**Initial Database Setup**

CREATE DATABASE bankdb;

USE bankdb;

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

DOB DATE,

Balance DECIMAL(10,2),

LastModified DATETIME

);

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

AccountType VARCHAR(20),

Balance DECIMAL(10,2),

LastModified DATETIME,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID INT PRIMARY KEY,

AccountID INT,

TransactionDate DATETIME,

Amount DECIMAL(10,2),

TransactionType VARCHAR(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount DECIMAL(10,2),

InterestRate DECIMAL(5,2),

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100),

Position VARCHAR(50),

Salary DECIMAL(10,2),

Department VARCHAR(50),

HireDate DATE

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'Tarun Karrthick', '1985-05-15', 1000.00, NOW()),

(2, 'Tarun', '1990-07-20', 1500.00, NOW());

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000.00, NOW()),

(2, 2, 'Checking', 1500.00, NOW());

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, NOW(), 200.00, 'Deposit'),

(2, 2, NOW(), 300.00, 'Withdrawal');

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Tarun Karrthick', 'AI Engineer', 98000.00, 'Customer Service', '2020-01-10'),

(2, 'Vignesh', 'Manager', 70000.00, 'HR', '2015-06-15'),

(3, 'Viswa', 'Developer', 60000.00, 'IT', '2017-03-20'),

(4, 'Sankar', 'Tester', 50000.00, 'QA', '2018-08-10'),

(5, 'Siva', 'Analyst', 55000.00, 'Finance', '2019-09-25');

**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates

**Solution:**

DELIMITER //

CREATE PROCEDURE ApplySeniorDiscount()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE customerId INT;

DECLARE dob DATE;

DECLARE age INT;

DECLARE customerCursor CURSOR FOR

SELECT CustomerID, DOB FROM Customers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN customerCursor;

readLoop: LOOP

FETCH customerCursor INTO customerId, dob;

IF done THEN

LEAVE readLoop;

END IF;

SET age = FLOOR(TIMESTAMPDIFF(MONTH, dob, CURDATE()) / 12);

IF age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = customerId;

END IF;

END LOOP;

CLOSE customerCursor;

END;

//

DELIMITER ;

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

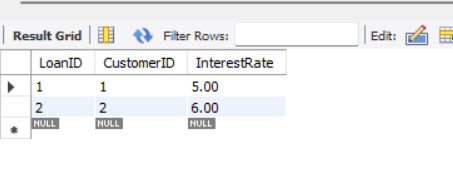
VALUES(1, 1, 5000.00, 5.00, CURDATE(), DATE\_ADD(CURDATE(), INTERVAL 24 MONTH)),

(2, 2, 7000.00, 6.00, CURDATE(), DATE\_ADD(CURDATE(), INTERVAL 36 MONTH));

CALL ApplySeniorDiscount();

SELECT LoanID, CustomerID, InterestRate FROM Loans;

**Output:**



**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Solution:**

ALTER TABLE Customers ADD IsVIP BOOLEAN DEFAULT FALSE;

DELIMITER //

CREATE PROCEDURE PromoteVIPCustomers()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE custId INT;

DECLARE balance DECIMAL(10,2);

DECLARE custCursor CURSOR FOR

SELECT CustomerID, Balance FROM Customers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN custCursor;

promoteLoop: LOOP

FETCH custCursor INTO custId, balance;

IF done THEN

LEAVE promoteLoop;

END IF;

IF balance > 10000 THEN

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = custId;

END IF;

END LOOP;

CLOSE custCursor;

END;

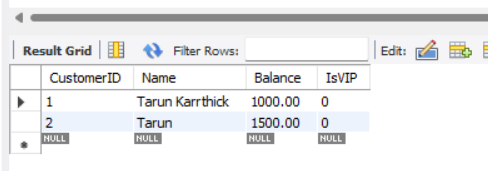
//

DELIMITER ;

CALL PromoteVIPCustomers();

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

**Output:**



**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**Solution:**

DROP PROCEDURE IF EXISTS SendLoanReminders;

DELIMITER //

CREATE PROCEDURE SendLoanReminders()

BEGIN

DELETE FROM LoanReminders;

INSERT INTO LoanReminders (Message)

SELECT

CONCAT('Reminder: Loan ID ', L.LoanID, ' for customer ', C.Name,

' (ID ', L.CustomerID, ') is due on ', L.EndDate)

