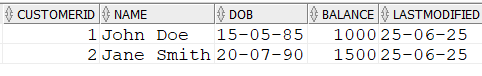
**TABLES BEFORE …**

**CUSTOMERS**

****

**LOANS**

****

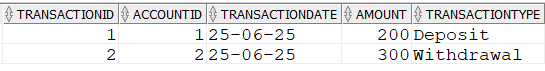
**ACCOUNTS**

****

**EMPLOYEES**

****

**TRANSACTIONS**

****

**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.

BEGIN

-- Scenario 1: Discount for customers > 60 years

FOR rec IN (

SELECT c.CustomerID, l.LoanID, l.InterestRate, c.DOB

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

)

LOOP

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

UPDATE Loans

SET InterestRate = rec.InterestRate - 1

WHERE LoanID = rec.LoanID;

END IF;

END LOOP;

END;

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

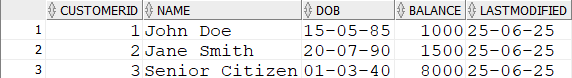
VALUES (3, 'Senior Citizen', TO\_DATE('1940-03-01', 'YYYY-MM-DD'), 8000, SYSDATE);

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

VALUES (2, 3, 4000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 36)); -- 3-year loan

SELECT \* FROM LOANS;





**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.

ALTER TABLE Customers

ADD IsVIP CHAR(1);

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

VALUES (4, 'Rich VIP', TO\_DATE('1980-01-01', 'YYYY-MM-DD'), 15000, SYSDATE);

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers)

LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'Y' WHERE CustomerID = rec.CustomerID;

ELSE

UPDATE Customers SET IsVIP = 'N' WHERE CustomerID = rec.CustomerID;

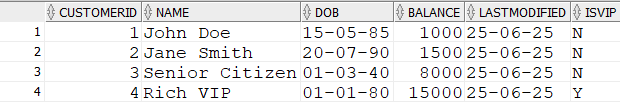
END IF;

END LOOP;

COMMIT;

END;

SELECT \* FROM CUSTOMERS;



**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.

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (

SELECT LoanID, CustomerID, EndDate

FROM Loans

WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30

)

LOOP

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

' for Customer ' || rec.CustomerID ||

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

END LOOP;

END;

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

VALUES (3, 2, 3000, 4, SYSDATE, SYSDATE + 10); -- Due in 10 days



**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.

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

VALUES (3, 4, 'Savings', 12000, SYSDATE);

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR rec IN (SELECT AccountID, Balance FROM Accounts WHERE LOWER(AccountType) = 'savings')

LOOP

UPDATE Accounts

SET Balance = rec.Balance \* 1.01,

LastModified = SYSDATE

WHERE AccountID = rec.AccountID;

END LOOP;

COMMIT;

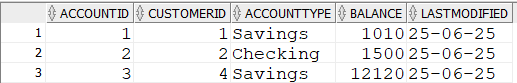
END;

BEGIN

processmonthlyinterest;

END;

SELECT \* FROM ACCOUNTS;



**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.

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

VALUES (3, 'Charlie King', 'Tester', 50000, 'IT', TO\_DATE('2018-01-01', 'YYYY-MM-DD'));

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_Department IN VARCHAR2,

p\_BonusPct IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

COMMIT;

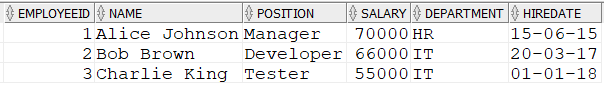
END;

BEGIN

updateemployeebonus('IT',10);

END;

SELECT \* FROM EMPLOYEES;



**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.

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAccount IN NUMBER,

p\_ToAccount IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts WHERE AccountID = p\_FromAccount;

IF v\_Balance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');

END IF;

UPDATE Accounts

SET Balance = Balance - p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_FromAccount;

UPDATE Accounts

SET Balance = Balance + p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_ToAccount;

COMMIT;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer Error: ' || SQLERRM);

ROLLBACK;

END;

BEGIN

transferfunds(2,3,1111);

END;

SELECT \* FROM ACCOUNTS;

