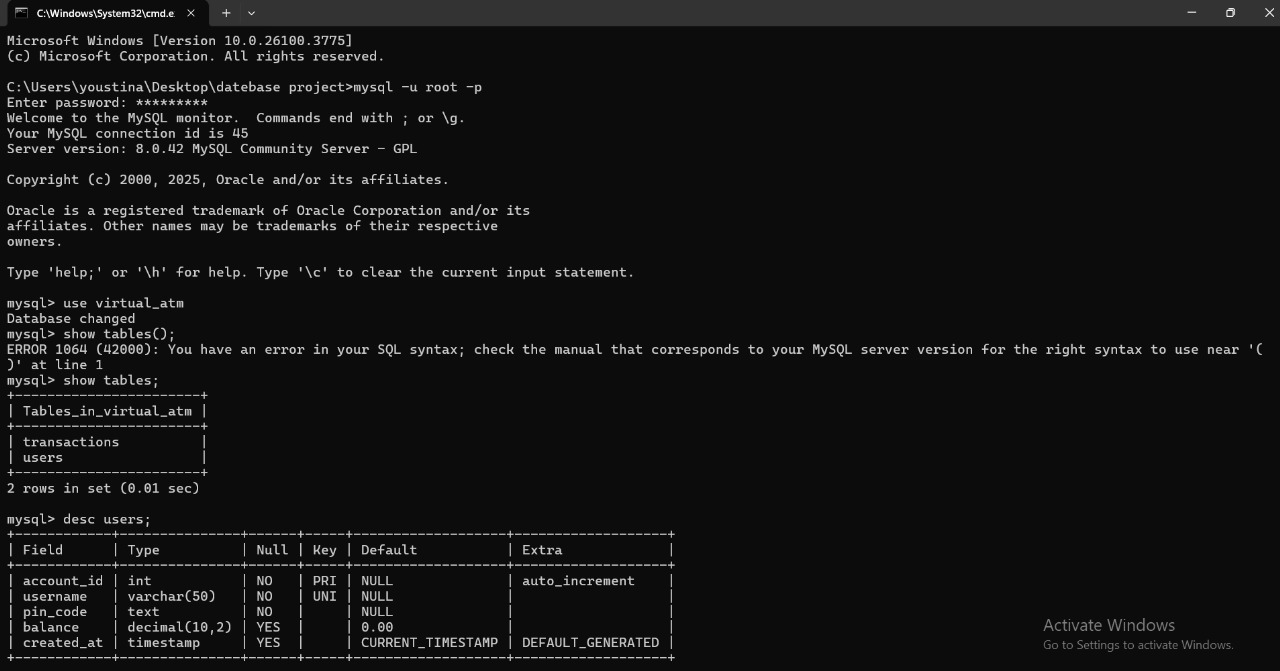
**Transactions:**

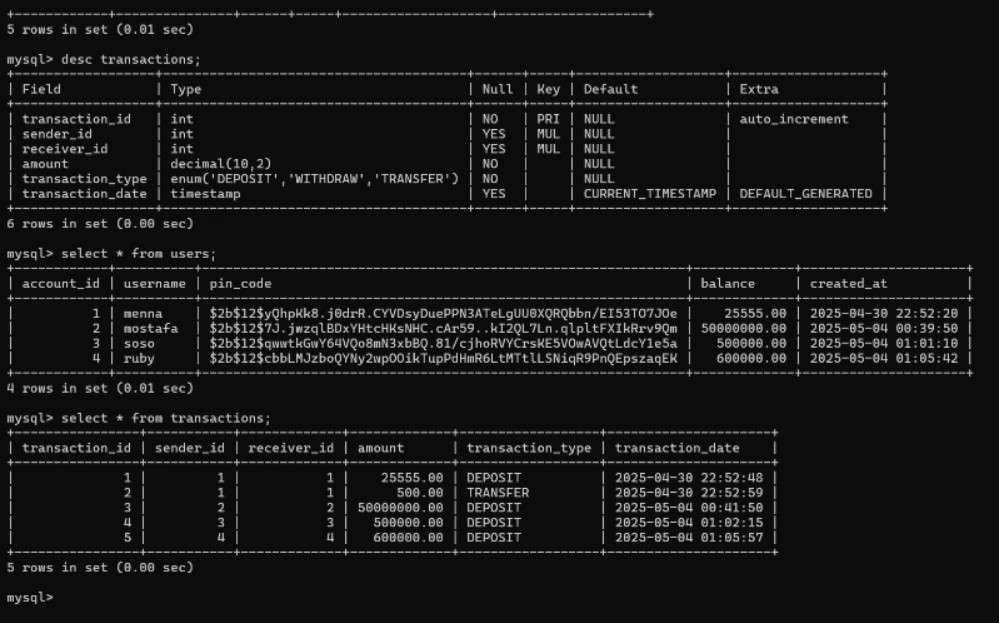
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **transaction\_id** | **sender\_id** | **receiver\_id** | **amount** | **type\_id** | **transaction\_date** |

**Transaction Types:**

|  |  |
| --- | --- |
| **type\_id** | **type\_name** |

**Implementation Details**

****

****

**Application Development**

**Technologies Used**

**Python 3**

Core programming language.

Used for logic, database operations, and UI event handling.

**PyQt5**

GUI toolkit for Python.

