**Exercise 1: Control Structures**

**Table Creation:**

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE,

    IsVIP VARCHAR2(5) DEFAULT 'FALSE'

);

ALTER TABLE Customers ADD IsVIP VARCHAR2(5) DEFAULT 'FALSE';

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 12000, SYSDATE, 'FALSE');

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 8000, SYSDATE, 'FALSE');

INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 15));

COMMIT;

**Scenario 1:**

**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

  FOR rec IN (

    SELECT CustomerID

    FROM Customers

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

  ) LOOP

    UPDATE Loans

    SET InterestRate = InterestRate - 1

    WHERE CustomerID = rec.CustomerID;

  END LOOP;

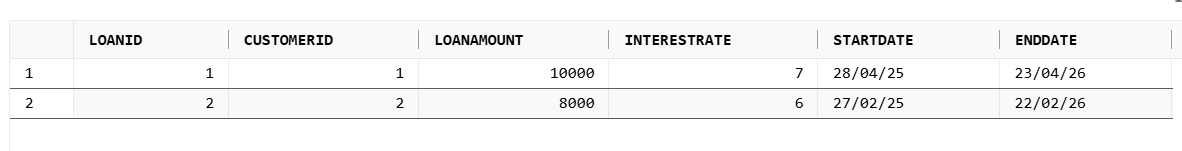
  COMMIT;

END;

/

SELECT \* FROM Loans;

**Output**



**Scenario 2:**

**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.

BEGIN

  FOR rec IN (

    SELECT CustomerID

    FROM Customers

    WHERE Balance > 10000

  ) LOOP

    UPDATE Customers

    SET IsVIP = 'TRUE'

    WHERE CustomerID = rec.CustomerID;

  END LOOP;

  COMMIT;

END;

/

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

**Output**



**Scenario 3:**

**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;

DECLARE

  v\_name Customers.Name%TYPE;

BEGIN

  FOR rec IN (

    SELECT LoanID, CustomerID, EndDate

    FROM Loans

    WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

    BEGIN

      SELECT Name INTO v\_name

      FROM Customers

      WHERE CustomerID = rec.CustomerID;

      DBMS\_OUTPUT.PUT\_LINE(

        'Reminder: ' || v\_name || ', your loan (ID ' || rec.LoanID ||

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

      );

    EXCEPTION

      WHEN NO\_DATA\_FOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('No customer found for Loan ID: ' || rec.LoanID);

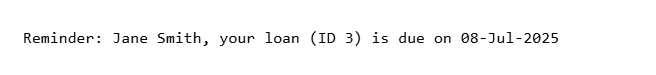
    END;

  END LOOP;

END;

/

**Output**



**Exercise 2: Error Handling**

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 Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

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 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;

**Scenario 1:**

**Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

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 FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

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

  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('Transfer successful.');

EXCEPTION

  WHEN OTHERS THEN

    ROLLBACK;

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

END;

/

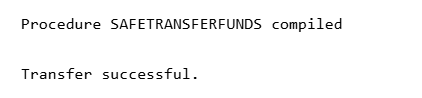
BEGIN

    SafeTransferFunds(1, 2, 500);

    END;

    /

**Output**



**Scenario 2:**

**Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

CREATE OR REPLACE PROCEDURE UpdateSalary (

  p\_emp\_id   IN NUMBER,

  p\_percent  IN NUMBER

)

IS

BEGIN

  UPDATE Employees

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

  WHERE EmployeeID = p\_emp\_id;

  IF SQL%ROWCOUNT = 0 THEN

    RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID not found.');

  END IF;

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('Salary updated.');

EXCEPTION

  WHEN OTHERS THEN

    ROLLBACK;

    DBMS\_OUTPUT.PUT\_LINE('Salary update failed: ' || SQLERRM);

END;

/

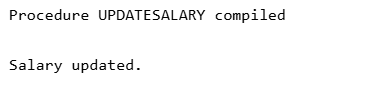
BEGIN

    UpdateSalary(1, 10);

END;

/

**Output**



**Scenario 3:**

**Question:** Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

CREATE OR REPLACE PROCEDURE AddNewCustomer (

  p\_customer\_id 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\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('Customer added.');

EXCEPTION

  WHEN DUP\_VAL\_ON\_INDEX THEN

    DBMS\_OUTPUT.PUT\_LINE('Customer ID already exists.');

  WHEN OTHERS THEN

    ROLLBACK;

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

END;

