**Week-2**

**PLSQL**

**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.

**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 is VIP to TRUE for those with a balance over $10,000.

**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.

**Table Creation:**

**1.Customers Tab**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Age INT,

InterestRate DECIMAL(5,2),

Balance DECIMAL(10,2),

VIP VARCHAR(5) DEFAULT 'FALSE'

);

**2.Loans Table**

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**Data Insertion:**

-- Customers

INSERT INTO Customers VALUES (1, 'John Doe', 65, 7.50, 12000.00, 'FALSE');

INSERT INTO Customers VALUES (2, 'Jane Smith', 45, 8.00, 8000.00, 'FALSE');

INSERT INTO Customers VALUES (3, 'Bob Johnson', 70, 6.50, 15000.00, 'FALSE');

INSERT INTO Customers VALUES (4, 'Alice Brown', 30, 9.00, 9500.00, 'FALSE');

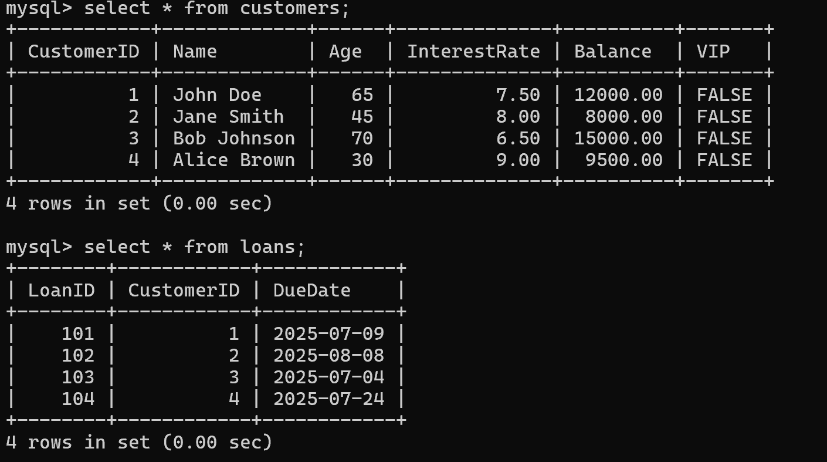
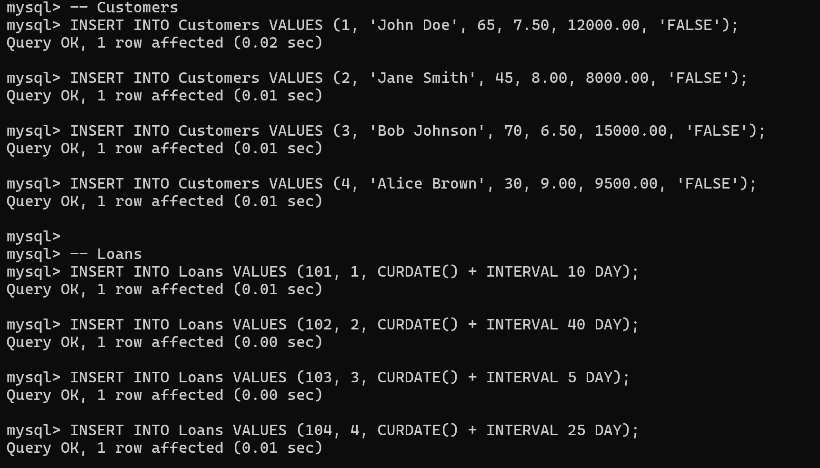
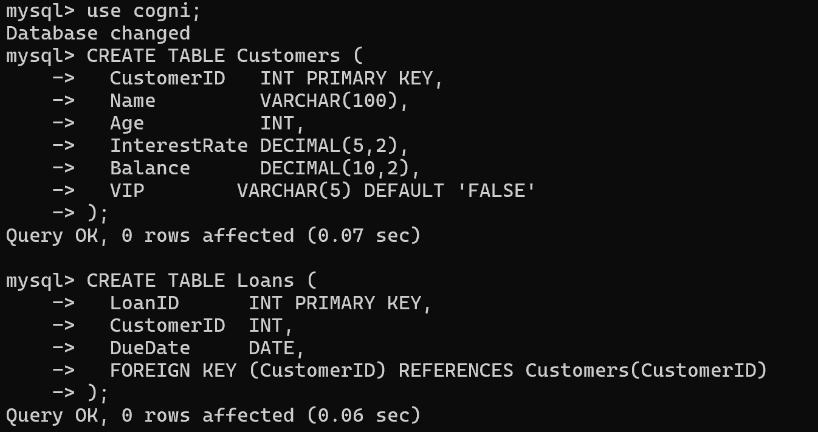
-- Loans

