**WEEK 2**

**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.

**PROGRAM**

DECLARE

TYPE CustomerRec IS RECORD (

CustomerID NUMBER,

Age NUMBER,

LoanInterestRate NUMBER

);

TYPE CustomerTable IS TABLE OF CustomerRec;

customers CustomerTable := CustomerTable(

CustomerRec(1, 65, 8.5),

CustomerRec(2, 45, 7.0),

CustomerRec(3, 70, 9.0)

);

BEGIN

FOR i IN 1 .. customers.COUNT LOOP

IF customers(i).Age > 60 THEN

customers(i).LoanInterestRate := customers(i).LoanInterestRate \* 0.99;

END IF;

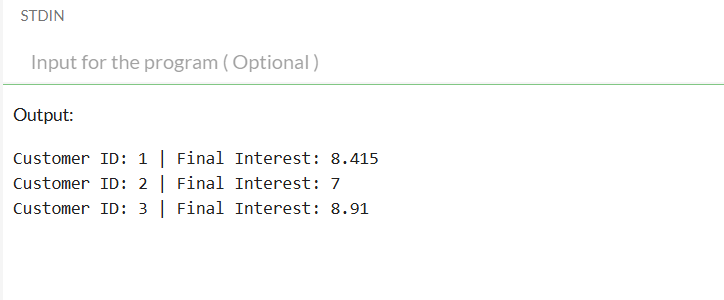
DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || customers(i).CustomerID ||

' | Final Interest: ' || customers(i).LoanInterestRate);

END LOOP;

END;

**OUTPUT**



**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**PROGRAM**

DECLARE

TYPE CustomerRec IS RECORD (

CustomerID NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5)

);

TYPE CustomerTable IS TABLE OF CustomerRec;

customers CustomerTable := CustomerTable(

CustomerRec(1, 9500, 'FALSE'),

CustomerRec(2, 15000, 'FALSE'),

CustomerRec(3, 20000, 'FALSE')

);

BEGIN

FOR i IN 1 .. customers.COUNT LOOP

IF customers(i).Balance > 10000 THEN

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

END IF;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || customers(i).CustomerID ||

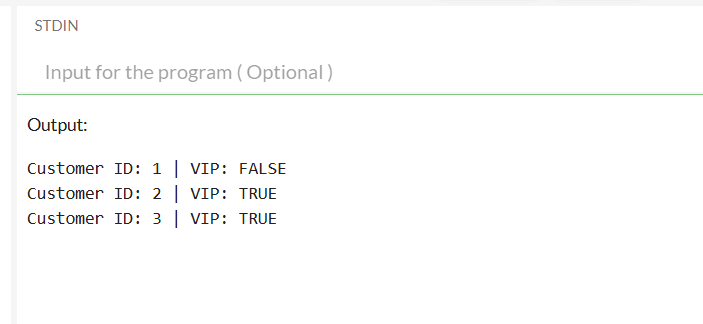
' | VIP: ' || customers(i).IsVIP);

END LOOP;

END;

/

**OUTPUT**

****

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**PROGRAM**

DECLARE

TYPE LoanRec IS RECORD (

LoanID NUMBER,

CustomerID NUMBER,

DueDate DATE

);

TYPE LoanTable IS TABLE OF LoanRec;

loans LoanTable := LoanTable(

LoanRec(101, 1, SYSDATE + 10),

LoanRec(102, 2, SYSDATE + 40),

LoanRec(103, 3, SYSDATE + 5)

);

BEGIN

FOR i IN 1 .. loans.COUNT LOOP

IF loans(i).DueDate BETWEEN SYSDATE AND SYSDATE + 30 THEN

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loans(i).LoanID ||

' for Customer ' || loans(i).CustomerID ||

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

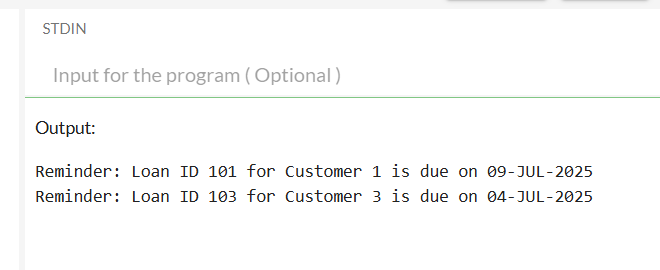
END IF;

END LOOP;

END;

/

**OUTPUT**



**EXERCISE 3: STORED PROCEDURES**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

**PROGRAM**

DECLARE

TYPE SavingsAccount IS RECORD (

AccountID NUMBER,

Balance NUMBER

);

