Week-2(PL/SQL):
EXERCISE-I

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.
- **Solution:**For this we should first create a database that contain required columns and data.

SQL QUERY:

1)Query-1

```
CREATE TABLE Customers (
   CustomerID NUMBER PRIMARY KEY,
   Name VARCHAR2(100),
   Age NUMBER,
   Balance NUMBER,
   IsVIP CHAR(1) DEFAULT 'N'
);
```

2)Query-2

```
INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'N');
INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 'N');
INSERT INTO Customers VALUES (3, 'Charlie', 70, 15000, 'N');
INSERT INTO Customers VALUES (4, 'Diana', 58, 9500, 'N');
```

OUTPUT:

1)output-1:

Table CUSTOMERS created.

2)output-2:

1 row inserted.
1 row inserted.
1 row inserted.
1 row inserted.

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.
- **Solution:**For this we should first create a database that contain required columns and data.

SQL QUERY:

1)Query-1

```
CREATE TABLE Loans (
   LoanID NUMBER PRIMARY KEY,
   CustomerID NUMBER,
   InterestRate NUMBER,
   DueDate DATE,
```

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
2)Query-2
INSERT INTO Loans VALUES (101, 1, 8.5, SYSDATE + 10);
INSERT INTO Loans VALUES (102, 2, 9.0, SYSDATE + 40);
INSERT INTO Loans VALUES (103, 3, 7.5, SYSDATE + 25);
INSERT INTO Loans VALUES (104, 4, 8.0, SYSDATE + 5);

OUTPUT:

1)output-1:
Table LOANS created.
2)output-2:
1 row inserted.
1 row inserted.
1 row inserted.
1 row inserted.

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.
- **Solution:**It is the combinations of two tables data we required.

PL/SQL codes for the above scenario:

Scenario 1: Discount for Age > 60:

PL/SQL CODE:

```
BEGIN
   FOR rec IN (SELECT LoanID, InterestRate
               FROM Loans l
               JOIN Customers c ON l.CustomerID = c.CustomerID
               WHERE c.Age > 60) LOOP
      UPDATE Loans
      SET InterestRate = rec.InterestRate - 1
      WHERE LoanID = rec.LoanID;
   END LOOP;
   COMMIT;
END;
```

/

OUTPUT:
PL/SQL procedure successfully completed.

Query:

SELECT * FROM Loans;

Updated Table:

LOANID	CUSTOMERID	INTERESTRATE	DUEDATE
101	1	7.5	<today+10>
102	2	9.0	<today+40>
103	3	6.5	<today+25>
104	4	8.0	<today+5>

Scenario 2: Promote VIPs:
PL/SQL CODE:

```
BEGIN
   FOR cust IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP
      UPDATE Customers
      SET IsVIP = 'Y'
      WHERE CustomerID = cust.CustomerID;
   END LOOP;
   COMMIT;
END;
/
```

Output:

PL/SQL procedure successfully completed.

Query:

SELECT * FROM Customers;

Updated Table:

CUSTOMERID	NAME	AGE	BALANCE	ISVIP
1	Alice	65	12000	Y
2	Bob	45	8000	N
3	Charlie	70	15000	Y
4	Diana	58	9500	N

Scenario 3: Print Loan Due Reminders:

PL/SQL CODE:

```
BEGIN
    FOR rec IN (
        SELECT c.Name, l.DueDate
        FROM Customers c
        JOIN Loans l ON c.CustomerID = l.CustomerID
        WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30
    ) LOOP
        DBMS_OUTPUT.PUT_LINE('Reminder: ' || rec.Name || ' has a loan due on ' ||
TO_CHAR(rec.DueDate, 'DD-Mon-YYYY'));
    END LOOP;
END;
/
```

Output:

Reminder: Alice has a loan due on 05-Jul-2025

Reminder: Charlie has a loan due on 20-Jul-2025

Reminder: Diana has a loan due on 30-Jun-2025

EXERCISE-III:

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.
- **Solution:**For this we should first create a database that contain required columns and data.