INSERT INTO Loans VALUES (101, 1, 10);

INSERT INTO Loans VALUES (102, 2, 40);

INSERT INTO Loans VALUES (103, 3, 5);

INSERT INTO Loans VALUES (104, 4, 25);



**PLSQL Queries:**

**1.Scenario 1**

DELIMITER $$

CREATE PROCEDURE ApplySeniorDiscount()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE v\_customerId INT;

DECLARE v\_interestRate DECIMAL(5,2);

DECLARE cur CURSOR FOR

SELECT CustomerID, InterestRate FROM Customers WHERE Age > 60;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_customerId, v\_interestRate;

IF done THEN

LEAVE read\_loop;

END IF;

UPDATE Customers

SET InterestRate = InterestRate - 1

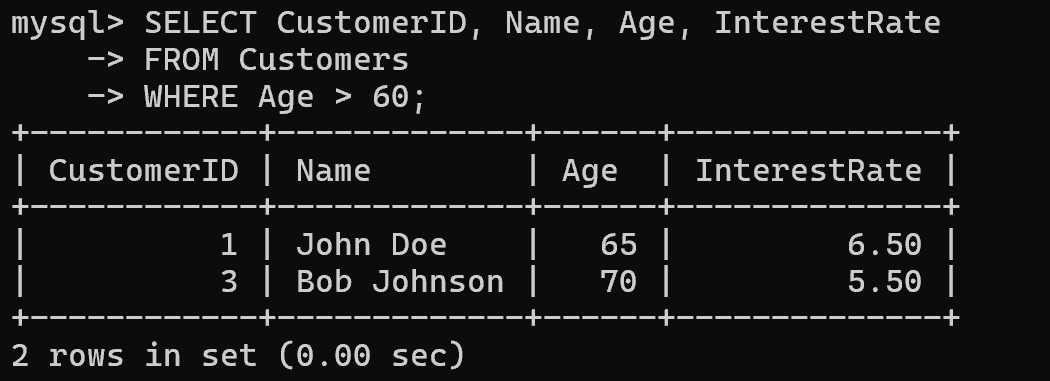
WHERE CustomerID = v\_customerId;

END LOOP;

CLOSE cur;

END$$

DELIMITER ;



**2.Scenario 2**

DELIMITER $$

CREATE PROCEDURE PromoteVIPs()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE v\_customerId INT;

DECLARE cur CURSOR FOR

SELECT CustomerID FROM Customers WHERE Balance > 10000;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_customerId;

IF done THEN

LEAVE read\_loop;

END IF;

UPDATE Customers

SET IsVIP = 'TRUE'

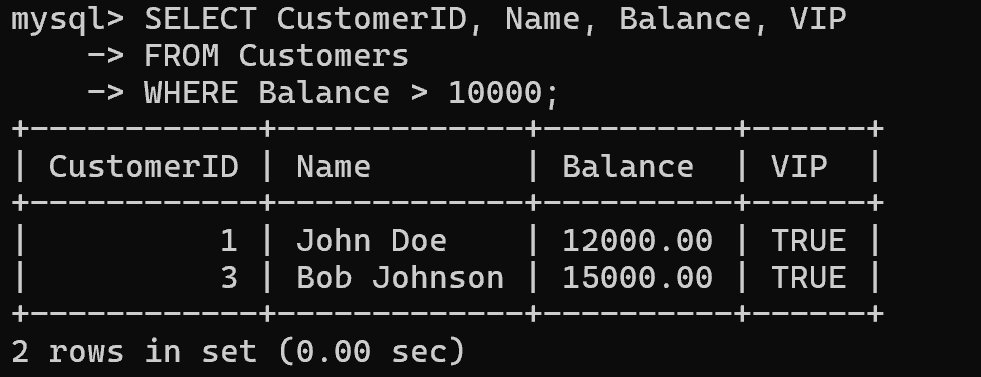
WHERE CustomerID = v\_customerId;

END LOOP;

CLOSE cur;

END$$

DELIMITER ;



**3.Scenario 3**

DELIMITER $$

CREATE PROCEDURE SendLoanReminders()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE v\_loanId INT;

DECLARE v\_customerId INT;

DECLARE v\_dueDate DATE;

DECLARE v\_name VARCHAR(100);

