**WEEK – 2**

**PL SQL PROGRAMMING**

**EXERCISE – 1: Control Structures.**

**SCENARIO-1:**

Apply 1% Interest Rate Discount for Customers Over 60.

**PROBLEM STATEMENT:**  
Loop through all customers, check if they are older than 60, and reduce their interest rate by 1%.

**SOURCE CODE:**

DECLARE

-- Mock customer table

TYPE customer\_rec IS RECORD (

id NUMBER,

age NUMBER,

interest\_rate NUMBER

);

TYPE customer\_table IS TABLE OF customer\_rec;

customers customer\_table := customer\_table(

customer\_rec(1, 65, 5.5),

customer\_rec(2, 45, 4.8),

customer\_rec(3, 70, 6.0)

);

BEGIN

FOR i IN 1 .. customers.COUNT LOOP

IF customers(i).age > 60 THEN

customers(i).interest\_rate := customers(i).interest\_rate - 1;

END IF;

END LOOP;

-- Output the results

FOR i IN 1 .. customers.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ' || customers(i).id ||

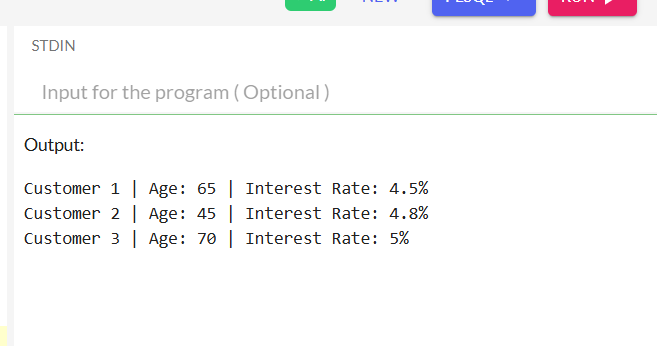
' | Age: ' || customers(i).age ||

' | Interest Rate: ' || customers(i).interest\_rate || '%');

END LOOP;

END;

**OUTPUT:**

****

**SCENARIO 2:** Promote Customers to VIP if Balance > $10,000.

**PROBLEM STATEMENT:**

Loop through customers and set IsVIP = TRUE if their balance exceeds $10,000.

**SOURCE CODE:**

DECLARE

TYPE customer\_rec IS RECORD (

id NUMBER,

balance NUMBER,

isvip VARCHAR2(5)

);

TYPE customer\_table IS TABLE OF customer\_rec;

customers customer\_table := customer\_table(

customer\_rec(1, 12000, 'FALSE'),

customer\_rec(2, 8000, 'FALSE'),

customer\_rec(3, 15000, 'FALSE')

);

BEGIN

FOR i IN 1 .. customers.COUNT LOOP

IF customers(i).balance > 10000 THEN

customers(i).isvip := 'TRUE';

END IF;

END LOOP;

-- Output the results

FOR i IN 1 .. customers.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Customer ' || customers(i).id ||

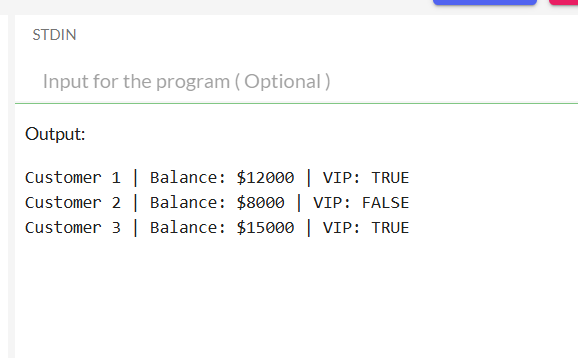
' | Balance: $' || customers(i).balance ||

' | VIP: ' || customers(i).isvip);

END LOOP;

END;

**OUTPUT:**

****

**SCENARIO 3:** Send Reminders for Loans Due in 30 Days.

**PROBLEM STATEMENT:**

Fetch loans due in the next 30 days and print a reminder for each.

**SOURCE CODE:**

DECLARE

TYPE loan\_rec IS RECORD (

loan\_id NUMBER,

customer VARCHAR2(50),

due\_date DATE

);

TYPE loan\_table IS TABLE OF loan\_rec;

loans loan\_table := loan\_table(

loan\_rec(101, 'John Doe', SYSDATE + 10),

loan\_rec(102, 'Jane Smith', SYSDATE + 40),

loan\_rec(103, 'Mike Lee', SYSDATE + 5)

);

BEGIN

FOR i IN 1 .. loans.COUNT LOOP

IF loans(i).due\_date <= SYSDATE + 30 THEN

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan #' || loans(i).loan\_id ||

' for ' || loans(i).customer ||

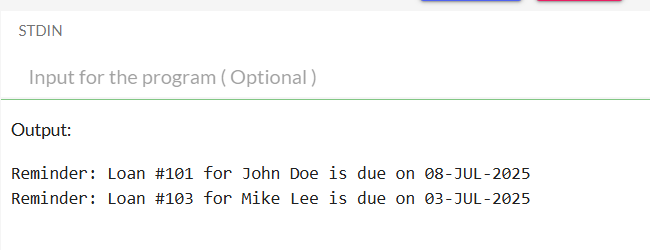
' is due on ' || TO\_CHAR(loans(i).due\_date, 'DD-MON-YYYY'));

END IF;

END LOOP;

END;

**OUTPUT:**

****

**EXERCISE-3:** Stored Procedures.

**SCENARIO 1:** Process Monthly Interest for Savings Accounts.

**PROBLEM STATEMENT:**

Create a procedure Process Monthly Interest that increases the balance of all savings accounts by 1%.

