**Schemas to be created :**

SQL> create table Customers(CustomerID int primary key,Name varchar(100),DOB Date,Balance Decimal(10,2),LastModified date);

Table created.

SQL> create table Accounts(AccountID INT primary key,CustomerID INT,AccountType VARCHAR(100),Balance Decimal(10,2),LastModified date,foreign key(CustomerID) references Customers(CustomerID));

Table created.

SQL> create table Transactions(TransactionID int primary key,AccountID int,TransactionDate Date,Amount Decimal(10,2),TransactionType varchar(10),foreign key(AccountID) references Accounts(AccountID));

Table created.

SQL> create table Loans(LoanID int primary key,CustomerID int,LoanAmount decimal(10,2),InterestRate decimal(10,2),StartDate date,EndDate date,foreign key(CustomerID) references Customers(CustomerID));

Table created.

SQL> create table Employees (EmployeeID INT PRIMARY KEY,Name VARCHAR(100),Position VARCHAR(50),Salary DECIMAL(10, 2),Department VARCHAR(50),HireDate DATE);

Table created.

**Example Scripts for Sample Data Insertion :**

SQL> INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) values (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

1 row created.

SQL> INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

1 row created.

SQL> INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (1, 1, 'Savings', 1000, SYSDATE);

1 row created.

SQL> INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified) VALUES (2,2, 'Checking', 1500, SYSDATE);

1 row created.

SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (1, 1, SYSDATE, 200, 'Deposit');

1 row created.

SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType) VALUES (2, 2, SYSDATE, 300, 'Withdrawal'

);

1 row created.

SQL> INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

1 row created.

SQL> INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

1 row created.

SQL> INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

1 row created.

**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.

**Source Code:**

SQL> BEGIN

FOR cust IN (

SELECT CustomerID

FROM Customers

WHERE FLOOR(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) > 60

)

LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Applied discount to Customer ID: ' || cust.CustomerID);

END LOOP;

END;

/**output :**

PL/SQL procedure successfully completed.

'Applied discount to Customer ID : 1

**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.

**Source Code:**

SQL> ALTER TABLE Customers ADD IsVIP CHAR(1) DEFAULT 'N';

SET SERVEROUTPUT ON;

BEGIN

FOR cust IN (

SELECT CustomerID, Balance

FROM Customers

WHERE Balance > 10000

)

LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Promoted Customer ID: ' || cust.CustomerID || ' to VIP status.');

END LOOP;

END;

/

**Output:**

Customer ID 1 promoted to VIP status.

PL/SQL procedure successfully completed.

**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.

**Source Code:**

SQL> set serveroutput on;

SQL> BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, l.CustomerID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

)

LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Dear ' || loan\_rec.Name ||

', your loan (Loan ID: ' || loan\_rec.LoanID ||

') is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-Mon-YYYY') ||

'. Please make necessary arrangements.'

);

END LOOP;

END;

/

PL/SQL procedure successfully completed.

SQL> select \* from Loans;

LOANID CUSTOMERID LOANAMOUNT INTERESTRATE STARTDATE ENDDATE

---------- ---------- ---------- ------------ --------- ---------

1 1 5000 5 24-JUN-25 24-JUN-30

**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.

**Source code:**

SQL> CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest of 1% applied to all savings accounts.');

END;

/

PL/SQL procedure successfully completed.

SQL> select \* from accounts;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACCOUNTID** | **CUSTOMERID** | **ACCOUNTTYPE** | **BALANCE** | **LASTMODIF** |
| 1 | 1 | Savings | 1010.00 | 24-JUN-25 |
| 2 | 2 | Checking | 1500.00 | 24-JUN-25 |

**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.

**Source Code:**

SQL> CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_Department VARCHAR,

p\_BonusPercent DECIMAL

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_BonusPercent / 100)

WHERE Department = p\_Department;

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' employee(s) in department "' || p\_Department ||

'" received a bonus of ' || p\_BonusPercent || '%.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('No employees found in department "' || p\_Department || '".');

END IF;

COMMIT;

END;

/Procedure created.

SQL> BEGIN

2 UpdateEmployeeBonus('It', 10);

3 END;

4 /

Output:

PL/SQL procedure successfully completed.

SQL> select \* from employees;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **EMPLOYEEID** | **NAME** | **POSITION** | **SALARY** | **DEPARTMENT** | **HIREDATE** |
| 1 | Alice Johnson | Manager | 70000.00 | HR | 15-JUN-15 |
| 2 | Bob Brown | Developer | 66000.00 | IT | 20-MAR-17 |

**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

**Source code:**

SQL> CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID INT,

p\_ToAccountID INT,

p\_Amount DECIMAL

) AS

v\_FromBalance DECIMAL;

BEGIN

-- Step 1: Check current balance of source account

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

IF v\_FromBalance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Transfer failed: Insufficient funds in source account.');

END IF;

-- Step 2: Deduct from source and add to destination

UPDATE Accounts

SET Balance = Balance - p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_FromAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_ToAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of $' || p\_Amount ||

' from Account ' || p\_FromAccountID ||

' to Account ' || p\_ToAccountID || ' completed.');

END;

/

Procedure created.

SQL> BEGIN

2 TransferFunds(1, 2, 200);

3 END;

4 /

PL/SQL procedure successfully completed.

SQL> select \* from accounts;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ACCOUNTID** | **CUSTOMERID** | **ACCOUNTTYPE** | **BALANCE** | **LASTMODIF** |
| 1 | 1 | Savings | 810.00 | 24-JUN-25 |
| 2 | 2 | Checking | 1700.00 | 24-JUN-25 |