**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.

DECLARE

CURSOR cur\_customers IS

SELECT customer\_id, interest\_rate

FROM Loans

WHERE customer\_id IN (

SELECT customer\_id FROM Customers WHERE age > 60

);

BEGIN

FOR loan\_rec IN cur\_customers LOOP

UPDATE Loans

SET interest\_rate = interest\_rate - 1

WHERE customer\_id = loan\_rec.customer\_id;

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.

BEGIN

FOR cust\_rec IN (

SELECT customer\_id FROM Customers WHERE balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE customer\_id = cust\_rec.customer\_id;

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 l.loan\_id, l.customer\_id, l.due\_date, c.name

FROM Loans l

JOIN Customers c ON l.customer\_id = c.customer\_id

WHERE l.due\_date BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan\_rec IN cur\_due\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.name ||

', your loan (ID: ' || loan\_rec.loan\_id ||

') is due on ' || TO\_CHAR(loan\_rec.due\_date, '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 IS

BEGIN

UPDATE Accounts

SET balance = balance + (balance \* 0.01)

WHERE account\_type = 'SAVINGS';

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 (

dept\_id IN NUMBER,

bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET salary = salary + (salary \* bonus\_percent / 100)

WHERE department\_id = dept\_id;

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 (

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

v\_balance NUMBER;

BEGIN

-- Get balance of source account

SELECT balance INTO v\_balance FROM Accounts WHERE account\_id = from\_account FOR UPDATE;

-- Check if sufficient balance

IF v\_balance < amount THEN

RAISE insufficient\_funds;

END IF;

-- Deduct from source

UPDATE Accounts

SET balance = balance - amount

WHERE account\_id = from\_account;

-- Add to destination

UPDATE Accounts

SET balance = balance + amount

WHERE account\_id = to\_account;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient funds.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;