**SOURCE CODE:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE savings\_accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE savings\_accounts (

account\_id NUMBER,

balance NUMBER

);

-- Insert test data

INSERT INTO savings\_accounts VALUES (101, 1000);

INSERT INTO savings\_accounts VALUES (102, 2500);

INSERT INTO savings\_accounts VALUES (103, 5000);

COMMIT;

-- Create Procedure

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE savings\_accounts

SET balance = balance \* 1.01;

END;

/

-- Run the procedure

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Before update:');

FOR r IN (SELECT \* FROM savings\_accounts) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ' || r.account\_id || ': $' || r.balance);

END LOOP;

-- Call procedure

ProcessMonthlyInterest;

DBMS\_OUTPUT.PUT\_LINE('After update:');

FOR r IN (SELECT \* FROM savings\_accounts) LOOP

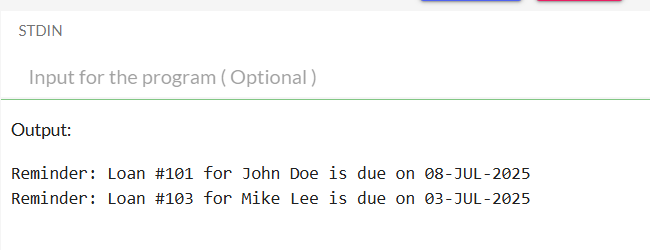
DBMS\_OUTPUT.PUT\_LINE('Account ' || r.account\_id || ': $' || ROUND(r.balance, 2));

END LOOP;

END;

/

**OUTPUT:**

****

**SCENARIO 2:** Update Employee Bonus.

**PROBLEM STATEMENT:**

Create a procedure Update Employee Bonus that increases the salary of employees in a specific department by a given bonus percentage.

**SOURCE CODE:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE employees (

emp\_id NUMBER,

name VARCHAR2(50),

department VARCHAR2(20),

salary NUMBER

);

-- Insert test data

INSERT INTO employees VALUES (1, 'Alice', 'HR', 3000);

INSERT INTO employees VALUES (2, 'Bob', 'Sales', 4000);

INSERT INTO employees VALUES (3, 'Charlie', 'Sales', 4500);

INSERT INTO employees VALUES (4, 'Diana', 'IT', 5000);

COMMIT;

-- Create Procedure

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept VARCHAR2,

p\_bonus\_pct NUMBER

) IS

BEGIN

UPDATE employees

SET salary = salary \* (1 + p\_bonus\_pct / 100)

WHERE department = p\_dept;

END;

/

-- Test Procedure

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Before Bonus (Sales):');

FOR r IN (SELECT \* FROM employees WHERE department = 'Sales') LOOP

DBMS\_OUTPUT.PUT\_LINE(r.name || ': $' || r.salary);

END LOOP;

-- Apply 10% bonus to Sales

UpdateEmployeeBonus('Sales', 10);

DBMS\_OUTPUT.PUT\_LINE('After Bonus (Sales):');

FOR r IN (SELECT \* FROM employees WHERE department = 'Sales') LOOP

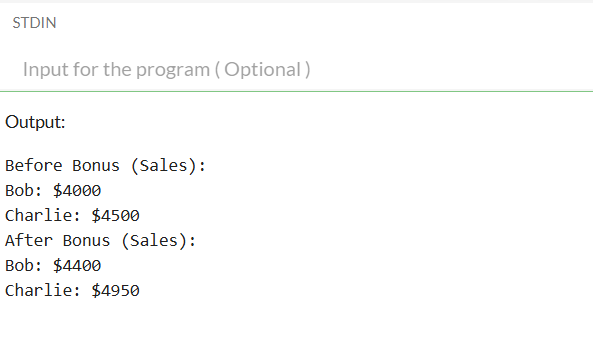
DBMS\_OUTPUT.PUT\_LINE(r.name || ': $' || ROUND(r.salary, 2));

END LOOP;

END;

/

**OUTPUT:**

****

**SCENARIO 3:** Transfer Funds Between Accounts.

**PROBLEM STATEMENT:**

Create a procedure Transfer Funds that moves money from one account to another if the source has enough balance.

**SOURCE CODE:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE accounts';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE accounts (

acc\_id NUMBER PRIMARY KEY,

balance NUMBER

);

-- Insert test data

INSERT INTO accounts VALUES (1, 5000);

INSERT INTO accounts VALUES (2, 2000);

COMMIT;

-- Create Procedure

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from NUMBER,

p\_to NUMBER,

p\_amount NUMBER

) IS

v\_from\_balance NUMBER;

BEGIN

-- Check source balance

SELECT balance INTO v\_from\_balance FROM accounts WHERE acc\_id = p\_from;

IF v\_from\_balance >= p\_amount THEN

UPDATE accounts SET balance = balance - p\_amount WHERE acc\_id = p\_from;

UPDATE accounts SET balance = balance + p\_amount WHERE acc\_id = p\_to;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: $' || p\_amount);

ELSE

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

END IF;

END;

/

-- Test Procedure

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Before transfer:');

FOR r IN (SELECT \* FROM accounts) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ' || r.acc\_id || ': $' || r.balance);

END LOOP;

-- Transfer $1500 from Account 1 to Account 2

TransferFunds(1, 2, 1500);

DBMS\_OUTPUT.PUT\_LINE('After transfer:');

FOR r IN (SELECT \* FROM accounts) LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ' || r.acc\_id || ': $' || r.balance);

END LOOP;

END;

/

**OUTPUT:**

