**Week 2 PLSQL 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 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.

BEGIN

FOR cust\_rec IN (SELECT customer\_id, age, loan\_interest\_rate

FROM customers)

LOOP

IF cust\_rec.age > 60 THEN

UPDATE customers

SET loan\_interest\_rate = loan\_interest\_rate - 1

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

END IF;

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, balance

FROM customers)

LOOP

IF cust\_rec.balance > 10000 THEN

UPDATE customers

SET isvip = 'TRUE'

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

ELSE

UPDATE customers

SET isvip = 'FALSE'

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

END IF;

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 loans\_due IS

SELECT 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 loans\_due LOOP

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

' (ID: ' || loan\_rec.customer\_id ||

') has a loan 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 savings\_accounts

SET balance = balance \* 1.01; -- Adds 1% interest

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(

p\_department\_id IN employees.department\_id%TYPE,

p\_bonus\_pct IN NUMBER

) IS

BEGIN

UPDATE employees

SET salary = salary + (salary \* p\_bonus\_pct / 100)

WHERE department\_id = p\_department\_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(

p\_from\_account IN accounts.account\_id%TYPE,

p\_to\_account IN accounts.account\_id%TYPE,

p\_amount IN NUMBER

) IS

v\_source\_balance NUMBER;

BEGIN

-- Get source balance

SELECT balance INTO v\_source\_balance

FROM accounts

WHERE account\_id = p\_from\_account

FOR UPDATE;

-- Check balance

IF v\_source\_balance < p\_amount THEN

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

END IF;

-- Debit source account

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

-- Credit destination account

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

COMMIT;

END;