PL / SQL EXERCISE

WEEK 2

SUPERSET ID : 6425093

Exercise 1: Control Structures

//CREATING TABLES AND INSERTING VALUES

Query:

```
CREATE TABLE CUSTOMER (
   CustomerID NUMBER PRIMARY KEY,
   Name VARCHAR2(100),
   Age NUMBER,
   Balance NUMBER,
   IsVIP CHAR(1)
);
```

//Inserting Value into CUSTOMER table.

Query:

```
INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (1, 'Alice Johnson', 65, 15000, 'N');

INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (2, 'Bob Smith', 45, 8000, 'N');

INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (3, 'Carol Lee', 70, 11000, 'N');

INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (4, 'David Kim', 55, 5000, 'N');

INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (5, 'Emma Wilson', 62, 9500, 'N');

INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (6, 'Frank Miller', 68, 12500, 'N');

INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (7, 'Grace Chen', 59, 7000, 'N');

INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (8, 'Henry Patel', 61, 10500, 'N');

INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (9, 'Isabel Gomez', 72, 20000, 'N');

INSERT INTO CUSTOMER (CustomerID, Name, Age, Balance, IsVIP) VALUES (10, 'Jack Lee', 50, 4000, 'N');

COMMIT;


SELECT * FROM CUSTOMER;
```

Output:

CUSTOMERID	NAME	AGE	BALANCE	I
1	Alice Johnson	65	15000	N
2	Bob Smith	45	8000	N
3	Carol Lee	70	11000	N
4	David Kim	55	5000	N
5	Emma Wilson	62	9500	N
6	Frank Miller	68	12500	N
7	Grace Chen	59	7000	N
8	Henry Patel	61	10500	N
9	Isabel Gomez	72	20000	N
10	Jack Lee	50	4000	N

10 rows selected.

//Create LOAN table with LoanID, CustomerID, InterestRate, DueDate.

Query:

```
CREATE TABLE LOAN (
   LoanID NUMBER PRIMARY KEY,
   CustomerID NUMBER REFERENCES CUSTOMER(CustomerID),
   InterestRate NUMBER,
   DueDate DATE
);
```

//Inserting Value into CUSTOMER table.

Query:

```
INSERT INTO LOAN (LoanID, CustomerID, InterestRate, DueDate) VALUES (101, 1, 8.5, SYSDATE + 15);

INSERT INTO LOAN (LoanID, CustomerID, InterestRate, DueDate) VALUES (102, 2, 9.0, SYSDATE + 40);

INSERT INTO LOAN (LoanID, CustomerID, InterestRate, DueDate) VALUES (103, 3, 7.5, SYSDATE + 10);

INSERT INTO LOAN (LoanID, CustomerID, InterestRate, DueDate) VALUES (104, 4, 8.0, SYSDATE + 25);

INSERT INTO LOAN (LoanID, CustomerID, InterestRate, DueDate) VALUES (105, 5, 8.2, SYSDATE + 5);

INSERT INTO LOAN (LoanID, CustomerID, InterestRate, DueDate) VALUES (106, 6, 9.1, SYSDATE + 12);

INSERT INTO LOAN (LoanID, CustomerID, InterestRate, DueDate) VALUES (107, 7, 8.7, SYSDATE + 45);
```

INSERT INTO LOAN (LoanID, CustomerID, InterestRate, DueDate) VALUES (108, 8, 7.9, SYSDATE + 20);

INSERT INTO LOAN (LoanID, CustomerID, InterestRate, DueDate) VALUES (109, 9, 8.3, SYSDATE + 8);

Output:

LOANID	CUSTOMERID	INTERESTRATE	DUEDATE
101	1	8.5	14-07-25
102	2	9	08-08-25
103	3	7.5	09-07-25
104	4	8	24-07-25
105	5	8.2	04-07-25
106	6	9.1	11-07-25
107	7	8.7	13-08-25
108	8	7.9	19-07-25
109	9	8.3	07-07-25
110	10	8.8	03-08-25

10 rows selected.

SCENARIO 1

The bank wants to apply a discount to loan interest rates for customers above 60 years old.

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.

Query:

```
BEGIN
  FOR cust_rec IN (
    SELECT CustomerID
    FROM CUSTOMER
    WHERE Age > 60
  ) LOOP
    UPDATE LOAN
    SET InterestRate = InterestRate - 1
    WHERE CustomerID = cust_rec.CustomerID;
    DBMS_OUTPUT.PUT_LINE('Applied discount to CustomerID: ' || cust_rec.CustomerID);
  END LOOP;

  COMMIT;
END;
/
```

Output:

```
Applied discount to CustomerID: 1
Applied discount to CustomerID: 3
Applied discount to CustomerID: 5
Applied discount to CustomerID: 6
Applied discount to CustomerID: 8
Applied discount to CustomerID: 9
```

SCENARIO 2

A customer can be promoted to VIP status based on their balance.

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.

