**DIGITAL NURTURE 4.0 JavaFSE**

**WEEK 2**

**PLSQL\_Exercises**

**Exercise 1: Control Structures**

1. **Scenario 1:  
   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.**
2. **Scenario 2:  
   Write a PL/SQL block that iterates through all customers and sets a flag isVIP to TRUE ('Y') for those with a balance over $10,000.**
3. **Scenario 3:  
   Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.**

**Tables used and their Schema:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**Data Insertions before update:**

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

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

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

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

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

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

VALUES (2, 2, 7000, 6, SYSDATE, SYSDATE + 15);

COMMIT;

**PLSQL Code for Scenario 1:**

SET SERVEROUTPUT ON;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 1: Age-based Interest Discount ---');

FOR cust\_rec IN (

SELECT c.CustomerID, c.Name, c.DOB

FROM Customers c

) LOOP

DECLARE

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB)/12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(

'Applied 1% interest discount for customer: '

|| cust\_rec.Name

|| ', Age: ' || v\_age

);

END IF;

END;

END LOOP;

COMMIT;

END;

**PLSQL Code for Scenario 2:**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 2: VIP Promotion ---');

FOR cust\_rec IN (

SELECT CustomerID, Name, Balance

FROM Customers

) LOOP

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE(

'Customer promoted to VIP: '

|| cust\_rec.Name

);

ELSE

DBMS\_OUTPUT.PUT\_LINE(

'Customer not eligible for VIP: '

|| cust\_rec.Name

);

END IF;

END LOOP;

COMMIT;

END;

**PLSQL Code for Scenario 3:**

BEGIN

DBMS\_OUTPUT.PUT\_LINE('--- Scenario 3: Loan Due Reminders ---');

FOR rec IN (

SELECT l.LoanID, l.EndDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan ID ' || rec.LoanID

|| ' for customer ' || rec.Name

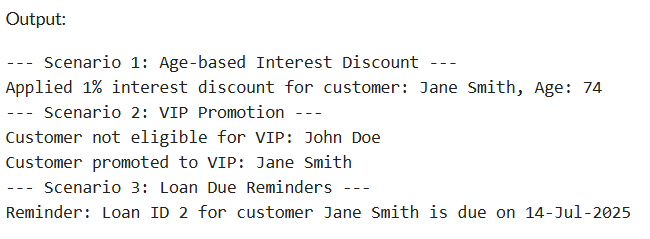
|| ' is due on ' || TO\_CHAR(rec.EndDate, 'DD-Mon-YYYY')

);

END LOOP;

END;

**Output after the execution of the Scenarios:**



**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.**

**Tables used and their Schema:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);  
**Data Insertions before update:**

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

COMMIT;

**PLSQL Code for Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE LOWER(AccountType) = 'savings';

COMMIT;

END;

**PLSQL Code for Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE Department = p\_department;

COMMIT;

END;

**PLSQL Code for Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_source\_account IN NUMBER,

p\_target\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_source\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_source\_balance FROM Accounts WHERE AccountID = p\_source\_account;

IF v\_source\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account');

ELSE

UPDATE Accounts SET Balance = Balance - p\_amount WHERE AccountID = p\_source\_account;

UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_target\_account;

COMMIT;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Account not found');

END;

**Calling the Procedures in Scenario 1, 2 and 3:**

SET SERVEROUTPUT ON;

BEGIN

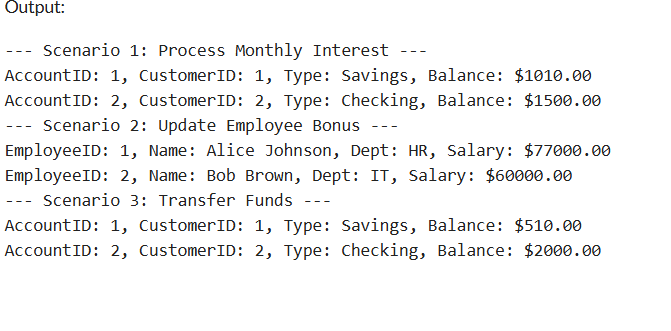
ProcessMonthlyInterest;

UpdateEmployeeBonus('HR', 10);

TransferFunds(1, 2, 500);

END;

**Output after the Execution of these scenarios:**

****