**PL/SQL EXERCISES**

**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 loop block that loops throught all customers, check their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

BEGIN

FOR rec IN (

SELECT l.LoanID, l.InterestRate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

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

) LOOP

-- Apply 1% discount

UPDATE Loans

SET InterestRate = rec.InterestRate - (rec.InterestRate \* 0.01)

WHERE LoanID = rec.LoanID;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to interest rates for customers above 60.');

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.

**First we have to add a isVIP column in our table:**

ALTER TABLE Customers

ADD IsVIP CHAR(1);

**PL/SQL BLOCK :**

BEGIN

FOR rec IN (

SELECT CustomerID, Balance

FROM Customers

) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = rec.CustomerID;

ELSE

UPDATE Customers

SET IsVIP = 'N'

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('VIP flags updated for all customers based on balance.');

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.

BEGIN

FOR rec IN (

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

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: Dear ' || rec.Name ||

', your loan (Loan ID: ' || rec.LoanID ||

') is due on ' || TO\_CHAR(rec.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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR rec IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = 'Savings'

) LOOP

UPDATE Accounts

SET Balance = rec.Balance + (rec.Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = rec.AccountID;

END LOOP;

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

END;

/

**To use this procedure:**

BEGIN

ProcessMonthlyInterest

END;

/



**SCENARIO 2:**

**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 Employees.Department%TYPE,

p\_bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_percent || '% applied to department: ' || p\_department);

END;

/

**To use this procedure:**

BEGIN

UpdateEmployeeBonus('IT', 10); -- Adds 10% bonus to IT department employees

END;

/



**SCENARIO 3:**

**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\_source\_account\_id IN Accounts.AccountID%TYPE,

p\_target\_account\_id IN Accounts.AccountID%TYPE,

p\_amount IN NUMBER

) AS

v\_source\_balance NUMBER;

v\_target\_balance NUMBER;

BEGIN

-- Check if source account exists and get balance

SELECT Balance INTO v\_source\_balance

FROM Accounts

WHERE AccountID = p\_source\_account\_id;

-- Check if target account exists (optional, forces exception if not found)

SELECT Balance INTO v\_target\_balance

FROM Accounts

WHERE AccountID = p\_target\_account\_id;

-- Validate sufficient funds

IF v\_source\_balance < p\_amount THEN

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

END IF;

-- Perform transfer

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_source\_account\_id;

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_target\_account\_id;

-- Inserting transaction records

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (Transactions\_seq.NEXTVAL, p\_source\_account\_id, SYSDATE, p\_amount, 'Debit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (Transactions\_seq.NEXTVAL, p\_target\_account\_id, SYSDATE, p\_amount, 'Credit');

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Rs.' || p\_amount || ' transferred from Account ' || p\_source\_account\_id || ' to Account ' || p\_target\_account\_id);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20002, 'One or both accounts do not exist.');

WHEN OTHERS THEN

ROLLBACK;

RAISE;

END;

/

CREATE SEQUENCE Transactions\_seq START WITH 3 INCREMENT BY 1;

--Calling the procedure

BEGIN

TransferFunds(1, 2, 500);

END;

/