/

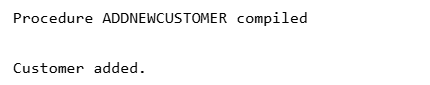
BEGIN

    AddNewCustomer(3, 'Samuel Jackson', TO\_DATE('1975-01-10','YYYY-MM-DD'), 4000);

    END;

    /

**Output**



**Exercise 3: Stored Procedures**

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE

);

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 10000, SYSDATE);

INSERT INTO Accounts VALUES (2, 101, 'Checking', 5000, SYSDATE);

INSERT INTO Accounts VALUES (3, 102, 'Savings', 15000, SYSDATE);

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;

**Scenario 1:**

**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.

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

BEGIN

  UPDATE Accounts

  SET Balance = Balance + (Balance \* 0.01)

  WHERE AccountType = 'Savings';

  COMMIT;

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

EXCEPTION

  WHEN OTHERS THEN

    ROLLBACK;

    DBMS\_OUTPUT.PUT\_LINE('Interest application failed: ' || SQLERRM);

END;

/

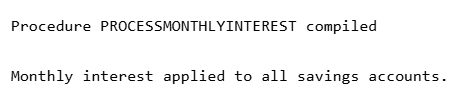
BEGIN

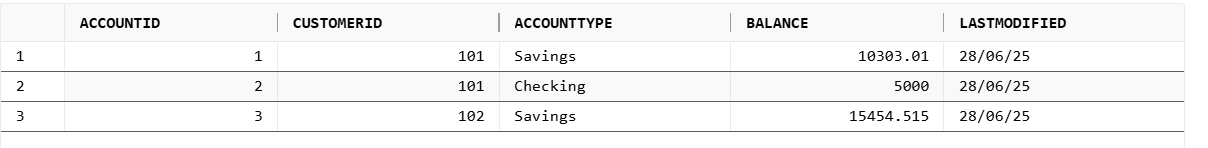
  ProcessMonthlyInterest;

END;

/

**Output**





**Scenario 2:**

**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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

  p\_department IN VARCHAR2,

  p\_bonus\_percent IN NUMBER

)

IS

BEGIN

  UPDATE Employees

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

  WHERE Department = p\_department;

  IF SQL%ROWCOUNT = 0 THEN

    RAISE\_APPLICATION\_ERROR(-20002, 'No employees found in department: ' || p\_department);

  END IF;

  COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('✅ Bonus applied to department: ' || p\_department);

EXCEPTION

  WHEN OTHERS THEN

    ROLLBACK;

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

END;

/

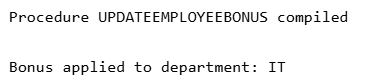
BEGIN

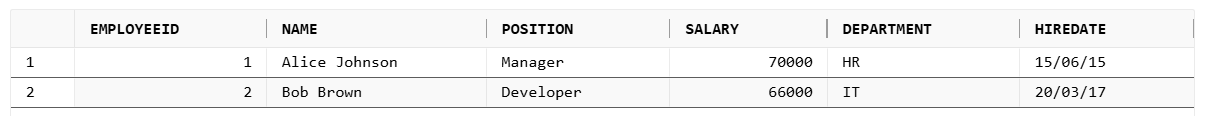
  TransferFunds(1, 2, 3000);

END;

/

**Output**





**Scenario 3:**

**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\_from\_account IN NUMBER,

  p\_to\_account   IN NUMBER,

  p\_amount       IN NUMBER

)

IS

  v\_balance NUMBER;

BEGIN

  SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account FOR UPDATE;

IF v\_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;

  UPDATE Accounts SET Balance = Balance + p\_amount WHERE AccountID = p\_to\_account;

COMMIT;

  DBMS\_OUTPUT.PUT\_LINE('✅ Funds transferred successfully.');

EXCEPTION

  WHEN OTHERS THEN

    ROLLBACK;

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

END;

/

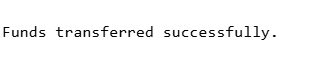
BEGIN

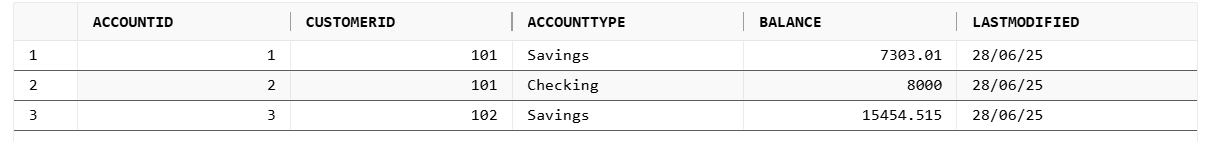
  TransferFunds(1, 2, 3000);

