**Exercise 1: Control Structures**

**Scenario 1:**

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

)

LOOP

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

UPDATE Loans

SET InterestRate = InterestRate - (rec.InterestRate \* 0.01)

WHERE LoanID = rec.LoanID;

DBMS\_OUTPUT.PUT\_LINE(' Interest rate discounted for ' || rec.Name ||

' (Loan ID: ' || rec.LoanID || ')');

END IF;

END LOOP;

COMMIT;

END;

/

**OUTPUT:**

**Screenshot 2025-06-26 144105**

**Scenario 2:**

SET SERVEROUTPUT ON;

BEGIN

  FOR cust\_rec IN (

    SELECT CustomerID, Name, Balance

    FROM Customers

  )

  LOOP

    IF cust\_rec.Balance > 10000 THEN

      UPDATE Customers

      SET IsVIP = 'TRUE'

      WHERE CustomerID = cust\_rec.CustomerID;

      DBMS\_OUTPUT.PUT\_LINE(' ' || cust\_rec.Name || ' promoted to VIP (Balance: ₹' || cust\_rec.Balance || ')');

    END IF;

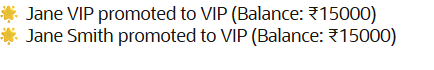
  END LOOP;

  COMMIT;

END;

/

**OUTPUT:**



**Scenario 3:**

SET SERVEROUTPUT ON;

BEGIN

FOR rec IN (

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

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: Loan ID ' || rec.LoanID ||

' for ' || rec.Name ||

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

END LOOP;

END;

/

**OUTPUT:**



**Exercise 2: Error Handling**

**Scenario 1:**

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

  p\_from\_account\_id IN NUMBER,

  p\_to\_account\_id   IN NUMBER,

  p\_amount          IN NUMBER

)

IS

  v\_from\_balance NUMBER;

BEGIN

  SELECT Balance INTO v\_from\_balance

  FROM Accounts

  WHERE AccountID = p\_from\_account\_id;

  IF v\_from\_balance < p\_amount THEN

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

  END IF;

  UPDATE Accounts

  SET Balance = Balance - p\_amount

  WHERE AccountID = p\_from\_account\_id;

  UPDATE Accounts

  SET Balance = Balance + p\_amount

  WHERE AccountID = p\_to\_account\_id;

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('✅ ₹' || p\_amount || ' transferred successfully.');

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

    DBMS\_OUTPUT.PUT\_LINE('❌ One or both account IDs not found.');

    ROLLBACK;

  WHEN OTHERS THEN

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

    ROLLBACK;

END;

/

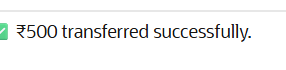
BEGIN

  SafeTransferFunds(2, 1, 500);

END;

/

**OUTPUT:**



**Scenario 2:**

SET SERVEROUTPUT ON;

-- Procedure only

CREATE OR REPLACE PROCEDURE UpdateSalary (

    p\_EmpID IN NUMBER,

    p\_Percent IN NUMBER

) AS

    v\_Count NUMBER;

BEGIN

    SELECT COUNT(\*) INTO v\_Count FROM Employees WHERE EmployeeID = p\_EmpID;

    IF v\_Count = 0 THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'Employee ID ' || p\_EmpID || ' does not exist.');

    END IF;

    UPDATE Employees

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

    WHERE EmployeeID = p\_EmpID;

    DBMS\_OUTPUT.PUT\_LINE('Salary updated successfully for Employee ID ' || p\_EmpID);

EXCEPTION

    WHEN OTHERS THEN

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

END;

/

-- Testing (only this part if table/data already exists)

BEGIN

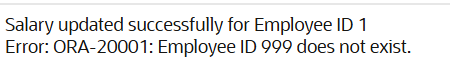
    UpdateSalary(1, 10); -- Valid

    UpdateSalary(999, 10); -- Invalid

END;

/

**OUTPUT:**



**Scenario 3:**

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE AddNewCustomer (

    p\_CustomerID IN NUMBER,

    p\_Name IN VARCHAR2,

    p\_DOB IN DATE,

    p\_Balance IN NUMBER

)

IS

BEGIN

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

    VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

    DBMS\_OUTPUT.PUT\_LINE('Customer added successfully: ID = ' || p\_CustomerID);

EXCEPTION

    WHEN DUP\_VAL\_ON\_INDEX THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: Customer with ID ' || p\_CustomerID || ' already exists. Insertion aborted.');

