**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.

**The above scenario can be executed using PL/SQL using the following code:**

We initially, create the tables – ‘Customers’ and ‘Loans’.

-- Customer Table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP CHAR(1) DEFAULT 'N'

);

-- Loans Table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER,

DueDate DATE

);

Next, we add some sample values to the respective tables to verify the PL/SQL operations mentioned.

-- Insert values to verify the outputs

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'N');

INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 'N');

INSERT INTO Customers VALUES (3, 'Charlie', 70, 15000, 'N');

INSERT INTO Loans VALUES (101, 1, 5.5, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, 6.0, SYSDATE + 40);

INSERT INTO Loans VALUES (103, 3, 7.0, SYSDATE + 20);

COMMIT;

We insert the values into both the tables i.e. ‘Customers’ and ‘Loans’. Also, we use the ‘COMMIT’ command in order to save the insertion operation, respectively.

**The PL/SQL code for the scenario 1:**

BEGIN

FOR rec IN (

SELECT c.CustomerID, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE c.Age > 60

) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

END LOOP;

COMMIT;

END;

-- for checking the output.

SELECT \* FROM Loans;

**Output screenshot for the above scenario:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

**The above scenario can be executed using PL/SQL using the following code:**

BEGIN

FOR rec IN (

SELECT CustomerID, Balance FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = rec.CustomerID;

END LOOP;

COMMIT;

END;

-- for checking the output

SELECT \* FROM Customers;

**Output screenshot for the above scenario:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

**The above scenario can be executed using PL/SQL using the following code:**

BEGIN

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID ||

' for customer ' || rec.Name ||

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

END LOOP;

END;

**Output screenshot for the above scenario:**

A screenshot of a computer

AI-generated content may be incorrect.

**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.

**The above scenario can be executed using PL/SQL using the following code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END;

/

-- Run it

BEGIN

ProcessMonthlyInterest;

END;

/

-- Check results

SELECT \* FROM Accounts;

**Output for the above scenario:**

A screenshot of a computer

AI-generated content may be incorrect.

**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.

**The above scenario can be executed using PL/SQL using the following code:**

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;

/

BEGIN

UpdateEmployeeBonus('Sales', 10);

END;

/

SELECT \* FROM Employees;

**Output for the above scenario:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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.

**The logic for the above scenario is:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccount IN NUMBER,

p\_ToAccount IN NUMBER,

p\_Amount IN NUMBER

) AS

v\_Balance NUMBER;

BEGIN

-- Check source balance

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_FromAccount;

IF v\_Balance < p\_Amount THEN

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

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccount;

-- Add to target

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccount;

COMMIT;

END;

/

-- Example: Transfer 1000 from Alice (1) to Bob (2)

BEGIN

TransferFunds(1, 2, 1000);

END;

/

-- Check results

SELECT \* FROM Accounts;

**Output for the above scenario is given below:**

**A screenshot of a computer

AI-generated content may be incorrect.**