END;

/

**Output**





**Exercise 4: Functions**

CREATE TABLE Customers (

  CustomerID NUMBER PRIMARY KEY,

  Name VARCHAR2(100),

  DOB DATE

);

CREATE TABLE Accounts (

  AccountID NUMBER PRIMARY KEY,

  CustomerID NUMBER,

  Balance NUMBER,

  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1980-05-10', 'YYYY-MM-DD'));

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('2000-07-20', 'YYYY-MM-DD'));

INSERT INTO Accounts VALUES (1, 1, 8000);

INSERT INTO Accounts VALUES (2, 2, 4000);

COMMIT;

**Scenario 1:**

**Question:** Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

CREATE OR REPLACE FUNCTION CalculateAge (p\_dob DATE)

RETURN NUMBER

IS

  v\_age NUMBER;

BEGIN

  v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

  RETURN v\_age;

END;

/DECLARE

  v\_age NUMBER;

BEGIN

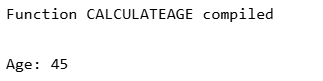
  v\_age := CalculateAge(TO\_DATE('1980-05-10', 'YYYY-MM-DD'));

  DBMS\_OUTPUT.PUT\_LINE('Age: ' || v\_age);

END;

/

**Output**



**Scenario 2:**

**Question:** Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

  p\_loan\_amount NUMBER,

  p\_annual\_rate NUMBER,

  p\_years NUMBER

)

RETURN NUMBER

IS

  v\_monthly\_rate NUMBER := p\_annual\_rate / 12 / 100;

  v\_months NUMBER := p\_years \* 12;

  v\_emi NUMBER;

BEGIN

  IF v\_monthly\_rate = 0 THEN

    v\_emi := p\_loan\_amount / v\_months;

  ELSE

    v\_emi := p\_loan\_amount \* v\_monthly\_rate \* POWER(1 + v\_monthly\_rate, v\_months) /

             (POWER(1 + v\_monthly\_rate, v\_months) - 1);

  END IF;

  RETURN ROUND(v\_emi, 2);

END;

/

DECLARE

  v\_emi NUMBER;

BEGIN

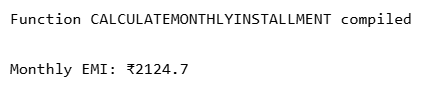
  v\_emi := CalculateMonthlyInstallment(100000, 10, 5); -- ₹100,000, 10% annual, 5 years

  DBMS\_OUTPUT.PUT\_LINE('Monthly EMI: ₹' || v\_emi);

END;

/

**Output**



**Scenario 3:**

**Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

CREATE OR REPLACE FUNCTION HasSufficientBalance (

  p\_account\_id NUMBER,

  p\_amount NUMBER

)

RETURN BOOLEAN

IS

  v\_balance NUMBER;

BEGIN

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

  RETURN v\_balance >= p\_amount;

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

    RETURN FALSE;

END;

/

DECLARE

  v\_result BOOLEAN;

BEGIN

  v\_result := HasSufficientBalance(1, 5000);

  IF v\_result THEN

    DBMS\_OUTPUT.PUT\_LINE('Account 1 has sufficient balance.');

  ELSE

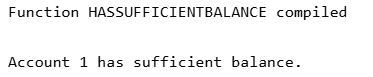
    DBMS\_OUTPUT.PUT\_LINE('Account 1 does NOT have sufficient balance.');

  END IF;

END;

/

**Output**



**Exercise 5: Triggers**

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), -- Deposit or Withdrawal

  FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE AuditLog (

  LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

  TransactionID NUMBER,

  ActionType VARCHAR2(50),

  LogDate DATE

);

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1980-05-10', 'YYYY-MM-DD'), 1000, SYSDATE);

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

COMMIT;

**Scenario 1:**

**Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

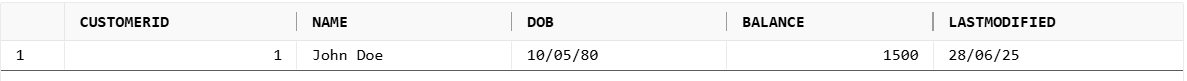
BEGIN

  :NEW.LastModified := SYSDATE;

END;

