**Week - 2**

**PL/SQL programming**

**Exercise 1: Control Structures**

**Project Name** : ControlStructures

**Exercise**  : Implementing PL/SQL Using Control Structures

**Name**  : Abhignya Gitti

**Superset Id** : 6374632

* **Introduction :**

This document presents PL/SQL solutions using control structures for a banking database. The goal is to manipulate customer and loan records using conditional and iterative logic.

* **Implementation :**

**Step 1:** Establish New Connection

**Step 2:** Create Tables and Insert Values

CREATE DATABASE bank\_control;

USE bank\_control;

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

Age INT,

Balance DECIMAL(10,2),

IsVIP BOOLEAN DEFAULT FALSE

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

InterestRate DECIMAL(5,2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES

(1, 'John', 65, 12000.00, FALSE),

(2, 'Alice', 45, 8000.00, FALSE),

(3, 'Bob', 70, 15000.00, FALSE);

INSERT INTO Loans VALUES

(101, 1, 7.5, CURDATE() + INTERVAL 10 DAY),

(102, 2, 8.0, CURDATE() + INTERVAL 35 DAY),

(103, 3, 6.5, CURDATE() + INTERVAL 5 DAY);

**SCENARIO1:**

DELIMITER $$

CREATE PROCEDURE ApplySeniorDiscount()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE custId INT;

DECLARE cur CURSOR FOR

SELECT CustomerID FROM Customers WHERE Age > 60;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO custId;

IF done THEN

LEAVE read\_loop;

END IF;

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = custId;

END LOOP;

CLOSE cur;

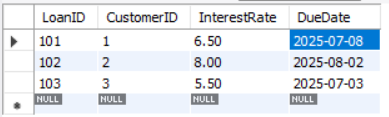
END$$

DELIMITER ;

CALL ApplySeniorDiscount();

SELECT \* FROM Loans;

**Loans Table After Scenario 1:**



**Scenario2:**

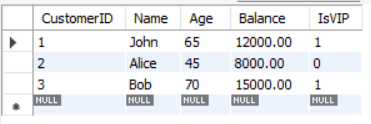
UPDATE Customers

SET IsVIP = TRUE

WHERE Balance > 10000;

SELECT \* FROM Customers;

**Outputs :**



**SCENARIO 3:**

DELIMITER $$

CREATE PROCEDURE SendLoanReminders()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE custName VARCHAR(100);

DECLARE dueDate DATE;

DECLARE cur CURSOR FOR

SELECT C.Name, L.DueDate

FROM Customers C

JOIN Loans L ON C.CustomerID = L.CustomerID

WHERE L.DueDate <= CURDATE() + INTERVAL 30 DAY;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

FETCH cur INTO custName, dueDate;

WHILE NOT done DO

SELECT CONCAT('Reminder: ', custName, ', your loan is due on ', dueDate) AS Reminder;

FETCH cur INTO custName, dueDate;

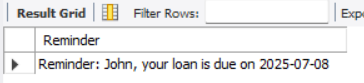
END WHILE;

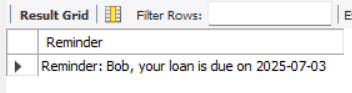
CLOSE cur;

END$$

DELIMITER ;

CALL SendLoanReminders();





**Exercise 2 : Stored Procedures**

**Project Name** : Stored Procedures(Banking Example)

**Exercise**  : Implementing PL/SQL Using Stored Procedures

**Name**  : Abhignya Gitti

**Superset Id** : 6374632

* **Introduction :**

This exercise demonstrates the use of **PL/SQL stored procedures** to automate key banking operations. It includes:

* Applying monthly interest to savings accounts
* Updating employee salaries with performance bonuses
* Transferring funds between customer accounts

These scenarios help develop skills in writing procedures, handling input parameters, and applying conditional logic in PL/SQL.

* **Implementation :**

**Step 1:** Create Tables and Insert Values

CREATE DATABASE bankdb\_ex3;

USE bankdb\_ex3;

**SCENARIO 1**

CREATE TABLE SavingsAccounts (

AccountID INT PRIMARY KEY,

CustomerName VARCHAR(100),

Balance DECIMAL(10,2)

);

INSERT INTO SavingsAccounts VALUES

(1, 'John', 10000.00),

(2, 'Alice', 15000.00),

(3, 'Bob', 20000.00);

DELIMITER $$

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* 0.01);

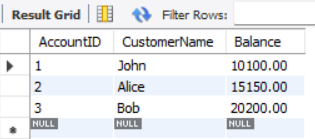
END$$

DELIMITER ;

CALL ProcessMonthlyInterest();

SELECT \* FROM SavingsAccounts;

**Output:**



**Scenario 2: Bonus Scheme for Employees**

CREATE TABLE Employees (

EmpID INT PRIMARY KEY,

Name VARCHAR(100),

Department VARCHAR(100),

Salary DECIMAL(10,2)

);

DELIMITER $$

CREATE PROCEDURE UpdateEmployeeBonus(

IN deptName VARCHAR(100),

IN bonusPercent DECIMAL(5,2)

)

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonusPercent / 100)

WHERE LOWER(Department) = LOWER(deptName);

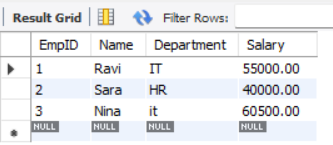
END$$

DELIMITER ;

CALL UpdateEmployeeBonus('IT', 10);

SELECT \* FROM Employees;

**OUTPUT:**

****

**SCENARIO 3:**

CREATE TABLE Accounts (

AccountNumber INT PRIMARY KEY,

HolderName VARCHAR(100),

Balance DECIMAL(10,2)

);

INSERT INTO Accounts VALUES

(101, 'John', 8000.00),

(102, 'Alice', 6000.00);

DELIMITER $$

CREATE PROCEDURE TransferFunds(

IN fromAccount INT,

IN toAccount INT,

IN amount DECIMAL(10,2)

)

BEGIN

DECLARE fromBalance DECIMAL(10,2);

SELECT Balance INTO fromBalance

FROM Accounts

WHERE AccountNumber = fromAccount;

IF fromBalance >= amount THEN

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountNumber = fromAccount;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountNumber = toAccount;

ELSE

SIGNAL SQLSTATE '45000'

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

END IF;

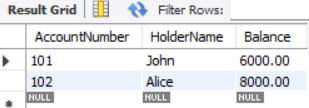
END$$

DELIMITER ;

CALL TransferFunds(101, 102, 2000);

SELECT \* FROM Accounts;

**OUTPUT:**

****