**Schema to be Created**

*CREATE TABLE Customers (*

*CustomerID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*DOB DATE,*

*Balance NUMBER,*

*LastModified DATE*

*);*

*CREATE TABLE Accounts (*

*AccountID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*AccountType VARCHAR2(20),*

*Balance NUMBER,*

*LastModified DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

*CREATE TABLE Transactions (*

*TransactionID NUMBER PRIMARY KEY,*

*AccountID NUMBER,*

*TransactionDate DATE,*

*Amount NUMBER,*

*TransactionType VARCHAR2(10),*

*FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)*

*);*

*CREATE TABLE Loans (*

*LoanID NUMBER PRIMARY KEY,*

*CustomerID NUMBER,*

*LoanAmount NUMBER,*

*InterestRate NUMBER,*

*StartDate DATE,*

*EndDate DATE,*

*FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)*

*);*

*CREATE TABLE Employees (*

*EmployeeID NUMBER PRIMARY KEY,*

*Name VARCHAR2(100),*

*Position VARCHAR2(50),*

*Salary NUMBER,*

*Department VARCHAR2(50),*

*HireDate DATE*

*);*

**Example Scripts for Sample Data Insertion**

*INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)*

*VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);*

*INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)*

*VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);*

*INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)*

*VALUES (1, 1, 'Savings', 1000, SYSDATE);*

*INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)*

*VALUES (2, 2, 'Checking', 1500, SYSDATE);*

*INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)*

*VALUES (1, 1, SYSDATE, 200, 'Deposit');*

*INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)*

*VALUES (2, 2, SYSDATE, 300, 'Withdrawal');*

*INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)*

*VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));*

*INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)*

*VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));*

*INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)*

*VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));*

**Note:**

Live Sql Script Link:

<https://livesql.oracle.com/ords/livesql/s/dmobrejgciv8mhc9i2kt0znih>

**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.

**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.

**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**

* **Scenario 1:**
* Code:

BEGIN

FOR cust\_rec IN (SELECT CustomerID, DOB FROM Customers) LOOP

DECLARE

v\_age NUMBER;

BEGIN

-- Calculate age

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, cust\_rec.DOB) / 12);

-- Check if customer age is greater than 60

IF v\_age > 60 THEN

-- Apply 1% discount on interest rate

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust\_rec.CustomerID;

-- Display success message

DBMS\_OUTPUT.PUT\_LINE('1% discount applied to Customer ID: ' || cust\_rec.CustomerID);

ELSE

-- Display message for customers not eligible

DBMS\_OUTPUT.PUT\_LINE('No discount applied. Customer ID: ' || cust\_rec.CustomerID || ' is aged ' || v\_age || ' years.');

END IF;

END;

END LOOP;

COMMIT;

END;

* Output:



* **Scenario 2:**
* Code:

ALTER TABLE Customers ADD (IsVIP VARCHAR2(5))

* Output:



* Code:

BEGIN

FOR cust\_rec IN (SELECT CustomerID, Name, Balance FROM Customers) LOOP

BEGIN

-- Check if customer balance is greater than 10000

IF cust\_rec.Balance > 10000 THEN

-- Promote the customer to VIP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust\_rec.CustomerID;

-- Display VIP Promotion message

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

ELSE

-- Display message for Non VIP customers

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = cust\_rec.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || cust\_rec.CustomerID || 'with Customer Name ' || cust\_rec.Name || ' not promoted to VIP.');

END IF;

END;

END LOOP;

COMMIT;

END;

* Output:



* **Scenario 3:**
* Code:

BEGIN

DECLARE

v\_customerName VARCHAR2(100);

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, l.CustomerID, l.EndDate

FROM Loans l

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

-- Extract the details of the specified customers

SELECT Name INTO v\_customerName

FROM Customers

WHERE CustomerID = loan\_rec.CustomerID;

-- Display the Reminder message

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.LoanID ||

' for customer ' || v\_customerName ||

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

END LOOP;

END;

COMMIT;

END;

* Output:



**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.

**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.

**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**

* **Scenario 1:**
* Code:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

END;

* Output:



* Code:

BEGIN

ProcessMonthlyInterest;

END;

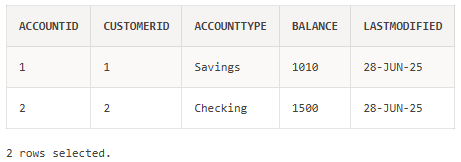
* Output:



* Code:

SELECT \* FROM Accounts

* Output:



* **Scenario 2:**
* Code:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) AS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percentage / 100)

WHERE Department = p\_department;

IF SQL%ROWCOUNT > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Bonus updated for department: ' || p\_department);

ELSE

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

END IF;

COMMIT;

END;

* Output:



* Code:

BEGIN

UpdateEmployeeBonus('HR', 10);

END;

* Output:



* Code:

BEGIN

UpdateEmployeeBonus('Sales', 8.5);

END;

* Output:



* Code:

SELECT \* FROM Employees

* Output:



* **Scenario 3:**
* Code:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_fromAccountID IN NUMBER,

p\_toAccountID IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

-- Check source balance

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_fromAccountID;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Insufficient balance in source account.');

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_fromAccountID;

-- Credit destination

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_toAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Funds transferred from Account ' || p\_fromAccountID || ' to ' || p\_toAccountID);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error during fund transfer: ' || SQLERRM);

END;

* Output:



* Code:

BEGIN

TransferFunds(1, 2, 500);

END;

* Output:



* Code:

SELECT \* FROM Accounts

* Output:

