**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.**

CREATE TABLE Customer (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Age NUMBER,

LoanInterestRate NUMBER

);

INSERT INTO Customer VALUES (1, 'Anil', 45, 8.5);

INSERT INTO Customer VALUES (2, 'Suman', 61, 9.0);

INSERT INTO Customer VALUES (3, 'Rekha', 70, 10.0);

INSERT INTO Customer VALUES (4, 'Amit', 30, 7.5);

COMMIT;

BEGIN

FOR rec IN (SELECT \* FROM Customer) LOOP

IF rec.Age > 60 THEN

UPDATE Customer

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

SELECT \* FROM Customer;

**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.**

CREATE TABLE Customer (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Balance NUMBER,

IsVIP CHAR(1) -- 'Y' for TRUE, 'N' for FALSE

);

INSERT INTO Customer VALUES (1, 'Anil', 8000, 'N');

INSERT INTO Customer VALUES (2, 'Suman', 15000, 'N');

INSERT INTO Customer VALUES (3, 'Rekha', 12000, 'N');

INSERT INTO Customer VALUES (4, 'Amit', 5000, 'N');

COMMIT;

BEGIN

FOR rec IN (SELECT \* FROM Customer) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customer

SET IsVIP = 'Y'

WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

SELECT \* FROM Customer;

**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.

CREATE TABLE Loan (

LoanID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(50),

DueDate DATE

);

INSERT INTO Loan VALUES (1, 'Anil', SYSDATE + 10);

INSERT INTO Loan VALUES (2, 'Suman', SYSDATE + 40);

INSERT INTO Loan VALUES (3, 'Rekha', SYSDATE + 5);

INSERT INTO Loan VALUES (4, 'Amit', SYSDATE - 1);

COMMIT;

BEGIN

FOR rec IN (

SELECT \* FROM Loan

WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

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

' for customer ' || rec.CustomerName ||

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

END LOOP;

END;

/ **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.

CREATE TABLE SavingsAccount (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER

);

INSERT INTO SavingsAccount VALUES (101, 1000);

INSERT INTO SavingsAccount VALUES (102, 2500);

INSERT INTO SavingsAccount VALUES (103, 5000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE SavingsAccount

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

COMMIT;

END;

/

BEGIN

ProcessMonthlyInterest;

END;

/

SELECT \* FROM SavingsAccount;  
  
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.**

CREATE TABLE Employee (

EmpID NUMBER PRIMARY KEY,

DepartmentID NUMBER,

Salary NUMBER

);

INSERT INTO Employee VALUES (1, 101, 50000);

INSERT INTO Employee VALUES (2, 101, 60000);

INSERT INTO Employee VALUES (3, 102, 55000);

INSERT INTO Employee VALUES (4, 103, 52000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employee

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

WHERE DepartmentID = p\_dept\_id;

COMMIT;

END;

/

BEGIN

UpdateEmployeeBonus(101, 10);

END;

/

SELECT \* FROM Employee;

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.**

CREATE TABLE BankAccount (

AccountID NUMBER PRIMARY KEY,

Balance NUMBER

);

INSERT INTO BankAccount VALUES (201, 5000);

INSERT INTO BankAccount VALUES (202, 3000);

INSERT INTO BankAccount VALUES (203, 1000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_acc IN NUMBER,

p\_to\_acc IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM BankAccount WHERE AccountID = p\_from\_acc FOR UPDATE;

IF v\_balance >= p\_amount THEN

UPDATE BankAccount

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_acc;

UPDATE BankAccount

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_acc;

COMMIT;

ELSE

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

END IF;

END;

/

BEGIN

TransferFunds(201, 202, 1000);

END;

/

SELECT \* FROM BankAccount;  
  
A screenshot of a computer

AI-generated content may be incorrect.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_