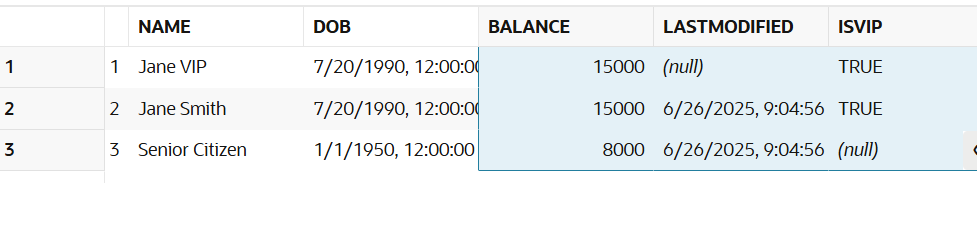
END;

/

EXEC AddNewCustomer(3, 'Charlie Green', TO\_DATE('1992-09-10', 'YYYY-MM-DD'), 2000);

SELECT \* FROM Customers ORDER BY CustomerID;

**OUTPUT:**

****

**Exercise 3: Stored Procedures**

**Scenario 1:**

-- Step 1: Enable output

SET SERVEROUTPUT ON;

-- Step 2: Create or replace the procedure

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

    -- Loop through all Savings accounts

    FOR rec IN (

        SELECT AccountID, Balance

        FROM Accounts

        WHERE AccountType = 'Savings'

    )

    LOOP

        -- Calculate new balance by adding 1% interest

        UPDATE Accounts

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

            LastModified = SYSDATE

        WHERE AccountID = rec.AccountID;

        -- Output message

        DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || rec.AccountID);

    END LOOP;

END;

/

-- Step 3: Execute the procedure

EXEC ProcessMonthlyInterest;

**OUTPUT:**



**Scenario 2:**

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_Department IN VARCHAR2,

    p\_BonusPercent IN NUMBER

)

IS

    v\_Count NUMBER := 0;

BEGIN

    -- Update salaries

    UPDATE Employees

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

    WHERE Department = p\_Department;

    -- Get number of affected rows

    v\_Count := SQL%ROWCOUNT;

    -- Output message

    IF v\_Count > 0 THEN

        DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_BonusPercent || '% applied to ' || v\_Count || ' employee(s) in department: ' || p\_Department);

    ELSE

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

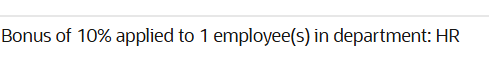
    END IF;

END;

/

EXEC UpdateEmployeeBonus('HR', 10);

**OUTPUT:**



**Scenario 3:**

-- DROP tables if already exist (for safe re-runs)

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE Transactions';

  EXECUTE IMMEDIATE 'DROP TABLE Accounts';

  EXECUTE IMMEDIATE 'DROP TABLE Loans';

  EXECUTE IMMEDIATE 'DROP TABLE Employees';

  EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- Drop sequence if exists

BEGIN

  EXECUTE IMMEDIATE 'DROP SEQUENCE Transactions\_seq';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- 1. Create Tables

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(20), -- Fixed: increased from 10 to 20

    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

);

-- 2. Create Sequence for TransactionID

CREATE SEQUENCE Transactions\_seq START WITH 1 INCREMENT BY 1;

-- 3. Insert Sample Data

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, 1, 'Checking', 1500, SYSDATE);

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

VALUES (Transactions\_seq.NEXTVAL, 1, SYSDATE, 200, 'Deposit');

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

VALUES (Transactions\_seq.NEXTVAL, 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'));

-- 4. Create Procedure: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_SourceAccountID IN NUMBER,

    p\_DestinationAccountID IN NUMBER,

    p\_Amount IN NUMBER

)

IS

    v\_SourceBalance NUMBER;

    v\_SourceCustomerID NUMBER;

    v\_DestCustomerID NUMBER;

BEGIN

    -- Prevent self-transfer

    IF p\_SourceAccountID = p\_DestinationAccountID THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: Source and destination accounts must be different.');

        RETURN;

    END IF;

    -- Get source account info

    SELECT Balance, CustomerID INTO v\_SourceBalance, v\_SourceCustomerID

    FROM Accounts

    WHERE AccountID = p\_SourceAccountID

    FOR UPDATE;

    -- Get destination account info

    SELECT CustomerID INTO v\_DestCustomerID

    FROM Accounts

    WHERE AccountID = p\_DestinationAccountID

    FOR UPDATE;

    -- Same customer check

    IF v\_SourceCustomerID != v\_DestCustomerID THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: Transfers allowed only between same customer''s accounts.');

        RETURN;

    END IF;

    -- Sufficient funds check

    IF v\_SourceBalance < p\_Amount THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in source account ' || p\_SourceAccountID);

        RETURN;

    END IF;

    -- Perform transfer

    UPDATE Accounts

    SET Balance = Balance - p\_Amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_SourceAccountID;

    UPDATE Accounts

    SET Balance = Balance + p\_Amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_DestinationAccountID;

    -- Record transactions

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

    VALUES (Transactions\_seq.NEXTVAL, p\_SourceAccountID, SYSDATE, p\_Amount, 'Transfer-Out');

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

    VALUES (Transactions\_seq.NEXTVAL, p\_DestinationAccountID, SYSDATE, p\_Amount, 'Transfer-In');

    DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || p\_Amount || ' from Account ' || p\_SourceAccountID || ' to Account ' || p\_DestinationAccountID || ' completed successfully.');

    COMMIT;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error: One or both account IDs not found.');

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

END;

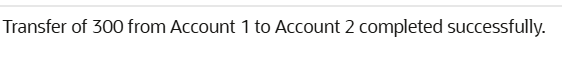
/

-- 5. Execute the procedure

SET SERVEROUTPUT ON;

EXEC TransferFunds(1, 2, 300);

**OUTPUT:**



**Exercise 4: Functions**

**Scenario 1:**

-- Enable output

SET SERVEROUTPUT ON;

-- Query to display name and calculated age

SELECT Name, DOB, CalculateAge(DOB) AS Age

FROM Customers;

CREATE OR REPLACE FUNCTION CalculateAge (

    p\_DOB DATE

) RETURN NUMBER

IS

    v\_Age NUMBER;

BEGIN

    -- Calculate age in years using MONTHS\_BETWEEN and TRUNC

    v\_Age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

    RETURN v\_Age;

EXCEPTION

    WHEN OTHERS THEN

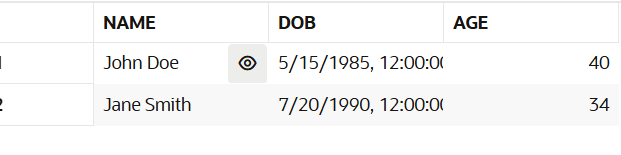
        DBMS\_OUTPUT.PUT\_LINE('Error calculating age: ' || SQLERRM);

        RETURN NULL;

END;

/

**OUTPUT:**



**Scenario 2:**

-- (Optional) Drop the function if it exists

BEGIN

  EXECUTE IMMEDIATE 'DROP FUNCTION CalculateMonthlyInstallment';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- 1. Create the FUNCTION to calculate EMI

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

    p\_LoanAmount      IN NUMBER,

    p\_AnnualInterest  IN NUMBER,

    p\_Years           IN NUMBER

) RETURN NUMBER

IS

    v\_MonthlyRate NUMBER;

    v\_NumPayments NUMBER;

    v\_Installment NUMBER;

BEGIN

    -- Convert annual rate to monthly rate

    v\_MonthlyRate := p\_AnnualInterest / 12 / 100;

    -- Total number of payments

    v\_NumPayments := p\_Years \* 12;

    -- EMI formula

    IF v\_MonthlyRate > 0 THEN

        v\_Installment := p\_LoanAmount \* v\_MonthlyRate \* POWER(1 + v\_MonthlyRate, v\_NumPayments)

                         / (POWER(1 + v\_MonthlyRate, v\_NumPayments) - 1);

    ELSE

        v\_Installment := p\_LoanAmount / v\_NumPayments;

    END IF;

    RETURN ROUND(v\_Installment, 2);

EXCEPTION

    WHEN OTHERS THEN

        RETURN NULL;

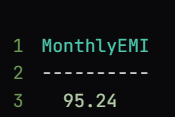
END;

/

-- This will show the EMI in query result

SELECT CalculateMonthlyInstallment(5000, 5, 5) AS MonthlyEMI FROM dual;

**OUTPUT:**



**Scenario 3:**

SET SERVEROUTPUT ON;

-- Function returning VARCHAR2 instead of BOOLEAN

CREATE OR REPLACE FUNCTION HasSufficientBalance (

    p\_AccountID IN NUMBER,

    p\_Amount IN NUMBER

) RETURN VARCHAR2 IS

    v\_Balance NUMBER;

BEGIN

    SELECT Balance INTO v\_Balance

    FROM Accounts

    WHERE AccountID = p\_AccountID;

    IF v\_Balance >= p\_Amount THEN

        RETURN 'TRUE';

    ELSE

        RETURN 'FALSE';

    END IF;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN 'ACCOUNT\_NOT\_FOUND';

    WHEN OTHERS THEN

        RETURN 'ERROR';

END;

/

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('Account 1, Amount 500: ' || HasSufficientBalance(1, 500));

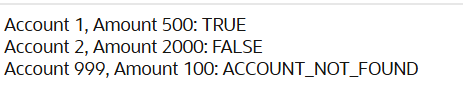
    DBMS\_OUTPUT.PUT\_LINE('Account 2, Amount 2000: ' || HasSufficientBalance(2, 2000));

    DBMS\_OUTPUT.PUT\_LINE('Account 999, Amount 100: ' || HasSufficientBalance(999, 100));

END;

/

**OUTPUT:**



**Exercise 5: Triggers**

**Scenario 1:**

-- Enable output

SET SERVEROUTPUT ON;

-- Drop the table if it exists (optional cleanup)

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN

NULL;

END;

/

-- 1. Create Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

-- 2. Insert sample customers

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);

COMMIT;

-- 3. Create trigger to auto-update LastModified on UPDATE

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

-- 4. Update a customer to activate the trigger

UPDATE Customers

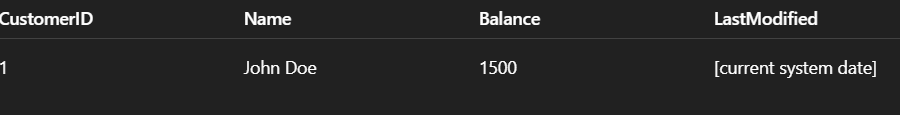
SET Balance = Balance + 500

WHERE CustomerID = 1;

-- 5. Show the updated row

SELECT \* FROM Customers WHERE CustomerID = 1;

**OUTPUT:**



**Scenario 2:**

-- Optional: Clean up if tables exist

BEGIN

  EXECUTE IMMEDIATE 'DROP TRIGGER LogTransaction';

  EXECUTE IMMEDIATE 'DROP TABLE AuditLog';

  EXECUTE IMMEDIATE 'DROP TABLE Transactions';

  EXECUTE IMMEDIATE 'DROP TABLE Accounts';

  EXECUTE IMMEDIATE 'DROP TABLE Customers';

  EXECUTE IMMEDIATE 'DROP TABLE Loans';

  EXECUTE IMMEDIATE 'DROP TABLE Employees';

  EXECUTE IMMEDIATE 'DROP SEQUENCE AuditLogSeq';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- Create Customers table

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

-- Create Accounts table

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Transactions table

CREATE TABLE Transactions (

    TransactionID NUMBER PRIMARY KEY,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

-- Create Loans table

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Employees table

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

-- FIXED: Create AuditLog without IDENTITY

CREATE TABLE AuditLog (

    LogID NUMBER PRIMARY KEY,

    TransactionID NUMBER,

    AccountID NUMBER,

    Action VARCHAR2(50),

    Timestamp DATE

);

-- Create sequence for AuditLog IDs

CREATE SEQUENCE AuditLogSeq START WITH 1 INCREMENT BY 1;

-- Insert sample data

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

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

COMMIT;

-- Create trigger that uses sequence to insert into AuditLog

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

   INSERT INTO AuditLog (LogID, TransactionID, AccountID, Action, Timestamp)

   VALUES (AuditLogSeq.NEXTVAL, :NEW.TransactionID, :NEW.AccountID, 'INSERTED', SYSDATE);

END;

/

-- Insert a transaction to fire trigger

INSERT INTO Transactions VALUES (3, 1, SYSDATE, 400, 'Deposit');

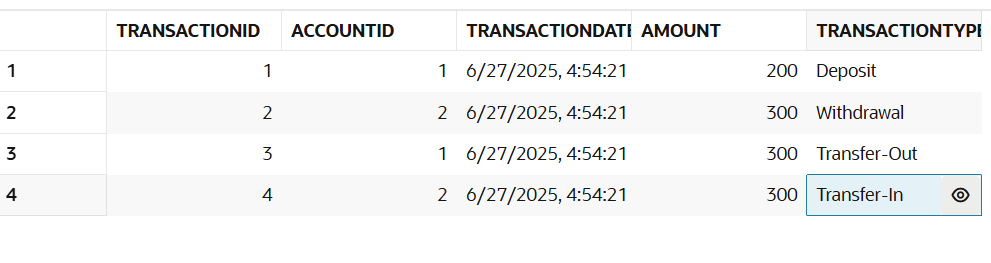
-- View transactions

SELECT \* FROM Transactions;

-- View audit log

SELECT \* FROM AuditLog;

**OUTPUT:**



**Scenario 3:**

-- Optional: Clean up if rerunning

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE Transactions';

  EXECUTE IMMEDIATE 'DROP TABLE Accounts';

  EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- Create Customers table

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

-- Create Accounts table

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Transactions table

CREATE TABLE Transactions (

    TransactionID NUMBER PRIMARY KEY,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

-- Insert sample data

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

COMMIT;

-- Create the trigger to enforce business rules

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

    v\_balance Accounts.Balance%TYPE;

BEGIN

    -- Check if account exists

    SELECT Balance INTO v\_balance

    FROM Accounts

    WHERE AccountID = :NEW.AccountID;

    -- Business Rule 1: Withdrawal > Balance not allowed

    IF :NEW.TransactionType = 'Withdrawal' AND :NEW.Amount > v\_balance THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance for withdrawal.');

    -- Business Rule 2: Amount must be > 0

    ELSIF :NEW.Amount <= 0 THEN

        RAISE\_APPLICATION\_ERROR(-20002, 'Transaction amount must be greater than 0.');

    END IF;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RAISE\_APPLICATION\_ERROR(-20003, 'Account does not exist.');

END;

/

-- Test: Valid Deposit

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 500, 'Deposit');

-- Test: Invalid Withdrawal

BEGIN

    INSERT INTO Transactions VALUES (2, 1, SYSDATE, 5000, 'Withdrawal');

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

-- Test: Negative Amount

BEGIN

    INSERT INTO Transactions VALUES (3, 1, SYSDATE, -100, 'Deposit');

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

END;

/

-- Test: Unknown Account

BEGIN

    INSERT INTO Transactions VALUES (4, 999, SYSDATE, 100, 'Deposit');

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE(SQLERRM);

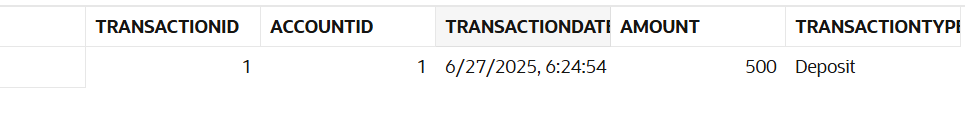
END;

/

-- Final: View successful transaction

SELECT \* FROM Transactions;

**OUTPUT:**



**Exercise 6: Cursors**

**Scenario 1:**

SET SERVEROUTPUT ON;

DECLARE

    -- Cursor to fetch transactions for the current month

    CURSOR txn\_cursor IS

        SELECT c.CustomerID, c.Name AS CustomerName, a.AccountID,

               t.TransactionDate, t.Amount, t.TransactionType

        FROM Customers c

        JOIN Accounts a ON c.CustomerID = a.CustomerID

        JOIN Transactions t ON a.AccountID = t.AccountID

        WHERE TO\_CHAR(t.TransactionDate, 'MM-YYYY') = TO\_CHAR(SYSDATE, 'MM-YYYY')

        ORDER BY c.CustomerID, t.TransactionDate;

    -- Record variable for cursor

    txn\_record txn\_cursor%ROWTYPE;

    current\_customer\_id NUMBER := 0;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('===== Monthly Transaction Statements =====');

    OPEN txn\_cursor;

    LOOP

        FETCH txn\_cursor INTO txn\_record;

        EXIT WHEN txn\_cursor%NOTFOUND;

        -- Print header when new customer begins

        IF txn\_record.CustomerID != current\_customer\_id THEN

            current\_customer\_id := txn\_record.CustomerID;

            DBMS\_OUTPUT.PUT\_LINE(CHR(10) || 'Customer: ' || txn\_record.CustomerName || ' (ID: ' || txn\_record.CustomerID || ')');

            DBMS\_OUTPUT.PUT\_LINE('---------------------------------------------------');

            DBMS\_OUTPUT.PUT\_LINE('Date       | AccountID | Type       | Amount');

            DBMS\_OUTPUT.PUT\_LINE('---------------------------------------------------');

        END IF;

