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.

**Ans:**

DECLARE

CURSOR cur\_customers IS

SELECT customer\_id, age

FROM customers;

v\_customer\_id customers.customer\_id%TYPE;

v\_age customers.age%TYPE;

BEGIN

FOR rec IN cur\_customers LOOP

v\_customer\_id := rec.customer\_id;

v\_age := rec.age;

IF v\_age > 60 THEN

UPDATE loans

SET interest\_rate = interest\_rate - 0.01

WHERE customer\_id = v\_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.

**Ans:**

DECLARE

CURSOR cur\_customers IS

SELECT customer\_id, balance

FROM customers;

BEGIN

FOR rec IN cur\_customers LOOP

IF rec.balance > 10000 THEN

UPDATE customers

SET IsVIP = 'TRUE'

WHERE customer\_id = 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.

**Ans:**

DECLARE

CURSOR cur\_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 <= SYSDATE + 30;

BEGIN

FOR rec IN cur\_loans LOOP

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

' for customer ' || rec.name ||

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

Ans:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

CURSOR cur\_savings IS

SELECT account\_id, balance

FROM savings\_accounts;

v\_acc\_id savings\_accounts.account\_id%TYPE;

v\_current\_bal savings\_accounts.balance%TYPE;

v\_updated\_bal savings\_accounts.balance%TYPE;

BEGIN

FOR acc\_rec IN cur\_savings LOOP

v\_acc\_id := acc\_rec.account\_id;

v\_current\_bal := acc\_rec.balance;

v\_updated\_bal := v\_current\_bal + (v\_current\_bal \* 0.01);

UPDATE savings\_accounts

SET balance = v\_updated\_bal

WHERE account\_id = v\_acc\_id;

END LOOP;

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.

Ans:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

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

p\_bonus\_pct IN NUMBER

) IS

v\_emp\_id employees.employee\_id%TYPE;

v\_old\_salary employees.salary%TYPE;

v\_new\_salary employees.salary%TYPE;

BEGIN

FOR emp\_rec IN (

SELECT employee\_id, salary

FROM employees

WHERE department\_id = p\_dept\_id

) LOOP

v\_emp\_id := emp\_rec.employee\_id;

v\_old\_salary := emp\_rec.salary;

v\_new\_salary := v\_old\_salary + (v\_old\_salary \* p\_bonus\_pct / 100);

UPDATE employees

SET salary = v\_new\_salary

WHERE employee\_id = v\_emp\_id;

END LOOP;

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.

Ans:

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_source\_acc\_id IN accounts.account\_id%TYPE,

p\_dest\_acc\_id IN accounts.account\_id%TYPE,

p\_amount IN NUMBER

) IS

v\_source\_balance accounts.balance%TYPE;

v\_dest\_balance accounts.balance%TYPE;

BEGIN

-- Get source balance

SELECT balance INTO v\_source\_balance

FROM accounts

WHERE account\_id = p\_source\_acc\_id;

-- Check for sufficient funds

IF v\_source\_balance < p\_amount THEN

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

END IF;

-- Deduct from source

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_source\_acc\_id;

-- Add to destination

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_dest\_acc\_id;

COMMIT;

END;

/