**WEEK – 2 HANDS ON**

**SKILL – PL/SQL programming**

**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.

**CODE:**

SET SERVEROUTPUT ON;

DECLARE

CURSOR cust\_cursor IS

SELECT customer\_id, age, loan\_interest\_rate

FROM customers

FOR UPDATE;

v\_new\_rate NUMBER;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

IF cust\_rec.age > 60 THEN

v\_new\_rate := cust\_rec.loan\_interest\_rate - (cust\_rec.loan\_interest\_rate \* 0.01)

UPDATE customers

SET loan\_interest\_rate = v\_new\_rate

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

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || cust\_rec.customer\_id || ' | New Interest Rate: ' || v\_new\_rate);

END IF;

END LOOP;

COMMIT;

END;

/

**OUTPUT:**

Customer ID: 101 | New Interest Rate: 6.93

Customer ID: 109 | New Interest Rate: 5.94

Customer ID: 120 | New Interest Rate: 7.92

**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.

**CODE:**

SET SERVEROUTPUT ON;

DECLARE

CURSOR cust\_cursor IS

SELECT customer\_id, balance

FROM customers

FOR UPDATE;

BEGIN

FOR cust\_rec IN cust\_cursor LOOP

IF cust\_rec.balance > 10000 THEN

UPDATE customers

SET IsVIP = 'TRUE'

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

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || cust\_rec.customer\_id || ' promoted to VIP.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || cust\_rec.customer\_id || ' not eligible for VIP.');

END IF;

END LOOP;

COMMIT;

END;

/

**OUTPUT:**

Customer ID: 101 promoted to VIP.

Customer ID: 102 not eligible for VIP.

Customer ID: 103 promoted to VIP.

**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.

**CODE:**

SET SERVEROUTPUT ON;

DECLARE

CURSOR due\_loan\_cursor IS

SELECT customer\_id, loan\_id, due\_date

FROM loans

WHERE due\_date BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan\_rec IN due\_loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ID ' || loan\_rec.customer\_id ||

' has Loan ID ' || loan\_rec.loan\_id ||

' due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-MON-YYYY') ||

'. Please make the payment on time.');

END LOOP;

END;

/

**OUTPUT:**

Reminder: Customer ID 101 has Loan ID L001 due on 28-JUN-2025. Please make the payment on time.

Reminder: Customer ID 108 has Loan ID L014 due on 10-JUL-2025. Please make the payment on time.

**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.

**CODE :**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (SELECT account\_id, balance

FROM accounts

WHERE account\_type = 'SAVINGS') LOOP

UPDATE accounts

SET balance = balance + (rec.balance \* 0.01)

WHERE account\_id = rec.account\_id;

DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.account\_id ||

' updated with interest. New balance: ' ||

TO\_CHAR(rec.balance + (rec.balance \* 0.01), '99999.99'));

END LOOP;

COMMIT;

END;

/

**CALL:**

BEGIN

ProcessMonthlyInterest;

END;

/

**OUTPUT:**

Account ID: 101 updated with interest. New balance: 10100.00

Account ID: 102 updated with interest. New balance: 15345.00

Account ID: 103 updated with interest. New balance: 20202.00

**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.

**CODE:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept\_id IN NUMBER,

bonus\_pct IN NUMBER

) IS

BEGIN

FOR rec IN (SELECT emp\_id, salary

FROM employees

WHERE department\_id = dept\_id) LOOP

UPDATE employees

SET salary = salary + (rec.salary \* bonus\_pct / 100)

WHERE emp\_id = rec.emp\_id;

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || rec.emp\_id ||

' received bonus. New salary: ' ||

TO\_CHAR(rec.salary + (rec.salary \* bonus\_pct / 100), '99999.99'));

END LOOP;

COMMIT;

END;

/

**CALL:**

BEGIN

UpdateEmployeeBonus(101, 10); -- 10% bonus for department 101

END;

/

**OUTPUT:**

Employee ID: 501 received bonus. New salary: 55000.00

Employee ID: 502 received bonus. New salary: 66000.00

Employee ID: 503 received bonus. New salary: 77000.00

**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.

**CODE:**

CREATE OR REPLACE PROCEDURE TransferFunds(

source\_account\_id IN NUMBER,

target\_account\_id IN NUMBER,

transfer\_amount IN NUMBER

) IS

source\_balance NUMBER;

BEGIN

-- Get source account balance

SELECT balance INTO source\_balance

FROM accounts

WHERE account\_id = source\_account\_id

FOR UPDATE;

IF source\_balance < transfer\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient funds in account ' || source\_account\_id);

ELSE

-- Deduct from source

UPDATE accounts

SET balance = balance - transfer\_amount

WHERE account\_id = source\_account\_id;

-- Add to target

UPDATE accounts

SET balance = balance + transfer\_amount

WHERE account\_id = target\_account\_id;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || transfer\_amount || ' from Account ' ||

source\_account\_id || ' to Account ' || target\_account\_id ||

' successful.');

END IF;

COMMIT;

END;

/

**CALL:**

BEGIN

TransferFunds(1001, 1002, 500);

END;

/

**OUTPUT:**

POSITIVE CASE:

Transfer of 500 from Account 1001 to Account 1002 successful.

NEGATIVE CASE:

Transfer failed: Insufficient funds in account 1001