PL/SQL ASSIGNMENT WEEK 3

**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 IsVIP 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.

**Ans:  
First we create the database for the above question to workout our PL/SQL commands**

**CREATE TABLE Customers (**

**CustomerID NUMBER PRIMARY KEY,**

**Name VARCHAR2(100),**

**Age NUMBER,**

**Balance NUMBER(12,2),**

**IsVIP CHAR(1) DEFAULT 'N'**

**);**

**CREATE TABLE Loans (**

**LoanID NUMBER PRIMARY KEY,**

**CustomerID NUMBER REFERENCES Customers(CustomerID),**

**InterestRate NUMBER(5,2),**

**DueDate DATE**

**);**

**-- Insert some dummy customers**

**INSERT INTO Customers (CustomerID, Name, Age, Balance)**

**VALUES (1, 'John Smith', 65, 12000);**

**INSERT INTO Customers (CustomerID, Name, Age, Balance)**

**VALUES (2, 'Mary Johnson', 45, 8000);**

**INSERT INTO Customers (CustomerID, Name, Age, Balance)**

**VALUES (3, 'Robert Brown', 70, 15000);**

**INSERT INTO Customers (CustomerID, Name, Age, Balance)**

**VALUES (4, 'Emily Davis', 30, 9500);**

**-- Insert dummy loans**

**INSERT INTO Loans (LoanID, CustomerID, InterestRate, DueDate)**

**VALUES (101, 1, 7.50, SYSDATE + 20);**

**INSERT INTO Loans (LoanID, CustomerID, InterestRate, DueDate)**

**VALUES (102, 2, 8.00, SYSDATE + 40);**

**INSERT INTO Loans (LoanID, CustomerID, InterestRate, DueDate)**

**VALUES (103, 3, 9.00, SYSDATE + 10);**

**INSERT INTO Loans (LoanID, CustomerID, InterestRate, DueDate)**

**VALUES (104, 4, 6.50, SYSDATE + 25);**

**COMMIT;**

**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.

**DECLARE**

**CURSOR cur\_senior\_loans IS**

**SELECT l.LoanID, l.InterestRate**

**FROM Loans l**

**JOIN Customers c ON l.CustomerID = c.CustomerID**

**WHERE c.Age > 60;**

**BEGIN**

**FOR rec IN cur\_senior\_loans LOOP**

**UPDATE Loans**

**SET InterestRate = InterestRate - 1**

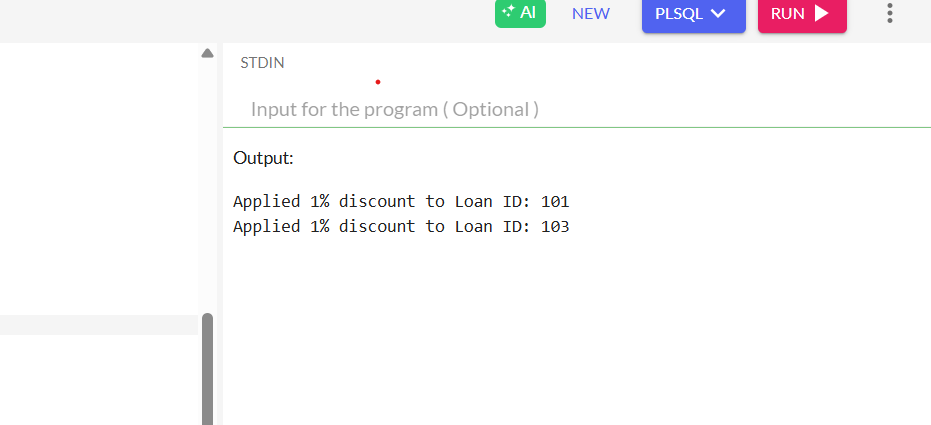
**WHERE LoanID = rec.LoanID;**

**DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount to Loan ID: ' || rec.LoanID);**

**END LOOP;**

**COMMIT;**

**END;/**

****

**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.

DECLARE

CURSOR cur\_vip IS

SELECT CustomerID

FROM Customers

WHERE Balance > 10000;

BEGIN

FOR rec IN cur\_vip LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = rec.CustomerID;