TYPE AccountTable IS TABLE OF SavingsAccount;

accounts AccountTable := AccountTable(

SavingsAccount(1, 1000),

SavingsAccount(2, 2000),

SavingsAccount(3, 3000)

);

PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR i IN 1 .. accounts.COUNT LOOP

accounts(i).Balance := accounts(i).Balance \* 1.01; -- add 1% interest

END LOOP;

END;

BEGIN

ProcessMonthlyInterest;

-- Output updated balances

FOR i IN 1 .. accounts.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ' || accounts(i).AccountID ||

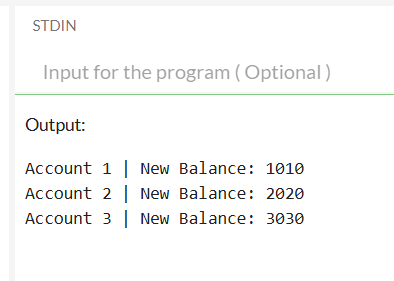
' | New Balance: ' || accounts(i).Balance);

END LOOP;

END;

/

**OUTPUT**

****

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

**PROGRAM**

DECLARE

TYPE Employee IS RECORD (

EmpID NUMBER,

DeptID NUMBER,

Salary NUMBER

);

TYPE EmployeeTable IS TABLE OF Employee;

employees EmployeeTable := EmployeeTable(

Employee(1, 10, 5000),

Employee(2, 20, 6000),

Employee(3, 10, 7000)

);

PROCEDURE UpdateEmployeeBonus(dept IN NUMBER, bonus\_percent IN NUMBER) IS

BEGIN

FOR i IN 1 .. employees.COUNT LOOP

IF employees(i).DeptID = dept THEN

employees(i).Salary := employees(i).Salary \* (1 + bonus\_percent / 100);

END IF;

END LOOP;

END;

BEGIN

UpdateEmployeeBonus(10, 10); -- Apply 10% bonus to Dept 10

FOR i IN 1 .. employees.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Emp ' || employees(i).EmpID ||

' | Dept ' || employees(i).DeptID ||

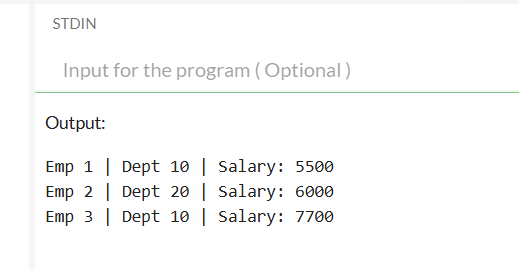
' | Salary: ' || employees(i).Salary);

END LOOP;

END;

/

**OUTPUT**

****

**Scenario 3:** Customers should be able to transfer funds between their accounts.

**PROGRAM**

DECLARE

TYPE Account IS RECORD (

AccountID NUMBER,

Balance NUMBER

);

TYPE AccountTable IS TABLE OF Account;

accounts AccountTable := AccountTable(

Account(1, 1000),

Account(2, 2000)

);

PROCEDURE TransferFunds(from\_id IN NUMBER, to\_id IN NUMBER, amt IN NUMBER) IS

from\_index NUMBER := 0;

to\_index NUMBER := 0;

BEGIN

-- Find index of both accounts

FOR i IN 1 .. accounts.COUNT LOOP

IF accounts(i).AccountID = from\_id THEN

from\_index := i;

ELSIF accounts(i).AccountID = to\_id THEN

to\_index := i;

END IF;

END LOOP;

IF from\_index = 0 OR to\_index = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Invalid account(s)');

RETURN;

END IF;

IF accounts(from\_index).Balance < amt THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds in account ' || from\_id);

ELSE

accounts(from\_index).Balance := accounts(from\_index).Balance - amt;

accounts(to\_index).Balance := accounts(to\_index).Balance + amt;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amt || ' from Account ' || from\_id || ' to ' || to\_id);

END IF;

END;

BEGIN

TransferFunds(1, 2, 500);

FOR i IN 1 .. accounts.COUNT LOOP

DBMS\_OUTPUT.PUT\_LINE('Account ' || accounts(i).AccountID ||

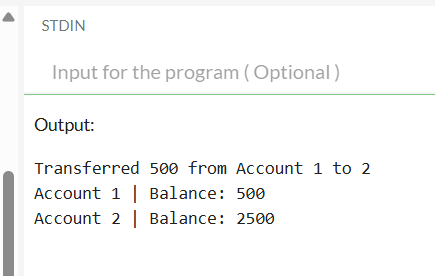
' | Balance: ' || accounts(i).Balance);

END LOOP;

END;

/

**OUTPUT**

****