**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> set serveroutput on;

SQL> BEGIN

FOR cust IN

( SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate FROM Customers c JOIN Loans l ON c.CustomerID = l.CustomerID )

LOOP

IF MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12 > 60 THEN

UPDATE Loans SET InterestRate = InterestRate - 1

WHERE LoanID = cust.LoanID;

DBMS\_OUTPUT.PUT\_LINE('Interest rate reduced for Customer ID: ' || cust.CustomerID);  
END IF;  
END LOOP;  
COMMIT;  
END; /

**output :**

PL/SQL procedure successfully completed.

Interest rate reduced for 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);

Table altered.

SQL> update Customers set balance=100000 where CustomerID=1;

1 row updated.

SQL> set serveroutput on;

SQL> BEGIN

FOR cust IN

(SELECT CustomerID, Balance FROM Customers)

LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

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

END IF;

END LOOP;

COMMIT;

END;

/

**Output:**

Customer ID 1 promoted to VIP.

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> DECLARE

2

3 CURSOR due\_loans\_cur IS

4 SELECT l.LOANID, l.CUSTOMERID, c.NAME, l.ENDDATE

5 FROM LOANS l

6 JOIN CUSTOMERS c ON l.CUSTOMERID = c.CUSTOMERID

7 WHERE l.ENDDATE BETWEEN SYSDATE-5 AND SYSDATE + 30;

8

9

10 v\_loanid LOANS.LOANID%TYPE;

11 v\_customerid LOANS.CUSTOMERID%TYPE;

12 v\_customername CUSTOMERS.NAME%TYPE;

13 v\_enddate LOANS.ENDDATE%TYPE;

14 BEGIN

15 OPEN due\_loans\_cur;

16 LOOP

17 FETCH due\_loans\_cur INTO v\_loanid, v\_customerid, v\_customername, v\_enddate;

18 EXIT WHEN due\_loans\_cur%NOTFOUND;

19

20

21 DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || v\_customername ||

22 ', your loan (Loan ID: ' || v\_loanid ||

23 ') is due on ' || TO\_CHAR(v\_enddate, 'DD-MON-YYYY') ||

24 '. Please ensure timely payment.');

25 END LOOP;

26 CLOSE due\_loans\_cur;

27 END;

28 /

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 IS

2 BEGIN

3

4 UPDATE ACCOUNTS

5 SET BALANCE = BALANCE + (BALANCE \* 0.01)

6 WHERE UPPER(ACCOUNTTYPE) = 'SAVINGS';

7

8 COMMIT;

9 END;

10 /

Procedure created.

SQL> BEGIN

2 ProcessMonthlyInterest;

3 END;

4 /

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 (

2 p\_department\_name IN VARCHAR2,

3 p\_bonus\_percent IN NUMBER

4 ) IS

5 BEGIN

6

7 UPDATE EMPLOYEES

8 SET SALARY = SALARY + (SALARY \* p\_bonus\_percent / 100)

9 WHERE UPPER(DEPARTMENT) = UPPER(p\_department\_name);

10

11 COMMIT;

12 END;

13 /

Procedure created.

SQL> BEGIN

2 UpdateEmployeeBonus('It', 10);

3 END;

4 /

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 (

2 p\_from\_account\_id IN NUMBER,

3 p\_to\_account\_id IN NUMBER,

4 p\_amount IN NUMBER

5 ) IS

6 v\_from\_balance ACCOUNTS.BALANCE%TYPE;

7 BEGIN

8 SELECT BALANCE INTO v\_from\_balance

9 FROM ACCOUNTS

10 WHERE ACCOUNTID = p\_from\_account\_id

11 FOR UPDATE;

12

13 IF v\_from\_balance < p\_amount THEN

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

15 END IF;

16

17 UPDATE ACCOUNTS

18 SET BALANCE = BALANCE - p\_amount

19 WHERE ACCOUNTID = p\_from\_account\_id;

20

21 UPDATE ACCOUNTS

22 SET BALANCE = BALANCE + p\_amount

23 WHERE ACCOUNTID = p\_to\_account\_id;

24

25 COMMIT;

26 END;

27 /

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 |