ABC Bank MySQL Database Project

PRESENTED BY : POOJA VHOTKAR

DATE: 5 –MAY-2025

Under Guidance of:
Priya Yadav

PROJECT OVERVIEW

Objective:

To design a relational database for a ABC bank with support for customers, accounts, transactions, loans, and loan payments.

Tools Used:

MySQL 8.0

Key Concepts covered:

- DDL, DML, DQL
- Constraints
- SQL Operators
- Aggregate Functions
- Joins

ER DIAGRAM

ER Diagram depicting relationships:

- Customer ↔ Account (1:M) Account ↔ Transactions (1:M)

- Customer ↔ Loans (1:M) Loans ↔ LoanPayments (1:M)

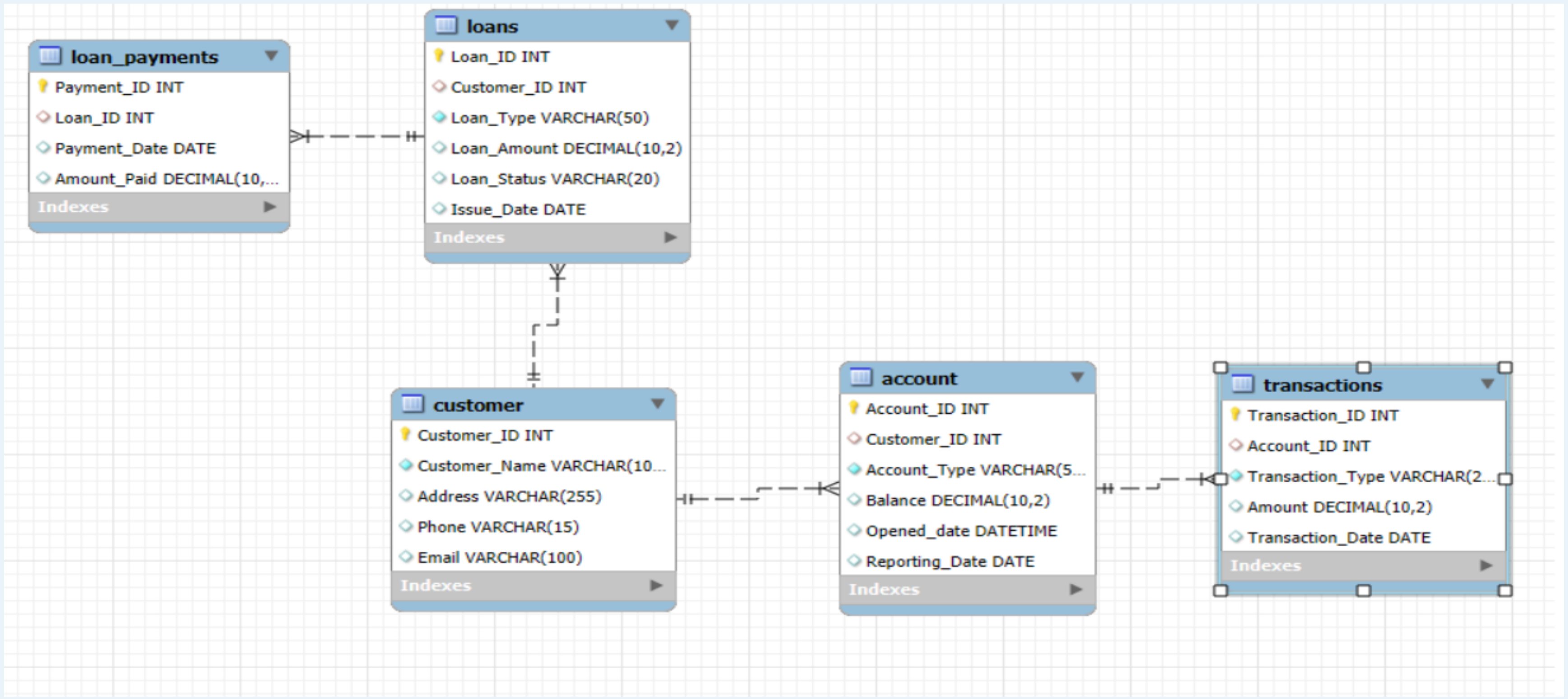

TABLE CREATION – CUSTOMER TABLE

```
CREATE TABLE Customer (
    Customer_ID INT PRIMARY KEY,
    First_Name VARCHAR(50),
    Last_Name VARCHAR(50),
    Email VARCHAR(100),
    Phone_Number VARCHAR(15)
);
```

TABLE CREATION – ACCOUNT TABLE

```
CREATE TABLE Account (
   Account_ID INT PRIMARY KEY,
   Customer_ID INT,
   Account_Type VARCHAR(50) NOT NULL,
   Balance DECIMAL(10,2) CHECK (Balance >= 0),
   Opened_date DATETIME DEFAULT CURRENT_TIMESTAMP,
   Reporting_Date DATE,
   FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
);
```