DECLARE cur CURSOR FOR

SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate BETWEEN CURDATE() AND CURDATE() + INTERVAL 30 DAY;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_loanId, v\_customerId, v\_dueDate;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT Name INTO v\_name FROM Customers WHERE CustomerID = v\_customerId;

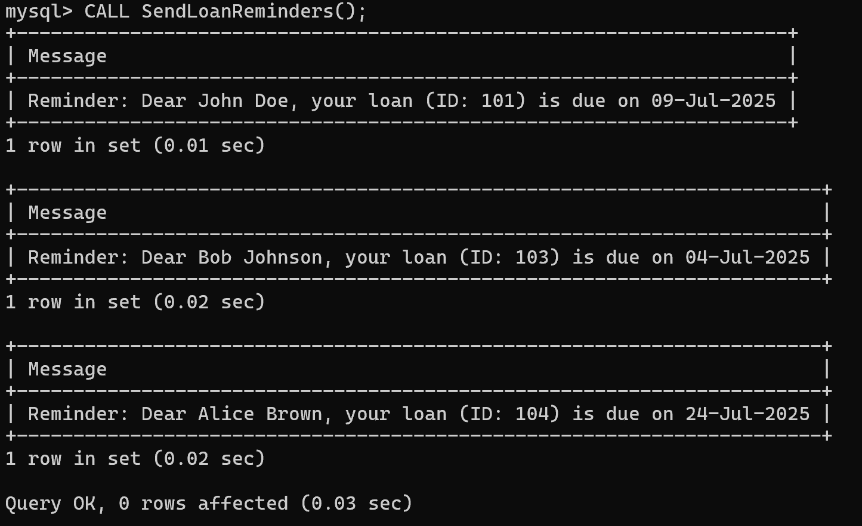
SELECT CONCAT('Reminder: Dear ', v\_name, ', your loan (ID: ', v\_loanId, ') is due on ', DATE\_FORMAT(v\_dueDate, '%d-%b-%Y')) AS Message;

END LOOP;

CLOSE cur;

END$$

DELIMITER ;

****

**Conclusion:**

In this workout we have looked into how we can automatically perform key banking operations by utilising control structures like cursors, loops, and conditionals. We successfully:  
  
Used at if then else logic to offer interest reductions for pensioners (over 60).Used a for loop to convert the customers to a VIP account if the account balance≈$1,000.Fetched and showed personalized loan reminders based on due dates and customer details.  
  
These control statements showed how to incorporate logic in SQL scripts for business-oriented workflows, which are expected to facilitate automatic, personalised and friendly service to customers in a banking domain.

**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.

**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.

**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.

**Table Creation:**

**1.Savings Account Table**

CREATE TABLE SavingsAccounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

Balance DECIMAL(10,2)

);

**2.Employees Table**

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(50),

Salary DECIMAL(10,2)

);

**3.Accounts Table**

CREATE TABLE Accounts (

AccountID INT PRIMARY KEY,

CustomerID INT,

Balance DECIMAL(10,2)

);

**Data Insertion:**

-- Savings Account

INSERT INTO SavingsAccounts VALUES (101, 1, 10000.00);

INSERT INTO SavingsAccounts VALUES (102, 2, 5000.00);

INSERT INTO SavingsAccounts VALUES (103, 3, 15000.00);

-- Employees

INSERT INTO Employees VALUES (1, 'Ravi', 'HR', 50000.00);

INSERT INTO Employees VALUES (2, 'Priya', 'Finance', 60000.00);

INSERT INTO Employees VALUES (3, 'Amit', 'HR', 52000.00);