        -- Print each transaction

        DBMS\_OUTPUT.PUT\_LINE(

            TO\_CHAR(txn\_record.TransactionDate, 'DD-MON-YYYY') || ' | ' ||

            txn\_record.AccountID || '        | ' ||

            RPAD(txn\_record.TransactionType, 10) || ' | ' ||

            txn\_record.Amount

        );

    END LOOP;

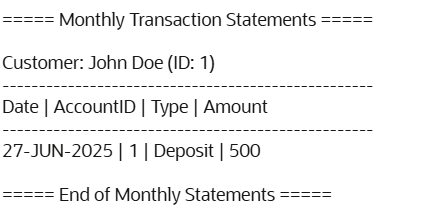
    CLOSE txn\_cursor;

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '===== End of Monthly Statements =====');

END;

/

**OUTPUT:**



**Scenario 2:**

-- Enable output

SET SERVEROUTPUT ON;

-- Step 1: Optional - Clean up if re-running

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE Transactions';

  EXECUTE IMMEDIATE 'DROP TABLE Accounts';

  EXECUTE IMMEDIATE 'DROP TABLE Customers';

  EXECUTE IMMEDIATE 'DROP TABLE Loans';

  EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- Step 2: Create tables

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

);

-- Step 3: Insert Sample Data

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

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

COMMIT;

-- Step 4: Apply Annual Fee with Cursor (PL/SQL Block)

DECLARE

    CURSOR acc\_cursor IS

        SELECT AccountID, Balance

        FROM Accounts;

    v\_acc acc\_cursor%ROWTYPE;

    v\_fee CONSTANT NUMBER := 100;  -- Flat annual fee

BEGIN

    OPEN acc\_cursor;

    LOOP

        FETCH acc\_cursor INTO v\_acc;

        EXIT WHEN acc\_cursor%NOTFOUND;

        UPDATE Accounts

        SET Balance = Balance - v\_fee,

            LastModified = SYSDATE

        WHERE AccountID = v\_acc.AccountID;

        DBMS\_OUTPUT.PUT\_LINE('Annual fee of ' || v\_fee || ' applied to Account ID: ' || v\_acc.AccountID);

    END LOOP;

    CLOSE acc\_cursor;

    COMMIT;

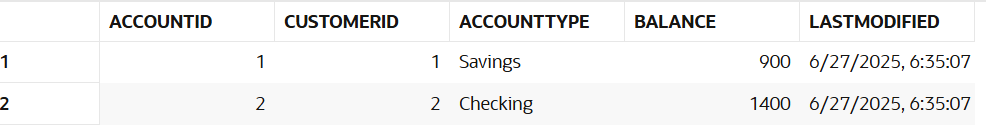
END;

/

-- Step 5: Verify balances after fee deduction

SELECT \* FROM Accounts;

**OUTPUT:**



**Scenario 3:**

-- Enable output for DBMS\_OUTPUT

SET SERVEROUTPUT ON;

-- 1. Optional Cleanup

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE Transactions';

  EXECUTE IMMEDIATE 'DROP TABLE Accounts';

  EXECUTE IMMEDIATE 'DROP TABLE Customers';

  EXECUTE IMMEDIATE 'DROP TABLE Loans';

  EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- 2. Create Tables

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

);

-- 3. Insert Sample Data

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Loans VALUES (2, 2, 4000, 4.5, SYSDATE, ADD\_MONTHS(SYSDATE, 48));

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

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

COMMIT;

-- 4. PL/SQL Block to Update Interest Rates Based on Loan Policy

DECLARE

    CURSOR loan\_cur IS

        SELECT LoanID, LoanAmount, InterestRate FROM Loans;

    v\_loan loan\_cur%ROWTYPE;

    v\_new\_rate NUMBER;

BEGIN

    OPEN loan\_cur;

    LOOP

        FETCH loan\_cur INTO v\_loan;

        EXIT WHEN loan\_cur%NOTFOUND;

        IF v\_loan.LoanAmount >= 5000 THEN

            v\_new\_rate := v\_loan.InterestRate + 1.5;

        ELSE

            v\_new\_rate := v\_loan.InterestRate + 1.0;

        END IF;

        UPDATE Loans

        SET InterestRate = v\_new\_rate

        WHERE LoanID = v\_loan.LoanID;

        DBMS\_OUTPUT.PUT\_LINE('LoanID: ' || v\_loan.LoanID ||

                             ' | Old Rate: ' || v\_loan.InterestRate ||

                             ' | New Rate: ' || v\_new\_rate);

    END LOOP;

    CLOSE loan\_cur;

    COMMIT;