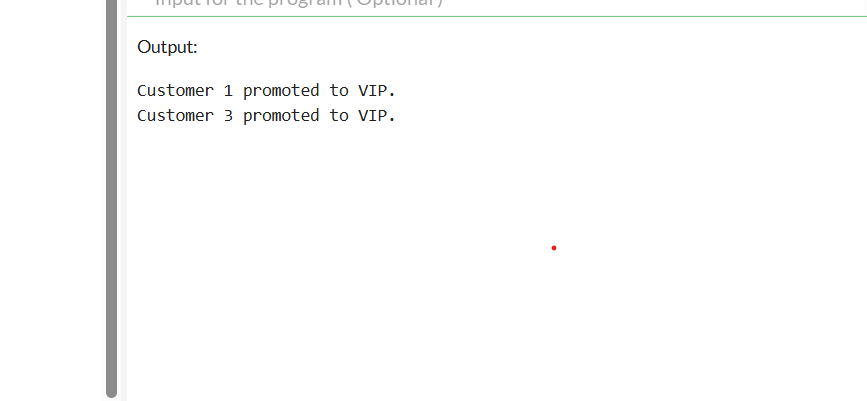
DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CustomerID || ' promoted to VIP.');

END LOOP;

COMMIT;

END;

/

****

**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.

DECLARE

CURSOR cur\_due\_loans IS

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate <= SYSDATE + 30;

BEGIN

FOR rec IN cur\_due\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Dear ' || rec.Name ||

', your Loan ID ' || rec.LoanID ||

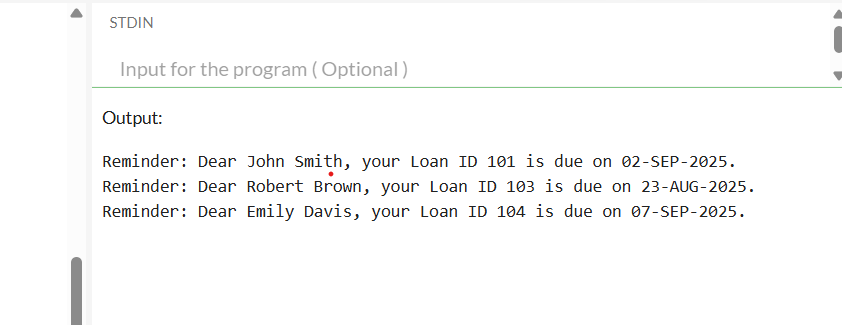
' is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY') || '.'

);

END LOOP;

END;

/



**Exercise 3: Stored Procedures**

**Ans: First, modifying the database for the new question and add the savings\_Account section to it.**

-- Savings Accounts Table

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER REFERENCES Customers(CustomerID),

Balance NUMBER(12,2)

);

-- Employees Table

CREATE TABLE Employees (

EmpID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER(10,2),

PerformanceScore NUMBER(3) -- Example: out of 100

);

-- Insert Dummy Savings Accounts

INSERT INTO SavingsAccounts (AccountID, CustomerID, Balance)

VALUES (201, 1, 5000);

INSERT INTO SavingsAccounts (AccountID, CustomerID, Balance)

VALUES (202, 2, 8000);

INSERT INTO SavingsAccounts (AccountID, CustomerID, Balance)

VALUES (203, 3, 15000);

INSERT INTO SavingsAccounts (AccountID, CustomerID, Balance)

VALUES (204, 4, 3000);

-- Insert Dummy Employees

INSERT INTO Employees (EmpID, Name, Department, Salary, PerformanceScore)

VALUES (1, 'Alice White', 'HR', 40000, 85);

INSERT INTO Employees (EmpID, Name, Department, Salary, PerformanceScore)

VALUES (2, 'Bob Green', 'IT', 50000, 90);

INSERT INTO Employees (EmpID, Name, Department, Salary, PerformanceScore)

VALUES (3, 'Charlie Black', 'IT', 60000, 75);

INSERT INTO Employees (EmpID, Name, Department, Salary, PerformanceScore)

VALUES (4, 'Diana Gray', 'Finance', 55000, 88);

**COMMIT;**

**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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

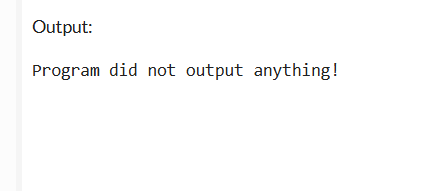
UPDATE SavingsAccounts

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

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

COMMIT;

END;

/  


**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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* (p\_BonusPercent / 100))

WHERE Department = p\_Department;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_BonusPercent || '% applied to department: ' || p\_Department);

COMMIT;

END;

/

**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.

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_SourceAccount IN NUMBER,

p\_TargetAccount IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_SourceBalance NUMBER;

BEGIN

SELECT Balance INTO v\_SourceBalance

FROM SavingsAccounts

WHERE AccountID = p\_SourceAccount

FOR UPDATE;

IF v\_SourceBalance < p\_Amount THEN

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

END IF;

UPDATE SavingsAccounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_SourceAccount;

UPDATE SavingsAccounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_TargetAccount;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_Amount || ' from Account ' || p\_SourceAccount ||

' to Account ' || p\_TargetAccount);

COMMIT;

END;

/