BANKING SYSTEM DATABASE

Authored By:

Marlon Mavinier 000372201

Nico Hammer 000377237

Course: CPRO 1301A Database Design and SQL

# Table of Contents

## Introduction

## Requirement Analysis

## Entity-Relationship Diagram

## Data Dictionary

## SQL Implementation

### Data Definition Language (DDL)

### Data Manipulation Language (DML)

### Single-Table Queries

### Multi-Table Queries

### Advanced Queries

## Testing

## Conclusion

## References

# Introduction

## Project overview

## For our database design project, we have chosen to use the domain of a banking system.

## Objectives

## The main objectives of this project are to:

1. have multiple types of users (customer, teller, administrator) with different abilities in interacting with the database.
2. Provide a useable and functional mock banking system with multiple types of users and accounts as well as transaction records.

## Tools Used

To create our database, we will be using the following tools:

* MySQL as the query language used to interact with the database
* MySQL Workbench and DBeaver as the development environments used to create the database
* Draw.io to create the E-R diagram of the database

# Requirement Analysis

## Business Requirements

Customers may create either one or both of the following: a chequing account, a savings account. Each customer is assigned a unique customer id and must provide their first and last name as well as one or both of their email or phone number upon account creation.

Each account must have a unique account id as well as the corresponding customer id, the account balance, and the account type. A record of every account’s transactions must be recorded and contain the transaction amount, transaction date, transaction type, and the sender and receiver IDs; accounts may have more than one transaction but a transaction can only be associated with one account.

Tellers must be able to view and update account balances as well as account emails/phone numbers, view the account and transaction information and ID’s, and create additional accounts for new or existing customers.

Administrators must be able to view and update all information available within the database.

## Data Requirements

The database will have at least the following entities: customers, accounts, transactions. Customers will be a strong entity, and the accounts and transactions will be weak entities as they are reliant on the customer existing. The relationship between them will be such that: each customer may have one or two accounts, each account can only have one customer\_id associated with it, each account may have 0 or more transactions, and each transaction may only have one account\_id associated with it. Each entities attributes are listed below.

Customers: customer\_id, first\_name, last\_name, email\_address, phone\_number

Accounts: account\_id, customer\_id, account\_type, account\_balance

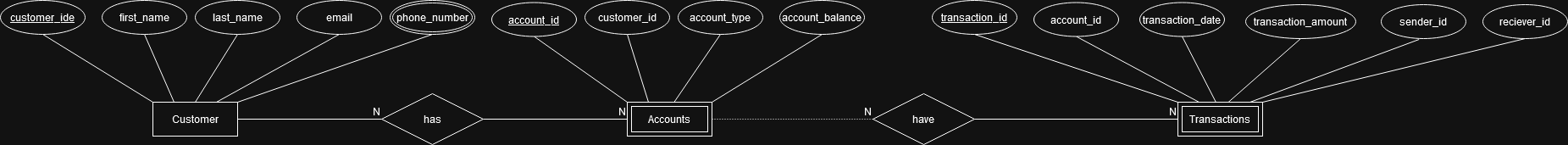
Transactions: transaction\_id, account\_id, transaction\_date, transaction\_amount, sender\_id, receiver\_id

## User Roles

The database will have at least the following user roles:

* Customers who can create one of two account types, view their account balance and email/phone number, as well as being able to view their transactions.
* Tellers who can view and update account balances and id’s, view the id’s associated with both accounts and transactions, create additional accounts
* Administrators who will be able to view and modify all information in the database

# Entity Relationship Diagram



## Entities

The entities currently in the E-R diagram are the following: Customers, Accounts, Transactions.

* Customers are the strong entity and have the attributes: customer\_id,first\_name,last\_name, email,phone\_number
* Accounts is a weak entity and has the attributes: account\_id,customer\_id,account\_type,account\_balance
* Transactions is a weak entity and has the attributes: transaction\_id,account\_id,transaction\_date, transaction\_amount,sender\_id,receiver\_id

## Relationships

The entities in the database will have at least the following relationships:

1. Customers will have a 1:1 total relationship with Accounts as a customer can have up to two account types but they must have at least one account, customer\_id will be the primary key for the Customers table
2. Accounts will have a 1:1 total relationship with Customers as each account can only be associated with one customer using the customer\_id as the foreign key and account\_id as its primary key
3. Accounts will have a 0:M partial relationship with transactions as an account doesn’t necessarily need to have any transactions
4. Transactions will have a M:1 total relationship with Accounts as each account can have multiple transactions using the account\_id as the foreign key and transaction\_id as its primary key

# Data Dictionary

## Normalization

Each table in the database is normalized to at least the third normal form (3NF). This is proved by the fact that:

* Each column/cell in the tables have only one value (1NF)
* Each column/cell in the tables have a single primary key (2NF)
* There are no transitive dependencies between the tables or columns

**Table: customer**

| **Column Name** | **Data Type** | **Size** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- |
| **customer\_id** | INT | - | PRIMARY KEY, NOT NULL, AUTO\_INCREMENT, UNIQUE | Unique identifier for each customer. |
| **first\_name** | VARCHAR | 50 | NOT NULL | First name of the customer. |
| **last\_name** | VARCHAR | 100 | NOT NULL | Last name of the customer. |
| **email** | VARCHAR | 255 | - | customers email address. |
| **phone** | VARCHAR | 9 | NOT NULL, UNIQUE | Contact number of the customer. |