Query:

```
BEGIN
  FOR cust_rec IN (
    SELECT CustomerID, Balance
    FROM CUSTOMER
  ) LOOP
    IF cust_rec.Balance > 10000 THEN
      UPDATE CUSTOMER
      SET IsVIP = 'Y'
      WHERE CustomerID = cust_rec.CustomerID;
      DBMS_OUTPUT.PUT_LINE('Promoted CustomerID ' || cust_rec.CustomerID || ' to VIP.');
    ELSE
      UPDATE CUSTOMER
      SET IsVIP = 'N'
      WHERE CustomerID = cust_rec.CustomerID;
      DBMS_OUTPUT.PUT_LINE('Demoted CustomerID ' || cust_rec.CustomerID || ' from VIP.');
    END IF;
  END LOOP;

  COMMIT;
END;
/
```

Output:

```
Promoted CustomerID 1 to VIP.
Demoted CustomerID 2 from VIP.
Promoted CustomerID 3 to VIP.
Demoted CustomerID 4 from VIP.
Demoted CustomerID 5 from VIP.
Promoted CustomerID 6 to VIP.
Demoted CustomerID 7 from VIP.
Promoted CustomerID 8 to VIP.
Promoted CustomerID 9 to VIP.
Demoted CustomerID 10 from VIP.
```

SCENARIO 3

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

Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

Query:

```
BEGIN
  FOR loan_rec IN (
    SELECT l.LoanID, l.CustomerID, l.DueDate, c.Name
    FROM LOAN l
    JOIN CUSTOMER c ON l.CustomerID = c.CustomerID
    WHERE l.DueDate <= SYSDATE + 30
  ) LOOP
    DBMS_OUTPUT.PUT_LINE(
      'Reminder: Customer ' || loan_rec.Name ||
      ' (ID: ' || loan_rec.CustomerID ||
      ') has Loan ID ' || loan_rec.LoanID ||
      ' due on ' || TO_CHAR(loan_rec.DueDate, 'DD-MON-YYYY')
    );
  END LOOP;
END;
```

Output:

```
Reminder: Customer Alice Johnson (ID: 1) has Loan ID 101 due on 14-JUL-2025
Reminder: Customer Carol Lee (ID: 3) has Loan ID 103 due on 09-JUL-2025
Reminder: Customer David Kim (ID: 4) has Loan ID 104 due on 24-JUL-2025
Reminder: Customer Emma Wilson (ID: 5) has Loan ID 105 due on 04-JUL-2025
Reminder: Customer Frank Miller (ID: 6) has Loan ID 106 due on 11-JUL-2025
Reminder: Customer Henry Patel (ID: 8) has Loan ID 108 due on 19-JUL-2025
Reminder: Customer Isabel Gomez (ID: 9) has Loan ID 109 due on 07-JUL-2025
```

Exercise 3: Stored Procedures

//CREATING TABLES AND INSERTING VALUES

Query:

```
CREATE TABLE ACCOUNTS (
   AccountID NUMBER PRIMARY KEY,
   CustomerID NUMBER,
   AccountType VARCHAR2(20),
   Balance NUMBER
);
```

//Inserting values into ACCOUNTS Table.

Query:

```
INSERT INTO ACCOUNTS VALUES (1, 101, 'SAVINGS', 5000);
INSERT INTO ACCOUNTS VALUES (2, 102, 'SAVINGS', 12000);
INSERT INTO ACCOUNTS VALUES (3, 103, 'SAVINGS', 8000);
INSERT INTO ACCOUNTS VALUES (4, 104, 'CHECKING', 4000);
INSERT INTO ACCOUNTS VALUES (5, 105, 'CHECKING', 7000);
INSERT INTO ACCOUNTS VALUES (6, 106, 'SAVINGS', 15000);
INSERT INTO ACCOUNTS VALUES (7, 107, 'SAVINGS', 9500);
INSERT INTO ACCOUNTS VALUES (8, 108, 'CHECKING', 6000);
INSERT INTO ACCOUNTS VALUES (9, 109, 'SAVINGS', 11000);
INSERT INTO ACCOUNTS VALUES (10, 110, 'CHECKING', 3000);

COMMIT;
SELECT * FROM ACCOUNTS;
```

Output:

ACCOUNTID	CUSTOMERID	ACCOUNTTYPE	BALANCE
1	101	SAVINGS	5000
2	102	SAVINGS	12000
3	103	SAVINGS	8000
4	104	CHECKING	4000
5	105	CHECKING	7000
6	106	SAVINGS	15000
7	107	SAVINGS	9500
8	108	CHECKING	6000
9	109	SAVINGS	11000
10	110	CHECKING	3000

10 rows selected.