Provides widgets for creating modern, responsive interfaces.

Enables full-screen mode, custom-styled ATM screens, and interactive buttons.

**MySQL**

Relational database system.

Stores user information, transactions, and balances securely.

**bcrypt**

Used to securely hash and verify user PIN codes.

How the Application Interacts with the Database

1. Database Connection and Initialization

Your DatabaseHandler class manages the database connection.

When the application starts:

It connects to MySQL.

Creates the database (CREATE DATABASE IF NOT EXISTS) if it doesn’t exist.

Creates two tables:

users: stores account info (username, hashed PIN, balance).

transactions: records deposits, withdrawals, and transfers.

2. User Registration (Card Activation)

In the Activate New Card screen:

User enters a username and 4-digit PIN.

PIN is hashed using bcrypt for security.

New user is inserted into the users table.

3. User Login

On the login screen:

User provides credentials.

The application fetches the hashed PIN from MySQL.

bcrypt is used to compare the provided PIN to the stored hash.

If valid, the user is logged in.

4. Banking Transactions

After login, users can:

* Check Balance: SELECT balance FROM users.
* Deposit:Adds amount to user’s balance (UPDATE users SET balance = balance + ?).

Adds a record in transactions table.

* Withdraw:Checks if sufficient funds are available.

Deducts amount and logs transaction.

* Transfer: Checks funds.

Moves money from sender to receiver using two update statements.

Records the transaction.

5. Secure Data Handling

PINs are never stored or transmitted in plain text.

bcrypt ensures passwords are hashed with a salt before storing.

Input validation prevents invalid or malicious entries.

**Challenges and Solution**

1. Secure PIN Storage

Challenge: Storing users’ PIN codes securely to protect sensitive data.

Solution: Integrated the bcrypt library to hash PIN codes before storing them in the database.

During login, hashed values are verified using secure comparison methods to prevent exposure of raw credentials.

2. Automatic Database and Table Initialization

Challenge: Ensuring the database and necessary tables are available on first run.

Solution: The system first connects to MySQL without a database, creates it if not present, and then connects with the specified schema. Tables are created using CREATE TABLE IF NOT EXISTS statements, ensuring smooth first-time deployment.

3. Full-Screen Kiosk Interface with Custom Styling

Challenge: Building a full-screen, user-friendly ATM-like interface with no window frame.

Solution: PyQt5's showFullScreen() and FramelessWindowHint flags were used to simulate an ATM UI. Custom stylesheets were applied to ensure consistent and visually appealing design across all widgets. A custom exit button was added for accessibility.

4. Seamless Multi-Screen Navigation

Challenge: Managing multiple screens (login, deposit, withdraw, etc.) in a responsive way.

Solution: Implemented QStackedWidget to hold and switch between different screens, maintaining UI context and enabling smooth transitions.

5. Safe Financial Transactions

Challenge: Avoiding inconsistencies and ensuring data integrity during deposits, withdrawals, and transfers.

Solution: Transactions are validated (e.g., checking for sufficient balance), and balances are updated atomically. Each transaction is recorded in a separate transactions table with proper linkage to user accounts.

6. Preventing Duplicate Usernames

Challenge: Avoiding duplicate usernames which could lead to user conflicts.

Solution: Enforced a UNIQUE constraint on the username field in the database and checked for duplicates using a pre-insertion query to provide real-time feedback.

7. Robust Error Handling

Challenge: Preventing crashes due to invalid input or database errors.

Solution: Wrapped database logic in try-except blocks and used QMessageBox to notify users of errors such as invalid credentials, insufficient balance, or incorrect formats.

**Conclusion**

The Virtual ATM system effectively replicates the core functionalities of a real-world ATM using a combination of Python, PyQt5, and MySQL.

It provides users with secure access to banking features like login, card activation, balance inquiry, deposit, withdrawal, and fund transfer through an intuitive full-screen graphical interface.

By separating the database logic (using DatabaseHandler) from the UI components, the system achieves clean modularity, easier debugging, and scalability.

Security measures like hashed PIN storage and transaction logging ensure both data protection and accountability.

Additionally, the responsive user interface, enhanced with PyQt5 styling, mimics the experience of physical ATM machines, making it suitable for educational use or prototyping real banking systems.

Potential Future Enhancements

Biometric Authentication Integration

Incorporate facial recognition or fingerprint sensors to enhance security beyond PIN codes.

Support for Multiple Currencies

Expand the system to handle transactions in different currencies with real-time exchange rate updates.

Admin Dashboard

Create an administrative interface to monitor usage, manage accounts, and generate reports.

Transaction History Display

Let users view and export their transaction history within the interface.

Improved Concurrency Management

Implement database transactions and locking mechanisms to safely handle multiple users accessing the system simultaneously.

Touchscreen Optimization

Design the UI specifically for touchscreen kiosks with larger buttons and simplified navigation.

Integration with External APIs

Integrate with external financial APIs for bank-level features like loan applications, bill payments, or balance syncing with real accounts.

Mobile Version

Develop a companion mobile app using technologies like Kivy or Flutter to access the same services remotely.

Voice Assistant or Accessibility Features

Add text-to-speech or voice-command support to improve accessibility for visually impaired users.

Cloud Deployment

Move the database and application to the cloud (e.g., AWS, Azure) for remote access and better scalability**.**