**Table: accounts**

| **Column Name** | **Data Type** | **Size** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- |
| **account\_id** | INT | - | PRIMARY KEY, NOT NULL, AUTO\_INCREMENT, UNIQUE | Unique identifier for each account. |
| **customer\_id** | INT | - | NOT NULL | Foreign key reference to the customer in the customers table |
| **account\_type** | VARCHAR | 3 | NOT NULL | Type of the customer account (SAV or CHK) |
| **account\_balance** | DECIMAL(10,2) | - | DEFAULT 0.00 | Balance of the account, defaults to 0.00 if no balance is used to create the entry |

**Table: transactions**

| **Column Name** | **Data Type** | **Size** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- |
| **transaction\_id** | VARCHAR | 36 | PRIMARY KEY, NOT NULL, UNIQUE | Unique identifier for each transaction. |
| **account\_id** | INT | - | NOT NULL | Foreign key reference to the account in the accounts table |
| **transaction\_date** | DATE | - | NOT NULL | Date of which the transaction was carried out |
| **transaction\_amount** | DECIMAL(10,2) | - | NOT NULL | The amount of the transaction |
| **sender\_id** | INT | - | NOT NULL | The ID of the account that sent the transaction |
| **reciever\_id** | INT | - | NOT NULL | The ID of the account that recieved the transaction |

# DDL

# 

# A close up of a text AI-generated content may be incorrect.

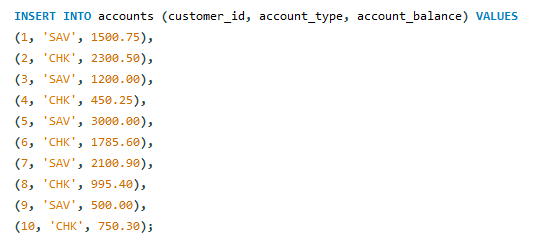
# 

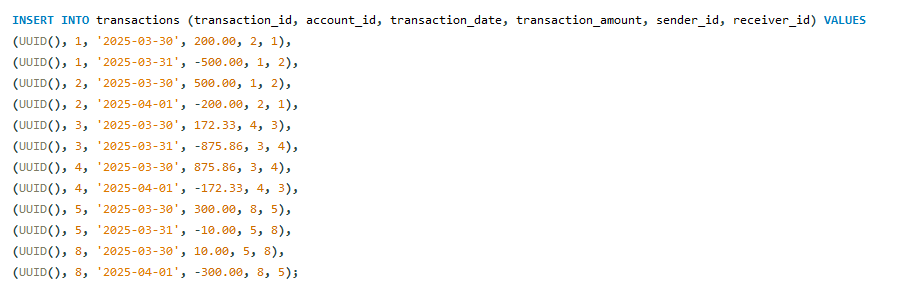
# 

# DML

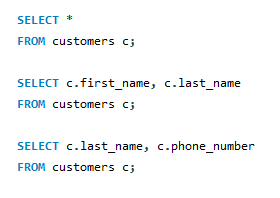
A screenshot of a computer code

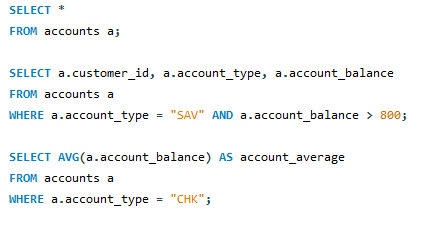
AI-generated content may be incorrect.

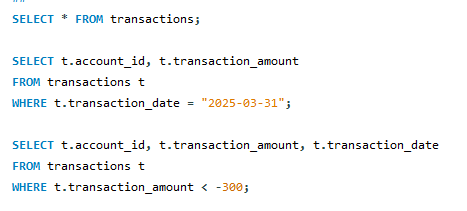




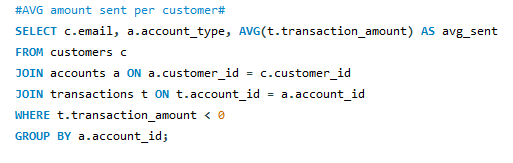
# Single-Table Queries

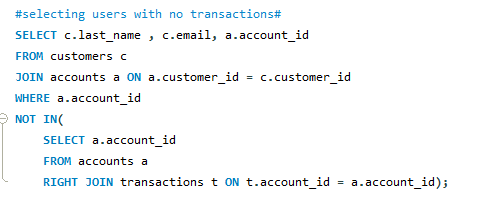




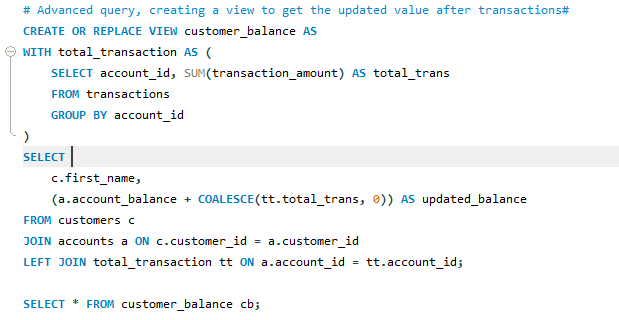


# Multi-Table Queries





# Advanced Queries



# Testing And Validation

# Conclusion

Using MYSQL, we were able to create a simple and functional banking database that handles customers, accounts, and transactions.

Some potential improvements that could be made for the database would be to implement multi-user permissions, or to have more tables such as branches, etc. to create a more scalable and robust database.

# References