**Exercise 1: Control Structures**

Sparshak Ghosh

Mandatory Hands-on

**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.**

**ANSWER:**

DECLARE

CURSOR loan IS

SELECT l.LoanID, l.interestRate FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE MONTHS\_BETWEEN (SYSDATE , c.DOB) / 12 > 60;

BEGIN

FOR entry IN loan LOOP

UPDATE Loans SET InterestRate=InterestRate\*0.99 WHERE LoanID=entry.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.**

**ANSWER:**

**As the customers table didn’t have a column to store the VIP status, I first added a column named 'isVIP' to store the values “TRUE” or “FALSE”.**

ALTER TABLE Customers ADD isVIP VARCHAR2(5);

DECLARE

CURSOR VIP IS SELECT CustomerID FROM Customers WHERE Balance > 10000;

BEGIN

FOR entry IN VIP LOOP

UPDATE Customers SET isVIP = 'TRUE', LastModified = SYSDATE

WHERE CustomerID = entry.CustomerID;

END LOOP;

COMMIT;

END;

**Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.**

Mandatory Hands-on

Sparshak Ghosh

* + **Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.**

**ANSWER:**

DECLARE

CURSOR loans\_due IS

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;

BEGIN

FOR entry IN loans\_due LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || entry.LoanID ||' for customer ' || entry.Name || ' is due on ' || TO\_CHAR(entry.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

**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.**

**ANSWER:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts SET Balance = Balance \* 1.01, LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

END;

**Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.**

Mandatory Hands-on

Sparshak Ghosh

* + **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.**

**ANSWER:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER) AS

BEGIN

UPDATE Employees SET Salary = Salary + (Salary \* p\_BonusPercent / 100) WHERE 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.**

**ANSWER:**

CREATE OR REPLACE PROCEDURE TransferFunds (p\_FromAccountID IN NUMBER, p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER) AS v\_FromBalance NUMBER;

BEGIN

SELECT Balance INTO v\_FromBalance FROM Accounts

WHERE AccountID = p\_FromAccountID FOR UPDATE;

IF v\_FromBalance < p\_Amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_Amount, LastModified = SYSDATE

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts SET Balance = Balance + p\_Amount, LastModified = SYSDATE

WHERE AccountID = p\_ToAccountID;

COMMIT;

END;