# DATABASE SYSTEMS

Sir Anwar Ali

# “BANKING SYSTEM DATABASE”

Presented By :
Marvi (B2433064)
Marina (B2433063)
Saamia(B2433104)

## DATABASE OVERVIEW

A database is an organized collection of data that can be easily accessed, managed, and updated. It stores information in a structured way—usually in tables made up of rows (records) and columns (fields).

**Key components:**

- **Tables:** Hold data (e.g., Customers, Accounts, Transactions).
- **Fields:** Define the type of data stored (e.g., Name, Balance, Date).
- **Primary Key:** A unique identifier for each record.
- **Foreign Key:** Connects data between tables (defines relationships).
- **Queries:** Used to retrieve or modify data.
- **DBMS (Database Management System):** Software that manages the database (e.g., MySQL, Oracle, SQL Server).

## PROJECT OVERVIEW (BANKING SYSTEM)

**Purpose:** To design a database system for a bank to manage customers, accounts, loans, and transactions.

**Goal:** Ensure security, accuracy, and easy access to financial data.

**Tools:** MySQL Workbench for schema design and implementation.

**Outcome:** A working database with sample data and relationships.

## Real-World Example: Habib Bank Limited (HBL)

One of Pakistan’s largest banks with millions of customers and branches worldwide.

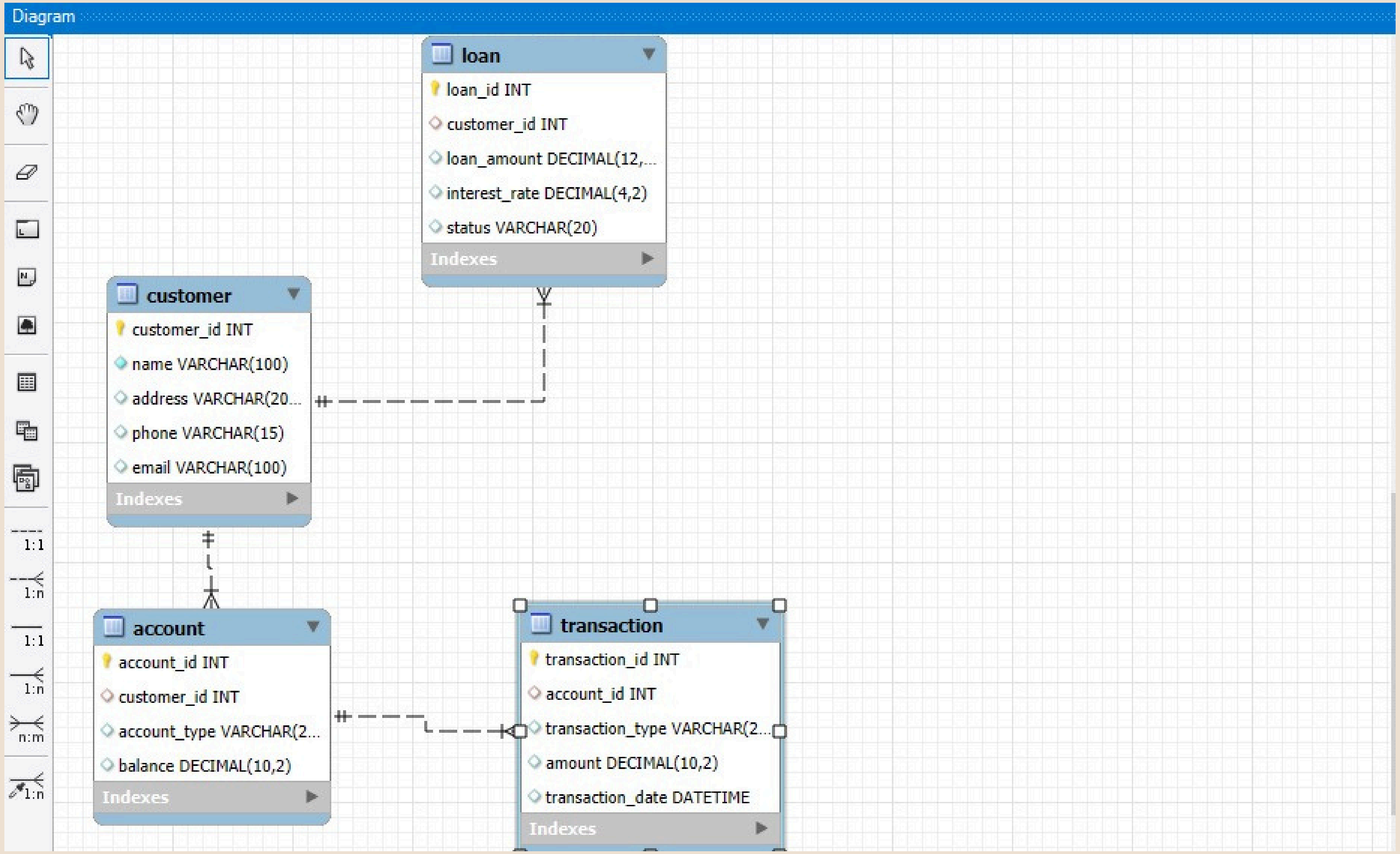
HBL uses advanced database systems to:

- Store and manage customer and transaction data.
- Handle online banking, loans, and payments.
- Ensure real-time updates and secure access.

# ENTITIES & ATTRIBUTES

| **Entity** | **Main Attributes** |
|---|---|
| Customer | Customer_ID, Name, Address, Phone, Email |
| Account | Customer_ID, Name, Address, Phone, Email |
| Loan | Loan_ID, Customer_ID (FK), Amount, InterestRate |
| Transaction | Transaction_ID, Account_ID (FK), Type, Amount, Date |
| Employee | Employee_ID, Name, Position |

# ENTITY RELATIONAL DIAGRAM (ERD)



loan
- loan_id INT
- customer_id INT
- loan_amount DECIMAL(12,...
- interest_rate DECIMAL(4,2)
- status VARCHAR(20)
- Indexes

customer
- customer_id INT
- name VARCHAR(100)
- address VARCHAR(20...
- phone VARCHAR(15)
- email VARCHAR(100)
- Indexes

account
- account_id INT
- customer_id INT
- account_type VARCHAR(2...
- balance DECIMAL(10,2)
- Indexes

transaction
- transaction_id INT
- account_id INT
- transaction_type VARCHAR(2...
- amount DECIMAL(10,2)
- transaction_date DATETIME
- Indexes

# NORMALIZATION

Normalization is the process of organizing data in a database to reduce duplication and ensure data consistency.
It divides large tables into smaller, related ones and connects them using Primary Keys and Foreign Keys.

**TYPES OF NORMAL FORMS:**

- **1NF (First Normal Form)**

Each table has a Primary Key.

- **2NF (Second Normal Form)**

The table is already in 1NF.

- **3NF (Third Normal Form)**

The table is in 2NF.

## SQL JOINS

Joins in SQL are used to combine data from two or more tables based on a related column — usually the foreign key.
They help us view connected information, such as which customer owns which account or which transactions belong to which account.

**TYPES OF JOINS:**

**INNER JOIN –** Returns only matching records from both tables.
**LEFT JOIN –** Returns all records from the left table, even if there's no match in the right one.
**RIGHT JOIN –** Returns all records from the right table, even if there's no match in the left one.
**FULL JOIN –** Combines all records from both tables (MySQL simulates this using UNION).