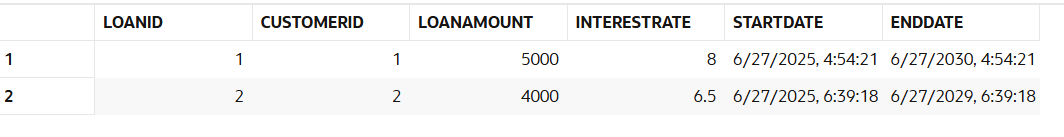
END;

/

-- 5. View updated Loans table

SELECT \* FROM Loans;

**OUTPUT:**



**Exercise 7: Packages**

**Scenario 1:**

-- Enable output

SET SERVEROUTPUT ON;

-- 1. Create Customers table

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

-- 2. Insert sample customers

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

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

COMMIT;

-- 3. Create Package Specification

CREATE OR REPLACE PACKAGE CustomerManagement AS

  PROCEDURE AddCustomer(

    p\_CustomerID IN NUMBER,

    p\_Name       IN VARCHAR2,

    p\_DOB        IN DATE,

    p\_Balance    IN NUMBER

  );

  PROCEDURE UpdateCustomer(

    p\_CustomerID IN NUMBER,

    p\_Name       IN VARCHAR2,

    p\_DOB        IN DATE,

    p\_Balance    IN NUMBER

  );

  FUNCTION GetCustomerBalance(

    p\_CustomerID IN NUMBER

  ) RETURN NUMBER;

END CustomerManagement;

/

-- 4. Create Package Body

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

  PROCEDURE AddCustomer(

    p\_CustomerID IN NUMBER,

    p\_Name       IN VARCHAR2,

    p\_DOB        IN DATE,

    p\_Balance    IN NUMBER

  ) IS

  BEGIN

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

    VALUES (p\_CustomerID, p\_Name, p\_DOB, p\_Balance, SYSDATE);

  END AddCustomer;

  PROCEDURE UpdateCustomer(

    p\_CustomerID IN NUMBER,

    p\_Name       IN VARCHAR2,

    p\_DOB        IN DATE,

    p\_Balance    IN NUMBER

  ) IS

  BEGIN

    UPDATE Customers

    SET Name = p\_Name,

        DOB = p\_DOB,

        Balance = p\_Balance,

        LastModified = SYSDATE

    WHERE CustomerID = p\_CustomerID;

  END UpdateCustomer;

  FUNCTION GetCustomerBalance(

    p\_CustomerID IN NUMBER

  ) RETURN NUMBER IS

    v\_balance NUMBER;

  BEGIN

    SELECT Balance INTO v\_balance

    FROM Customers

    WHERE CustomerID = p\_CustomerID;

    RETURN v\_balance;

  END GetCustomerBalance;

END CustomerManagement;

/

-- 5. Test the package

-- a. Add a new customer

BEGIN

  CustomerManagement.AddCustomer(3, 'Charlie King', TO\_DATE('1992-11-30','YYYY-MM-DD'), 1200);

END;

/

-- b. Update customer details

BEGIN

  CustomerManagement.UpdateCustomer(3, 'Charles King', TO\_DATE('1992-11-30','YYYY-MM-DD'), 1300);

END;

/

-- c. Get balance

DECLARE

  v\_bal NUMBER;

BEGIN

  v\_bal := CustomerManagement.GetCustomerBalance(3);

  DBMS\_OUTPUT.PUT\_LINE('Customer Balance: ' || v\_bal);

END;

/

-- d. See table contents

SELECT \* FROM Customers;

**OUTPUT:**



**Scenario 2:**

-- Optional: Drop table if it already exists

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

  WHEN OTHERS THEN

    NULL;

END;

/

-- Create Employees table

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

-- Insert initial sample data

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

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

COMMIT;

-- Create package specification

CREATE OR REPLACE PACKAGE EmployeeManagement AS

    PROCEDURE HireEmployee(

        p\_EmployeeID IN NUMBER,

        p\_Name IN VARCHAR2,

        p\_Position IN VARCHAR2,

        p\_Salary IN NUMBER,

        p\_Department IN VARCHAR2,

        p\_HireDate IN DATE

    );

    PROCEDURE UpdateEmployeeDetails(

        p\_EmployeeID IN NUMBER,

        p\_Salary IN NUMBER,

        p\_Position IN VARCHAR2

    );