Query:

```
CREATE TABLE EMPLOYEE (
   EmpID NUMBER PRIMARY KEY,
   Name VARCHAR2(100),
   DepartmentID NUMBER,
   Salary NUMBER
);
```

//Inserting Values into Employee table

Query:

```
INSERT INTO EMPLOYEE VALUES (1, 'Alice', 10, 50000);
INSERT INTO EMPLOYEE VALUES (2, 'Bob', 20, 55000);
INSERT INTO EMPLOYEE VALUES (3, 'Carol', 10, 60000);
INSERT INTO EMPLOYEE VALUES (4, 'David', 20, 58000);
INSERT INTO EMPLOYEE VALUES (5, 'Eve', 10, 62000);
INSERT INTO EMPLOYEE VALUES (6, 'Frank', 30, 45000);
```

```
INSERT INTO EMPLOYEE VALUES (7, 'Grace', 30, 48000);
INSERT INTO EMPLOYEE VALUES (8, 'Henry', 20, 52000);
INSERT INTO EMPLOYEE VALUES (9, 'Ivy', 10, 61000);
INSERT INTO EMPLOYEE VALUES (10, 'Jack', 30, 47000);

COMMIT;
SELECT * FROM EMPLOYEE;
```

Output:

```
     EMPID NAME
---------- ------------------------
         1 Alice
         2 Bob
         3 Carol
         4 David
         5 Eve
         6 Frank
         7 Grace
         8 Henry
         9 Ivy
        10 Jack

10 rows selected.
```

SCENARIO 1

The bank needs to process monthly interest for all savings accounts.

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.

Query:

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  FOR acc_rec IN (
    SELECT AccountID, Balance
    FROM ACCOUNTS
    WHERE AccountType = 'SAVINGS'
  ) LOOP
    UPDATE ACCOUNTS
    SET Balance = Balance + (Balance * 0.01)
    WHERE AccountID = acc_rec.AccountID;

    DBMS_OUTPUT.PUT_LINE('Applied 1% interest to AccountID: ' || acc_rec.AccountID);
  END LOOP;

  COMMIT;
END;
/

//Calling Procedure:
BEGIN
  ProcessMonthlyInterest;
END;
/
```

Output:

```
Applied 1% interest to AccountID: 1
Applied 1% interest to AccountID: 2
Applied 1% interest to AccountID: 3
Applied 1% interest to AccountID: 6
Applied 1% interest to AccountID: 7
Applied 1% interest to AccountID: 9
```

SCENARIO 2

The bank wants to implement a bonus scheme for employees based on their performance.

Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

Query:

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
  p_DepartmentID IN NUMBER,
  p_BonusPercent IN NUMBER
) IS
BEGIN
  FOR emp_rec IN (
    SELECT EmpID, Salary
    FROM EMPLOYEE
    WHERE DepartmentID = p_DepartmentID
  ) LOOP
    UPDATE EMPLOYEE
    SET Salary = Salary + (Salary * p_BonusPercent / 100)
    WHERE EmpID = emp_rec.EmpID;

    DBMS_OUTPUT.PUT_LINE('Updated EmpID ' || emp_rec.EmpID || ' with bonus ' ||
p_BonusPercent || '%');
  END LOOP;

  COMMIT;
END;
/

//Calling Procedure:
BEGIN
  UpdateEmployeeBonus(10, 10);
END;
/
```

Output:

```
Updated EmpID 1 with bonus 10%
Updated EmpID 3 with bonus 10%
Updated EmpID 5 with bonus 10%
Updated EmpID 9 with bonus 10%
```

SCENARIO 3

Customers should be able to transfer funds between their accounts.

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.

Query:

```
CREATE OR REPLACE PROCEDURE TransferFunds (
  p_FromAccountID IN NUMBER,
  p_ToAccountID IN NUMBER,
  p_Amount IN NUMBER
) IS
  v_FromBalance NUMBER;
BEGIN
  SELECT Balance INTO v_FromBalance FROM ACCOUNTS WHERE AccountID =
p_FromAccountID;

  IF v_FromBalance < p_Amount THEN
    DBMS_OUTPUT.PUT_LINE('Insufficient funds in AccountID: ' || p_FromAccountID);
  ELSE
    UPDATE ACCOUNTS
    SET Balance = Balance - p_Amount
    WHERE AccountID = p_FromAccountID;

    UPDATE ACCOUNTS
    SET Balance = Balance + p_Amount
    WHERE AccountID = p_ToAccountID;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('Transferred ' || p_Amount || ' from AccountID ' ||
p_FromAccountID || ' to AccountID ' || p_ToAccountID);
  END IF;
END;
/
```

```
//Calling Procedure:
BEGIN
  TransferFunds(2, 1, 1000);
  TransferFunds(4, 5, 500);
  TransferFunds(2, 5, 200);
  TransferFunds(2, 3, 600);
  TransferFunds(1, 7, 400);
  TransferFunds(4, 6, 400);
END;
/
```

Output:

```
Transferred 1000 from AccountID 2 to AccountID 1
Transferred 500 from AccountID 4 to AccountID 5
Transferred 200 from AccountID 2 to AccountID 5
Transferred 600 from AccountID 2 to AccountID 3
Transferred 400 from AccountID 1 to AccountID 7
Transferred 400 from AccountID 4 to AccountID 6

Transferred 1000 from AccountID 2 to AccountID 5
```