# DATABASE TRIGGERS

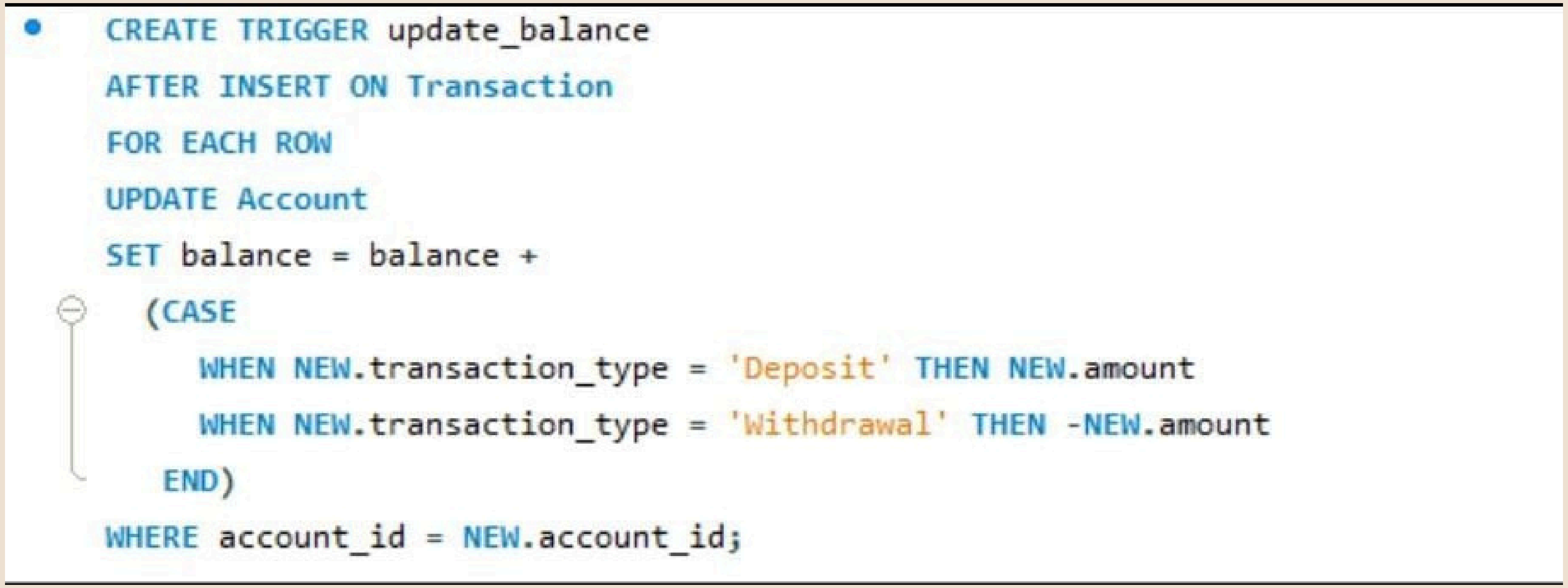
A trigger is an automatic action performed by the database when certain events happen — such as inserting, updating, or deleting data in a table.
Triggers help maintain data accuracy and consistency.

**PURPOSE OF TRIGGERS IN DATABASE:**
It runs automatically whenever something happens — like inserting, updating, or deleting data.

**Example (for banking system):**

When a new transaction is added:

- The trigger automatically updates the account balance,
- So you don’t need to change it manually each time.



```sql
CREATE TRIGGER update_balance
AFTER INSERT ON Transaction
FOR EACH ROW
UPDATE Account
SET balance = balance +
  (CASE
     WHEN NEW.transaction_type = 'Deposit' THEN NEW.amount
     WHEN NEW.transaction_type = 'Withdrawal' THEN -NEW.amount
   END)
WHERE account_id = NEW.account_id;
```

## ACID

**PROPERTIES:**

ACID ensures database transactions are reliable:

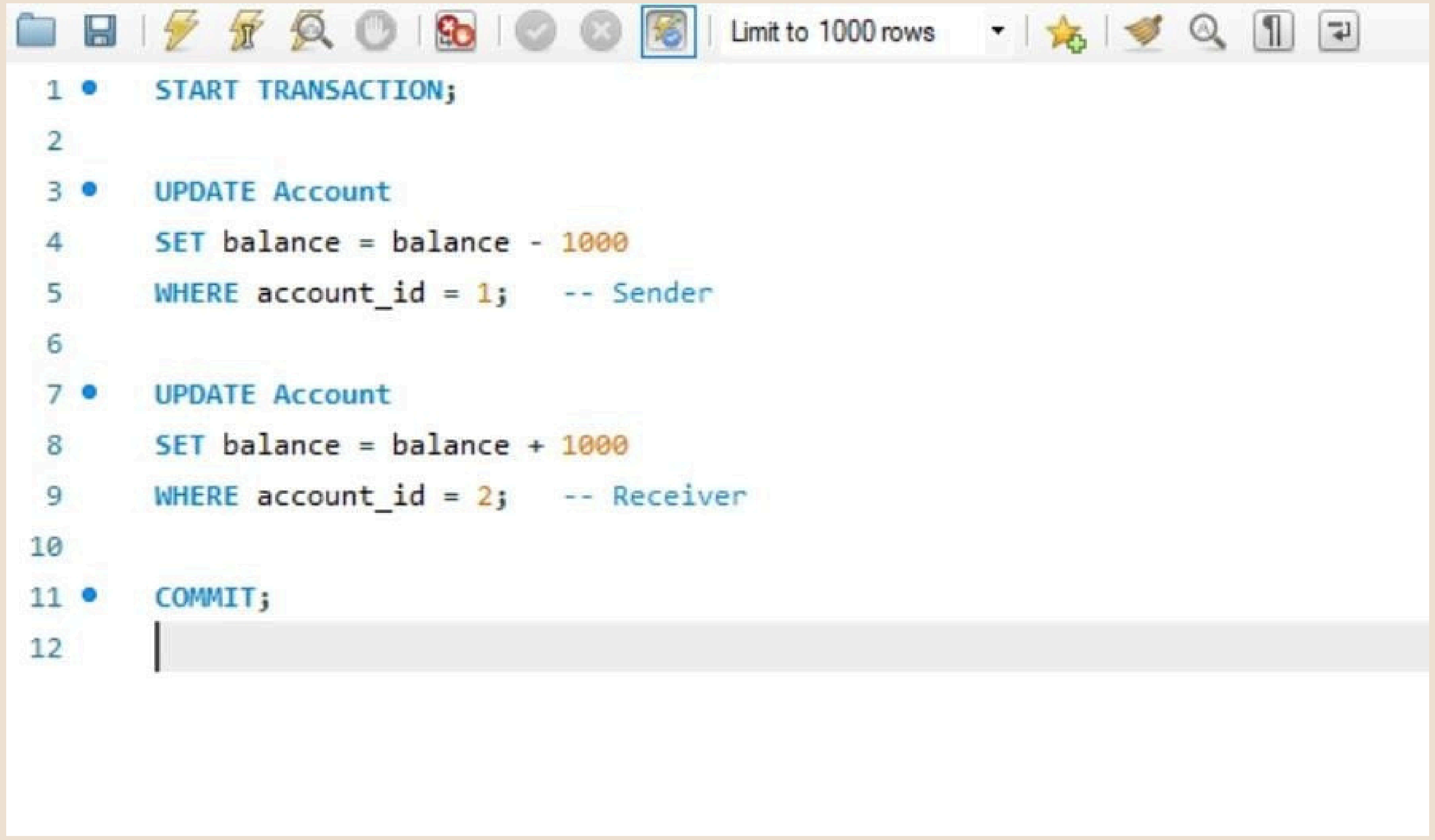
- **A** – Atomicity: All steps in a transaction complete or none do.
- **C** – Consistency: Data remains valid before and after the transaction.
- **I** – Isolation: Transactions don’t affect each other.
- **D** – Durability: Once saved, data stays saved even after failures.

**Example (for banking system):**

When a customer deposits money:

- The system adds money to the account (Atomicity).
- The total balance remains valid (Consistency).

- If another user deposits at the same time, their transaction doesn't interfere (Isolation).
- After saving, even if the system shuts down, the balance remains updated (Durability).



```sql
START TRANSACTION;

UPDATE Account
SET balance = balance - 1000
WHERE account_id = 1;    -- Sender

UPDATE Account
SET balance = balance + 1000
WHERE account_id = 2;    -- Receiver

COMMIT;
```

## SECURITY & RECOVERY

**Security:**
Security in a database ensures that data is protected from unauthorized access, misuse, or corruption. It involves user authentication, access control, and encryption to keep data safe.

**Recovery:**
Recovery is the process of restoring a database to a correct state after a failure, such as a crash or data loss. It uses backups and transaction logs to recover lost or damaged data.

# DATABASE SECURITY MEASURES

Database security measures are the techniques and controls used to protect data from unauthorized access, misuse, or loss. These measures ensure that information remains confidential, accurate, and available. Common database security measures include:

1. **Authentication:** Verifying the identity of users before granting access (e.g., passwords, biometrics).
2. **Authorization:** Defining user permissions and restricting access based on roles.
3. **Encryption:** Converting data into a secure format to prevent unauthorized reading.

4. **Backup and Recovery:** Regularly saving copies of data to restore it in case of loss or corruption.
5. **Auditing and Monitoring:** Tracking user activities to detect suspicious or unauthorized actions.
6. **Firewall and Network Security**: Protecting the database from external threats and cyberattacks.

Together, these measures help maintain the integrity, confidentiality, and availability of database systems.

## CONCLUSION

In this project, we designed a complete banking database system that stores and manages customer, account, loan, and transaction details efficiently.
The project demonstrated how database concepts like relationships, normalization, ACID properties, triggers, security, and recovery ensure data accuracy and reliability.
Overall, it helped us understand how real-world banking operations can be handled through a well-structured and secure database system.

# THANK YOU!