**CSA0593**

**DATABASE MANAGEMENT SYSTEM**

*PRAKALYA P V*

*192311418*

*ASSIGNMENT - 2*

Create a database to manage accounts, transactions, loans, and customer details.

- Model tables for accounts, transactions, loans, and customers.

- Write stored procedures for processing deposits, withdrawals, and loan applications.

- Implement triggers to update account balances and loan statuses.

- Write SQL queries to analyse transaction volumes and account balances.

1). A database schema for managing accounts, transactions, loans, and customer details in a financial application. Each table will serve a specific purpose in organizing customer information, managing accounts, recording transactions, and tracking loans.

**Customers Table:-**

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(100) NOT NULL,

last\_name VARCHAR(100) NOT NULL,

date\_of\_birth DATE,

email VARCHAR(255) UNIQUE,

phone VARCHAR(15),

address VARCHAR(255),

city VARCHAR(100),

state VARCHAR(100),

zip\_code VARCHAR(10),

created\_on TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**Accounts Table:-**

CREATE TABLE Accounts (

account\_id INT PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT NOT NULL,

account\_type ENUM('savings', 'checking', 'credit') NOT NULL,

balance DECIMAL(15, 2) DEFAULT 0.00,

interest\_rate DECIMAL(5, 2), -- Annual interest rate in percentage

created\_on TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

**Transactions Table:-**

CREATE TABLE Transactions (

transaction\_id INT PRIMARY KEY AUTO\_INCREMENT,

account\_id INT NOT NULL,

transaction\_type ENUM('deposit', 'withdrawal', 'transfer', 'loan\_payment') NOT NULL,

amount DECIMAL(15, 2) NOT NULL,

transaction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

description VARCHAR(255),

FOREIGN KEY (account\_id) REFERENCES Accounts(account\_id)

);

**Loans Table:-**

CREATE TABLE Loans (

loan\_id INT PRIMARY KEY AUTO\_INCREMENT,

customer\_id INT NOT NULL,

account\_id INT, -- Optional link to an account if needed

loan\_type ENUM('personal', 'mortgage', 'auto', 'student', 'business') NOT NULL,

principal\_amount DECIMAL(15, 2) NOT NULL,

interest\_rate DECIMAL(5, 2) NOT NULL, -- Annual interest rate in percentage

loan\_start\_date DATE NOT NULL,

loan\_due\_date DATE NOT NULL,

balance DECIMAL(15, 2) DEFAULT 0.00, -- Outstanding balance

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id),

FOREIGN KEY (account\_id) REFERENCES Accounts(account\_id)

);

**Stored Procedure for Processing Deposits:-**

DELIMITER $$

CREATE PROCEDURE ProcessDeposit (

IN p\_account\_id INT,

IN p\_amount DECIMAL(15, 2),

IN p\_description VARCHAR(255)

)

BEGIN

-- Check if the deposit amount is valid (greater than zero)

IF p\_amount > 0 THEN

-- Insert transaction record

INSERT INTO Transactions (account\_id, transaction\_type, amount, description, transaction\_date)

VALUES (p\_account\_id, 'deposit', p\_amount, p\_description, NOW());

-- Update account balance

UPDATE Accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_account\_id;

ELSE

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Deposit amount must be greater than zero';

END IF;

END $$

DELIMITER ;

**Example Usage of Stored Procedures**

To test and use these stored procedures, you can call them with the appropriate parameters as shown below:

1. Process a Deposit:

CALL ProcessDeposit(1, 500.00, 'Salary deposit');

1. Process a withdrawal :

CALL ProcessWithdrawal(1, 200.00, 'ATM withdrawal');

3.apply for a loan:

CALL ApplyForLoan(1, 1, 'personal', 10000.00, 5.5, '2024-01-01', '2025-01-01');

These stored procedures ensure transactions are handled consistently and that key validations are in place, providing a strong foundation for banking-related database operations.