SQL QUERY:

1)Query-1

```
CREATE TABLE SavingsAccounts (
   AccountID NUMBER PRIMARY KEY,
   CustomerName VARCHAR2(100),
   Balance NUMBER
);
```

2)Query-2

```
INSERT INTO SavingsAccounts VALUES (1, 'Alice', 10000);
INSERT INTO SavingsAccounts VALUES (2, 'Bob', 20000);
INSERT INTO SavingsAccounts VALUES (3, 'Charlie', 15000);
```

OUTPUT:

1)output-1:

Table SavingsAccounts created.

2)output-2:

1 row inserted.

1 row inserted.

1 row inserted.

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.
- **Solution:**For this we should first create a database that contain required columns and data.

SQL QUERY:

1)Query-1

```
CREATE TABLE Employees (
   EmployeeID NUMBER PRIMARY KEY,
   Name VARCHAR2(100),
   Department VARCHAR2(50),
   Salary NUMBER
);
```

2)Query-2

```
INSERT INTO Employees VALUES (101, 'John', 'HR', 50000);
INSERT INTO Employees VALUES (102, 'Jane', 'IT', 60000);
```

INSERT INTO Employees VALUES (103, 'Mike', 'IT', 55000);

OUTPUT:

1)output-1:

Table Employees created.

2)output-2:

1 row inserted.

1 row inserted.

1 row inserted.

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.
- **Solution:**For this we should first create a database that contain required columns and data.

SQL QUERY:

1)Query-1

```
CREATE TABLE BankAccounts (
   AccountID NUMBER PRIMARY KEY,
   CustomerName VARCHAR2(100),
   Balance NUMBER
);
```

2)Query-2

```
INSERT INTO BankAccounts VALUES (201, 'Alice', 25000);
INSERT INTO BankAccounts VALUES (202, 'Bob', 15000);
```

OUTPUT:

1)output-1:

Table Employees created.

2)output-2:

1 row inserted.

1 row inserted.

PL/SQL codes for the above scenario:

Scenario 1: `ProcessMonthlyInterest` (1% Interest)

PL/SQL CODE:(Procedure)

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS
BEGIN
   FOR acc IN (SELECT AccountID, Balance FROM SavingsAccounts) LOOP
      UPDATE SavingsAccounts
      SET Balance = Balance + (Balance * 0.01)
      WHERE AccountID = acc.AccountID;
   END LOOP;
   COMMIT;
END;
```

Run the procedure:

EXEC ProcessMonthlyInterest;

QUERY:

SELECT * FROM SavingsAccounts;

OUTPUT:

AccountID	CustomerName	Balance
1	Alice	10100.00
2	Bob	20200.00
3	Charlie	15150.00

Scenario 2: `UpdateEmployeeBonus:`

PL/SQL CODE:(Procedure)

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
    deptName IN VARCHAR2,
    bonusPct IN NUMBER  -- e.g., pass 10 for 10%
) AS
BEGIN
    UPDATE Employees
    SET Salary = Salary + (Salary * bonusPct / 100)
    WHERE Department = deptName;

    COMMIT;
END;
/
```

Run the procedure:

EXEC UpdateEmployeeBonus('IT', 10);

QUERY:

SELECT * FROM Employees;

OUTPUT:

EmployeeID	Name	Department	Salary
101	John	HR	50000
102	Jane	IT	66000
103	Mike	IT	60500

Scenario 3: TransferFunds

PL/SQL CODE:(Procedure)

```
CREATE OR REPLACE PROCEDURE TransferFunds(
   fromAcc IN NUMBER,
   toAcc IN NUMBER,
   amount IN NUMBER
) AS
   fromBalance NUMBER;
BEGIN
   -- Get balance of source account
   SELECT Balance INTO fromBalance FROM BankAccounts WHERE AccountID = fromAcc
FOR UPDATE;

   IF fromBalance < amount THEN
      RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source account.');
   ELSE
      -- Deduct from source
      UPDATE BankAccounts
      SET Balance = Balance - amount
      WHERE AccountID = fromAcc;

      -- Add to destination
      UPDATE BankAccounts
      SET Balance = Balance + amount
      WHERE AccountID = toAcc;

      COMMIT;
   END IF;
END;
/
```

Run the procedure:

```
EXEC TransferFunds(201, 202, 5000);
```

QUERY:

```
SELECT * FROM BankAccounts;
```

OUTPUT:

AccountID	CustomerName	Balance
201	Alice	20000
202	Bob	20000