FROM Loans L

JOIN Customers C ON L.CustomerID = C.CustomerID

WHERE L.EndDate BETWEEN CURDATE() AND DATE\_ADD(CURDATE(), INTERVAL 30 DAY);

SELECT \* FROM LoanReminders;

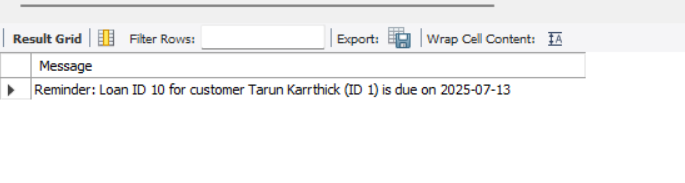
END;

//

DELIMITER ;

CALL SendLoanReminders();

**Output:**

****

**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

**Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Solution:**

SET SQL\_SAFE\_UPDATES = 0;

DELIMITER //

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

UPDATE Accounts SET Balance = Balance \* 1.01 WHERE AccountType = 'Savings';

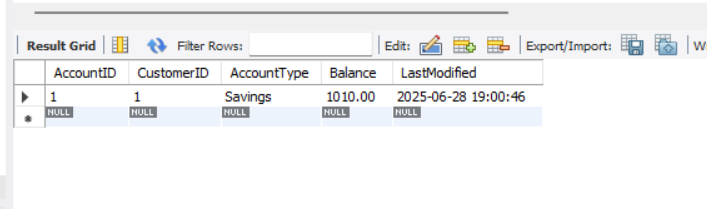
END //

DELIMITER ;

CALL ProcessMonthlyInterest();

SELECT \* FROM Accounts WHERE AccountType = 'Savings';

SET SQL\_SAFE\_UPDATES = 1;

**Output:**

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

**Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Solution:**

DELIMITER //

CREATE PROCEDURE UpdateEmployeeBonus (

IN dept\_name VARCHAR(100),

IN bonus\_percent DECIMAL(5,2)

)

BEGIN

UPDATE Employees SET Salary = Salary + (Salary \* bonus\_percent / 100)

WHERE Department = dept\_name;

END //

DELIMITER ;

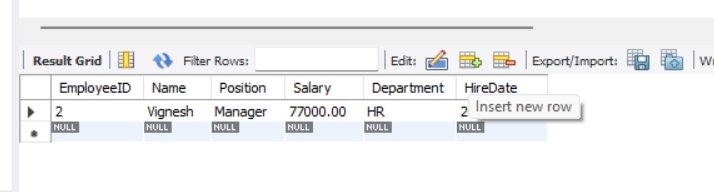
SET SQL\_SAFE\_UPDATES = 0;

CALL UpdateEmployeeBonus('HR', 10);

SELECT \* FROM Employees WHERE Department = 'HR';

SET SQL\_SAFE\_UPDATES = 1;

**Output:**

****

**Scenario 3:** Customers should be able to transfer funds between their accounts.

**Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**Solution:**

SET SQL\_SAFE\_UPDATES = 0;

DELIMITER //

CREATE PROCEDURE TransferFunds (

IN from\_account INT,

IN to\_account INT,

IN amount DECIMAL(10,2)

)

BEGIN

DECLARE from\_balance DECIMAL(10,2);

START TRANSACTION;

SELECT Balance INTO from\_balance FROM Accounts

WHERE AccountID = from\_account FOR UPDATE;

IF from\_balance >= amount THEN

UPDATE Accounts SET Balance = Balance - amount,

LastModified = NOW()

WHERE AccountID = from\_account;

UPDATE Accounts SET Balance = Balance + amount,

LastModified = NOW()

WHERE AccountID = to\_account;

COMMIT;

ELSE

ROLLBACK;

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Insufficient balance in source account';

END IF;

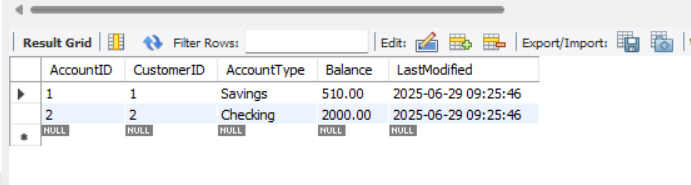
END //

DELIMITER ;

CALL TransferFunds(1, 2, 500.00);

SELECT \* FROM Accounts;

**Output:**

****