TABLE CREATION – TRANSACTIONS TABLE

```
CREATE TABLE Transactions (
    Transaction_ID INT PRIMARY KEY,
    Account_ID INT,
    Transaction_Type VARCHAR(50),
    Amount DECIMAL(10,2) CHECK (Amount > 0),
    Transaction_Date DATETIME DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (Account_ID) REFERENCES Account(Account_ID)
);
```

TABLE CREATION – LOANS TABLE

```
CREATE TABLE Loans (
   Loan_ID INT PRIMARY KEY,
   Customer_ID INT,
   Loan_Amount DECIMAL(12,2),
   Interest_Rate DECIMAL(4,2),
   Start_Date DATE,
   End_Date DATE,
   FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
);
```

TABLE CREATION – LOANPAYMENTS TABLE

```
CREATE TABLE LoanPayments (
    Payment_ID INT PRIMARY KEY,
    Loan_ID INT,
    Payment_Amount DECIMAL(10,2),
    Payment_Date DATE,
    FOREIGN KEY (Loan_ID) REFERENCES Loans(Loan_ID)
);
```

SAMPLE INSERT QUERIES

```
INSERT INTO Customer VALUES(1, 'John Doe', 'New York',
'1234567890', 'john@example.com’);

INSERT INTO Account VALUES(101, 1, 'Saving', 1000, '2024-01-01',
'2025-05-01’);

INSERT INTO Transactions VALUES(1001, 101, 'Deposit', 500, '2025-
01-10’);

INSERT INTO Loans VALUES(201, 1, 'Home', 500000, 'Approved',
'2024-01-01’);

INSERT INTO Loan_Payments VALUES(301, 201, '2025-01-10',
10000);
```

SAMPLE UPDATE QUERIES

```
-- Update address for Customer_ID 1.

UPDATE Customer SET Address = 'San Diego’ WHERE Customer_ID = 1;


-- Update balance using arithmetic operator (+).

UPDATE Account SET Balance = Balance + 100 WHERE Account_ID = 101


--Update loan status

UPDATE Loans SET Loan_Status = 'Closed’ WHERE Loan_ID = 205
```

SAMPLE DELETE QUERIES

```
-- Deletes transactions with amount less than 300 using comparison operator (<)

DELETE FROM Transactions WHERE Amount < 300;


-- Deletes loan payments with amount equal to 2000 using comparison operator (=)

DELETE FROM Loan_Payments WHERE Amount_Paid = 2000;
```

SELECT QUERIES

```
-- Get all customers
SELECT * FROM Customer;

-- Accounts with balance > 5000
SELECT * FROM Account WHERE Balance > 5000;

-- Transaction history of an account
SELECT * FROM Transactions WHERE Account_ID = 1001;
```

JOIN QUERIES

```sql
-- 1. Inner Join - List all customers along with their account balances

SELECT c.Customer_ID, c.Customer_Name, a.Account_ID, a.Balance

FROM Customer c

INNER JOIN Account a ON c.Customer_ID = a.Customer_ID;


-- 2. LEFT JOIN (LEFT OUTER JOIN)

-- Returns all rows from the left table, and matched rows from the right table. List
all customers and their accounts, even if they don’t have any account

SELECT c.Customer_ID, c.Customer_Name, a.Account_ID, a.Balance

FROM Customer c

LEFT JOIN Account a ON c.Customer_ID = a.Customer_ID;
```

JOIN QUERIES

3. **RIGHT JOIN (RIGHT OUTER JOIN)**

-- Returns all rows from the right table, and matched rows from the left table. List all accounts and their corresponding customers, even if some accounts have no associated customer (rare, but shown for structure)

SELECT c.Customer_ID, c.Customer_Name, a.Account_ID, a.Balance

FROM Customer c

RIGHT JOIN Account a ON c.Customer_ID = a.Customer_ID;

-- 4. **SELF JOIN** Join a table to itself. Example: Show pairs of customers in the same city

SELECT A.Customer_Name AS Customer1, B.Customer_Name AS Customer2, A.Address

FROM Customer A

JOIN Customer B ON A.Address = B.Address AND A.Customer_ID <> B.Customer_ID;

-- 5. **CROSS JOIN** Returns the Cartesian product (all combinations). Every customer with every account (use carefully!)

SELECT c.Customer_Name, a.Account_IDFROM Customer cCROSS JOIN Account a;

AGGREGATION QUERIES

```
-- Total number of customers
SELECT COUNT(*) FROM Customer;

-- Total balance
SELECT SUM(Balance) FROM Account;

-- Total loan amount disbursed
SELECT SUM(Loan_Amount) FROM Loans;
```

CONSTRAINT

- PRIMARY KEY: Unique identification
- FOREIGN KEY: Maintain relationships
- NOT NULL: Prevent empty values
- CHECK: Ensure valid ranges
- DEFAULT: Auto-fill fields

CHALLENGES & LEARNINGS

- MySQL does not support DEFAULT CURRENT_TIMESTAMP for DATE type

- Used DATETIME as data type instead DATE for Opened_date

- Learnt creating ER diagram using Reverse Engineering feature of MySQL workbench.

CONCLUSION

- Successfully implemented a mini banking system using SQL

- Covered table creation, constraints, and SQL queries

- Can be extended with views, stored procedures, and user interfaces

Q & A

Thank you!

Questions and Discussions