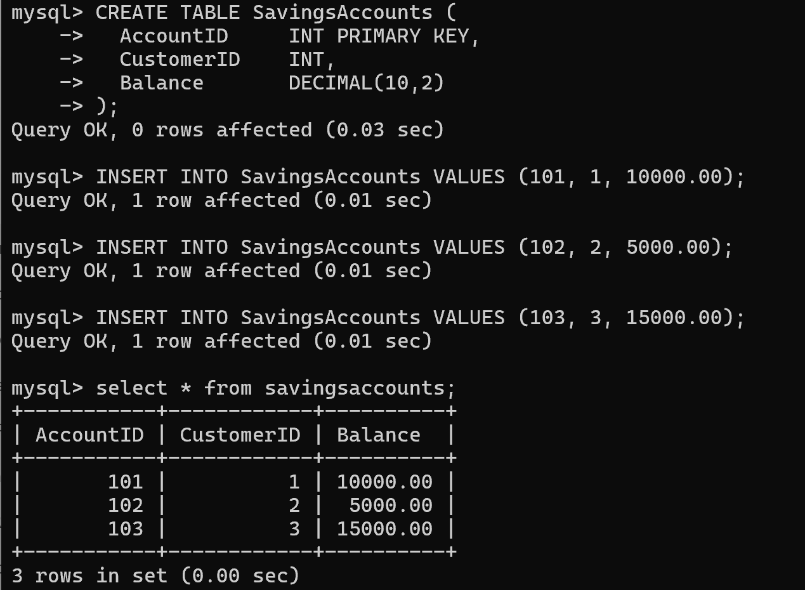
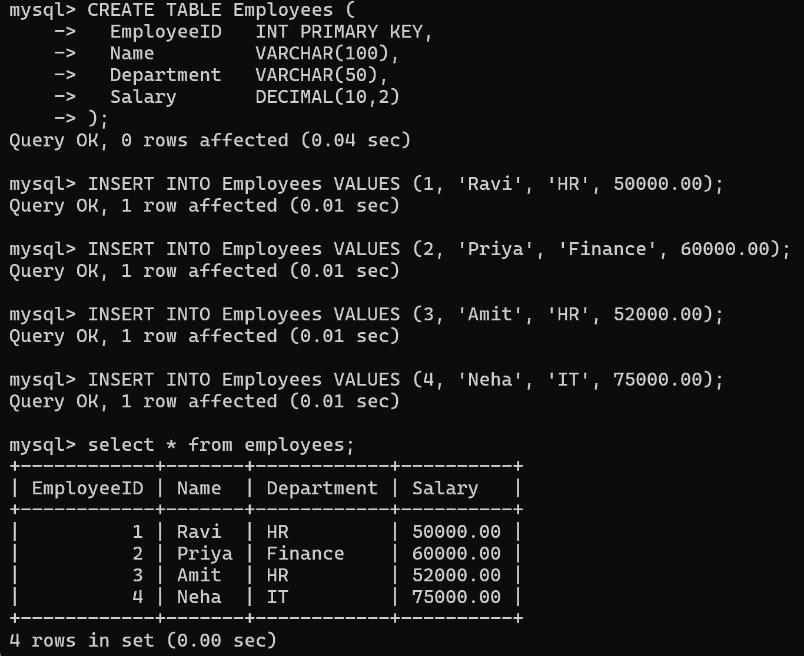
INSERT INTO Employees VALUES (4, 'Neha', 'IT', 75000.00);

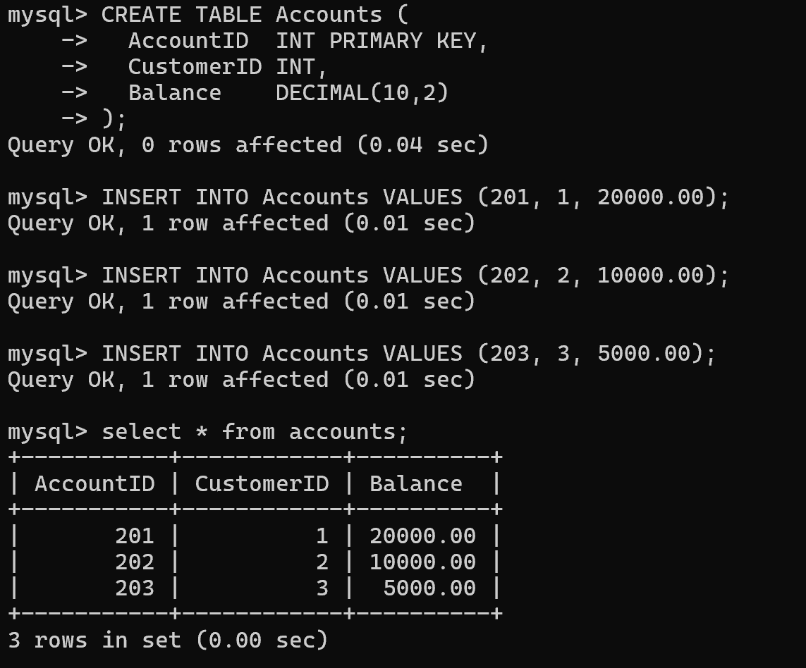
-- Accounts

INSERT INTO Accounts VALUES (201, 1, 20000.00);

INSERT INTO Accounts VALUES (202, 2, 10000.00);

INSERT INTO Accounts VALUES (203, 3, 5000.00);