3**). Triggers implementation:-**

* Trigger to Update Account Balances After Transactions:

DELIMITER $$

CREATE TRIGGER UpdateAccountBalanceAfterTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

-- Check if the transaction type is a deposit

IF NEW.transaction\_type = 'deposit' THEN

-- Add the deposit amount to the account balance

UPDATE Accounts

SET balance = balance + NEW.amount

WHERE account\_id = NEW.account\_id;

-- Check if the transaction type is a withdrawal

ELSEIF NEW.transaction\_type = 'withdrawal' THEN

-- Subtract the withdrawal amount from the account balance

UPDATE Accounts

SET balance = balance - NEW.amount

WHERE account\_id = NEW.account\_id;

END IF;

END $$

DELIMITER ;

* Trigger to Update Loan Status When the Balance Reaches Zero:

DELIMITER $$

CREATE TRIGGER UpdateLoanStatusAfterPayment

AFTER UPDATE ON Loans

FOR EACH ROW

BEGIN

-- Check if the loan balance has been paid off and status is 'approved'

IF NEW.balance = 0 AND NEW.status = 'approved' THEN

-- Update loan status to 'paid'

UPDATE Loans

SET status = 'paid'

WHERE loan\_id = NEW.loan\_id;

END IF;

END $$

DELIMITER ;

* Trigger to Deduct Loan Payments from Loan Balance

DELIMITER $$

CREATE TRIGGER DeductLoanPayment

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

-- Check if the transaction type is a loan payment

IF NEW.transaction\_type = 'loan\_payment' THEN

-- Deduct the payment amount from the loan balance

UPDATE Loans

SET balance = balance - NEW.amount

WHERE loan\_id = NEW.account\_id; -- Assuming account\_id is used as loan\_id in this context

END IF;

END $$

DELIMITER ;

These triggers maintain account and loan balances automatically:

1. **UpdateAccountBalanceAfterTransaction**: Updates account balance after each deposit or withdrawal transaction.
2. **UpdateLoanStatusAfterPayment**: Updates loan status to "paid" when the loan balance reaches zero.
3. **DeductLoanPayment**: Reduces the loan balance when a loan payment is made.

These triggers automate balance updates and ensure that loan statuses are correctly tracked, providing real-time data consistency for accounts and loans.

4). Here are several SQL queries that can help analyze transaction volumes and account balances within the database. These queries provide insights into transaction trends, account types, and customer account statuses.

**Total Transaction Volume by Account Type:**

This query calculates the total transaction volume (sum of all deposits and withdrawals) for each account type, showing which types of accounts are most active.

SELECT a.account\_type, SUM(t.amount) AS total\_transaction\_volume

FROM Transactions t

JOIN Accounts a ON t.account\_id = a.account\_id

GROUP BY a.account\_type;

**Total Number of Transactions by Transaction Type:**

This query provides the number of transactions for each type (e.g., deposits, withdrawals, loan payments). It helps identify the most common types of transactions in the system.

SELECT transaction\_type, COUNT(\*) AS transaction\_count

FROM Transactions

GROUP BY transaction\_type;

**Average Account Balance by Account Type**

This query calculates the average balance across different account types (e.g., savings, checking), which can be useful for understanding typical balances held in each type

SELECT account\_type, AVG(balance) AS average\_balance

FROM Accounts

GROUP BY account\_type;

* These queries provide a comprehensive analysis of transaction volumes, account balances, and customer and loan activity, enabling efficient monitoring and insights into customer behaviour and financial trends.

**Conclusion:-**

In summary, the structured database and accompanying SQL queries provide a comprehensive system for effectively managing and analysing accounts, transactions, loans, and customer data. Through targeted insights into transaction volumes, account balances, and loan statuses, financial institutions can optimize operations, enhance customer service, and mitigate risks. Automated triggers ensure data accuracy and consistency, while stored procedures streamline core financial functions, promoting efficiency and reliability.

This database framework ultimately supports informed decision-making, personalized customer experiences, and a data-driven approach to financial management, all of which are crucial for maintaining competitive, customer-focused services in the financial industry.