    FUNCTION GetAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

-- Create package body

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

    PROCEDURE HireEmployee(

        p\_EmployeeID IN NUMBER,

        p\_Name IN VARCHAR2,

        p\_Position IN VARCHAR2,

        p\_Salary IN NUMBER,

        p\_Department IN VARCHAR2,

        p\_HireDate IN DATE

    ) IS

    BEGIN

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

        VALUES (p\_EmployeeID, p\_Name, p\_Position, p\_Salary, p\_Department, p\_HireDate);

    END;

    PROCEDURE UpdateEmployeeDetails(

        p\_EmployeeID IN NUMBER,

        p\_Salary IN NUMBER,

        p\_Position IN VARCHAR2

    ) IS

    BEGIN

        UPDATE Employees

        SET Salary = p\_Salary,

            Position = p\_Position

        WHERE EmployeeID = p\_EmployeeID;

    END;

    FUNCTION GetAnnualSalary(p\_EmployeeID IN NUMBER) RETURN NUMBER IS

        v\_Salary Employees.Salary%TYPE;

    BEGIN

        SELECT Salary INTO v\_Salary

        FROM Employees

        WHERE EmployeeID = p\_EmployeeID;

        RETURN v\_Salary \* 12;

    END;

END EmployeeManagement;

/

-- Test block

SET SERVEROUTPUT ON;

BEGIN

    -- Add new employee

    EmployeeManagement.HireEmployee(

        p\_EmployeeID => 3,

        p\_Name => 'Charlie Ray',

        p\_Position => 'Tester',

        p\_Salary => 50000,

        p\_Department => 'QA',

        p\_HireDate => SYSDATE

    );

    -- Update details

    EmployeeManagement.UpdateEmployeeDetails(

        p\_EmployeeID => 3,

        p\_Salary => 52000,

        p\_Position => 'Senior Tester'

    );

    -- Display annual salary

    DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.GetAnnualSalary(3));

END;

/

-- View the result in the table

SELECT \* FROM Employees;

**OUTPUT:**



**Scenario 3:**

-- Optional: Drop tables if they already exist

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE Transactions';

  EXECUTE IMMEDIATE 'DROP TABLE Accounts';

  EXECUTE IMMEDIATE 'DROP TABLE Customers';

  EXECUTE IMMEDIATE 'DROP TABLE Loans';

  EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- Create tables

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

);

-- Insert sample data

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

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

INSERT INTO Accounts VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

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

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

COMMIT;

-- Create package specification

CREATE OR REPLACE PACKAGE AccountOperations AS

    PROCEDURE OpenAccount(

        p\_AccountID IN NUMBER,

        p\_CustomerID IN NUMBER,

        p\_AccountType IN VARCHAR2,

        p\_Balance IN NUMBER

    );

    PROCEDURE CloseAccount(

        p\_AccountID IN NUMBER

    );

    FUNCTION GetTotalBalance(p\_CustomerID IN NUMBER) RETURN NUMBER;

END AccountOperations;

/

-- Create package body

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

    PROCEDURE OpenAccount(

        p\_AccountID IN NUMBER,

        p\_CustomerID IN NUMBER,

        p\_AccountType IN VARCHAR2,

        p\_Balance IN NUMBER

    ) IS

    BEGIN

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

        VALUES (p\_AccountID, p\_CustomerID, p\_AccountType, p\_Balance, SYSDATE);

    END;

    PROCEDURE CloseAccount(

        p\_AccountID IN NUMBER

    ) IS

    BEGIN

        DELETE FROM Accounts WHERE AccountID = p\_AccountID;

    END;

    FUNCTION GetTotalBalance(p\_CustomerID IN NUMBER) RETURN NUMBER IS

        v\_total NUMBER;

    BEGIN

        SELECT NVL(SUM(Balance), 0)

        INTO v\_total

        FROM Accounts

        WHERE CustomerID = p\_CustomerID;

        RETURN v\_total;

    END;

END AccountOperations;

/

-- Enable output

SET SERVEROUTPUT ON;

-- Test the package

BEGIN

    -- Open a new account

    AccountOperations.OpenAccount(3, 1, 'Fixed Deposit', 8000);

    -- Get total balance of customer 1

    DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer 1: ' || AccountOperations.GetTotalBalance(1));

    -- Close the account we just created

    AccountOperations.CloseAccount(3);

    -- Show balance again after deletion

    DBMS\_OUTPUT.PUT\_LINE('After Closing Account - Total Balance: ' || AccountOperations.GetTotalBalance(1));

END;

/

-- View accounts after test

SELECT \* FROM Accounts;

**OUTPUT:**

