**WEEK – 2(Hands-on)**

**PLSQL Exercise**

**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.

SET SERVEROUTPUT ON;

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

InterestRate NUMBER,

IsVIP VARCHAR2(5)

);

INSERT INTO Customers VALUES (1, 'Alice', 65, 15000, 7.5, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 9000, 8.0, 'FALSE');

INSERT INTO Customers VALUES (3, 'Charlie', 70, 12000, 6.5, 'FALSE');

COMMIT;

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

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

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

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

COMMIT;

DECLARE

CURSOR cust\_cursor IS

SELECT CustomerID, Name

FROM Customers

WHERE Age > 60;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

UPDATE Customers

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied for: ' || cust\_rec.Name);

END LOOP;

COMMIT;

END;

/

DECLARE

CURSOR vip\_cursor IS

SELECT CustomerID, Name, Balance

FROM Customers;

BEGIN

FOR vip\_rec IN vip\_cursor LOOP

IF vip\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = vip\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Promoted to VIP: ' || vip\_rec.Name);

END IF;

END LOOP;

COMMIT;

END;

/

DECLARE

CURSOR loan\_cursor IS

SELECT l.LoanID, l.CustomerID, l.DueDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

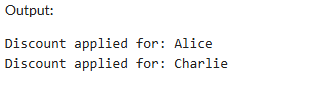
DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || loan\_rec.Name || ' has a loan (ID: ' || loan\_rec.LoanID || ') due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**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.

**Program:**

SET SERVEROUTPUT ON;

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

InterestRate NUMBER,

IsVIP VARCHAR2(5)

);

INSERT INTO Customers VALUES (1, 'Alice', 65, 15000, 7.5, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 9000, 8.0, 'FALSE');

INSERT INTO Customers VALUES (3, 'Charlie', 70, 12000, 6.5, 'FALSE');

COMMIT;

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

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

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

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

COMMIT;

DECLARE

CURSOR vip\_cursor IS

SELECT CustomerID, Name, Balance

FROM Customers;

BEGIN

FOR vip\_rec IN vip\_cursor LOOP

IF vip\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = vip\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Promoted to VIP: ' || vip\_rec.Name);

END IF;

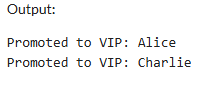
END LOOP;

COMMIT;

END;

/

**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.

**Program:**

SET SERVEROUTPUT ON;

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

InterestRate NUMBER,

IsVIP VARCHAR2(5)

);

INSERT INTO Customers VALUES (1, 'Alice', 65, 15000, 7.5, 'FALSE');

INSERT INTO Customers VALUES (2, 'Bob', 45, 9000, 8.0, 'FALSE');

INSERT INTO Customers VALUES (3, 'Charlie', 70, 12000, 6.5, 'FALSE');

COMMIT;

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

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

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

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

COMMIT;

DECLARE

CURSOR loan\_cursor IS

SELECT l.LoanID, l.CustomerID, l.DueDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

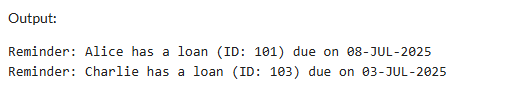
DBMS\_OUTPUT.PUT\_LINE('Reminder: ' || loan\_rec.Name || ' has a loan (ID: ' || loan\_rec.LoanID || ') due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/

**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.

**Program:**

SET SERVEROUTPUT ON;

-- Table for Savings Accounts

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

-- Table for Employees

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER

);

-- Table for Bank Accounts (for transfers)

CREATE TABLE BankAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE SavingsAccounts

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

DBMS\_OUTPUT.PUT\_LINE('Monthly interest of 1% applied to all savings accounts.');

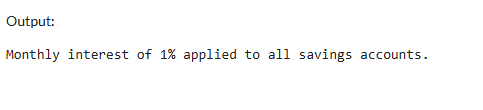
COMMIT;

END;

/

EXEC ProcessMonthlyInterest;

**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.

**Program:**

SET SERVEROUTPUT ON;

-- Table for Savings Accounts

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

-- Table for Employees

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER

);

-- Table for Bank Accounts (for transfers)

CREATE TABLE BankAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept\_name IN VARCHAR2,

bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = dept\_name;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || bonus\_percent || '% applied to department: ' || dept\_name);

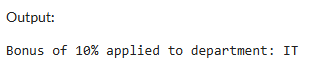
COMMIT;

END;

/

EXEC UpdateEmployeeBonus('IT', 10);

**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.

**Program:**

SET SERVEROUTPUT ON;

-- Table for Savings Accounts

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

-- Table for Employees

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER

);

-- Table for Bank Accounts (for transfers)

CREATE TABLE BankAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amount IN NUMBER

) IS

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance FROM BankAccounts WHERE AccountID = from\_acc FOR UPDATE;

IF from\_balance < amount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in Account ' || from\_acc);

ELSE

UPDATE BankAccounts

SET Balance = Balance - amount

WHERE AccountID = from\_acc;

UPDATE BankAccounts

SET Balance = Balance + amount

WHERE AccountID = to\_acc;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount || ' from Account ' || from\_acc || ' to Account ' || to\_acc);

COMMIT;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('One or both accounts not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END;

/

EXEC TransferFunds(101, 102, 500);

**Output:**