**PLSQL Queries:**

**1.Scenario 1**

DELIMITER $$

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

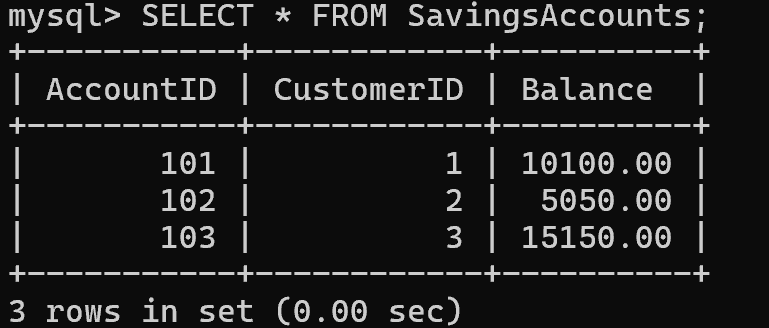
UPDATE SavingsAccounts

SET Balance = Balance \* 1.01;

END$$

DELIMITER ;

CALL ProcessMonthlyInterest();



**2.Scenario 2**

DELIMITER $$

CREATE PROCEDURE UpdateEmployeeBonus(IN dept\_name VARCHAR(50), IN bonus\_percent DECIMAL(5,2))

BEGIN

UPDATE Employees

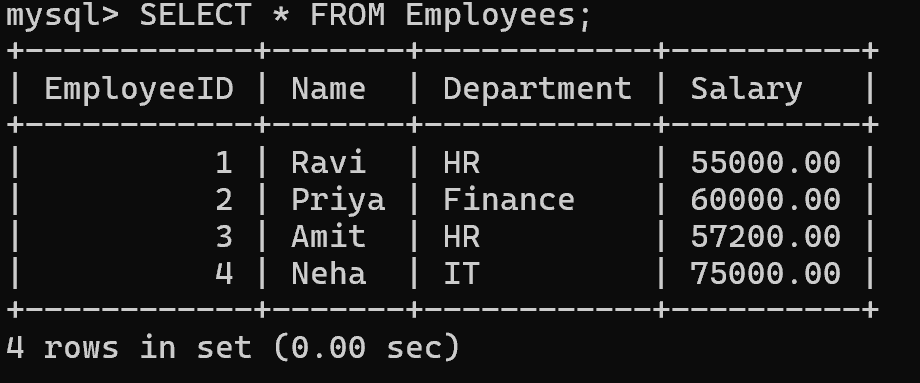
SET Salary = Salary + (Salary \* bonus\_percent / 100)

WHERE Department = dept\_name;

END$$

DELIMITER ;

CALL UpdateEmployeeBonus('HR', 10);



**3.Scenario 3**

DELIMITER $$

CREATE PROCEDURE TransferFunds(

IN from\_account INT,

IN to\_account INT,

IN amount DECIMAL(10,2)

)

BEGIN

DECLARE from\_balance DECIMAL(10,2);

-- Get balance of source account

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_account;

-- Check if sufficient funds

IF from\_balance >= amount THEN

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_account;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_account;

ELSE

SIGNAL SQLSTATE '45000'

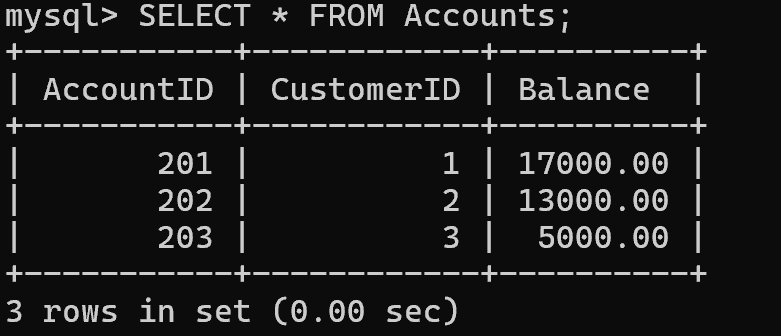
SET MESSAGE\_TEXT = 'Insufficient balance for transfer';

END IF;

END$$

DELIMITER ;

CALL TransferFunds(201, 202, 3000);



**Conclusion:**

In this exercise, you developed stored procedures to encapsulate any reusable banking logic. We developed procedures to:  
  
1.Work out and accrue monthly interest on all savings balances.  
  
2.Update employees Salaries based on the department based performance bonus.  
  
3.Process safe and verified transfers of funds between customer accounts.  
  
We improved modularity, maintainability, and enforced data integrity with the use of stored procedures, reducing any redundant code. These are also a first step towards implementing business logic in a more complex way within enterprise-level database applications.

**-- THE END --**