/

UPDATE Customers SET Balance = 1500 WHERE CustomerID = 1;



**Scenario 2:**

**Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

  INSERT INTO AuditLog (TransactionID, ActionType, LogDate)

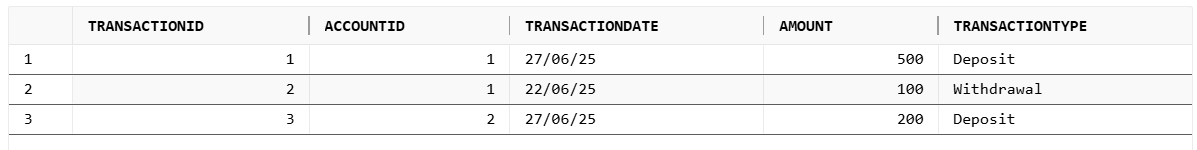
  VALUES (:NEW.TransactionID, 'INSERT', SYSDATE);

END;

/

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

VALUES (1, 1, SYSDATE, 500, 'Deposit');



**Scenario 3:**

**Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

  v\_balance NUMBER;

BEGIN

  SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = :NEW.AccountID;

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

    RAISE\_APPLICATION\_ERROR(-20001, 'Withdrawal amount exceeds balance.');

  ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

    RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');

  END IF;

END;

/

BEGIN

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

  VALUES (2, 1, SYSDATE, 10000, 'Withdrawal');

EXCEPTION

  WHEN OTHERS THEN

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

END;

/

**Output**

**Exercise 6: Cursors**

**Scenario 1:**

**Question:** Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

DECLARE

  CURSOR txn\_cursor IS

    SELECT c.Name, t.TransactionDate, t.Amount, t.TransactionType

    FROM Transactions t

    JOIN Accounts a ON t.AccountID = a.AccountID

    JOIN Customers c ON a.CustomerID = c.CustomerID

    WHERE TRUNC(t.TransactionDate, 'MM') = TRUNC(SYSDATE, 'MM');

  v\_name Customers.Name%TYPE;

  v\_date DATE;

  v\_amount NUMBER;

  v\_type VARCHAR2(10);

DBMS\_OUTPUT.PUT\_LINE('Monthly Statement - Transactions');

  FOR txn IN txn\_cursor LOOP

    DBMS\_OUTPUT.PUT\_LINE(txn.Name || ' | ' ||TO\_CHAR(txn.TransactionDate, 'DD-Mon-YYYY') || ' | ' ||

                         txn.TransactionType || ' | ₹' || txn.Amount);

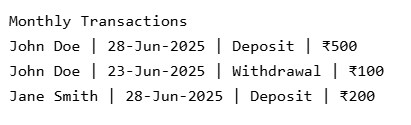
  END LOOP;

COMMIT;

END;

/

**Output**



**Scenario 2:**

**Question:** Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

DECLARE

CURSOR acc\_cursor IS

SELECT AccountID, Balance FROM Accounts

v\_fee NUMBER := 100;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Annual Maintenance Fee Deduction');

    FOR acc IN acc\_cursor LOOP

    IF acc.Balance >= v\_fee THEN

      UPDATE Accounts

      SET Balance = Balance - v\_fee,

          LastModified = SYSDATE

      WHERE AccountID = acc.AccountID;

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

    ELSE

      DBMS\_OUTPUT.PUT\_LINE('Skipped (Low Balance): Account ID: ' || acc.AccountID);

    END IF;

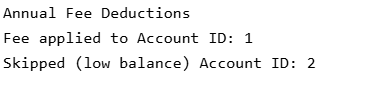
  END LOOP;

COMMIT;

END;

/

**Output**



**Scenario 3:**

**Question:** Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

DECLARE

CURSOR loan\_cursor IS

    SELECT LoanID, InterestRate FROM Loans;

  v\_new\_rate NUMBER;

BEGIN

  DBMS\_OUTPUT.PUT\_LINE('Loan Interest Rate Adjustment');

    FOR l IN loan\_cursor LOOP

    IF l.InterestRate > 6 THEN

      v\_new\_rate := l.InterestRate - 0.5;

    ELSE

      v\_new\_rate := l.InterestRate;

    END IF;

    UPDATE Loans

    SET InterestRate = v\_new\_rate

    WHERE LoanID = l.LoanID;

    DBMS\_OUTPUT.PUT\_LINE('Loan ID ' || l.LoanID || ' new rate: ' || v\_new\_rate || '%');

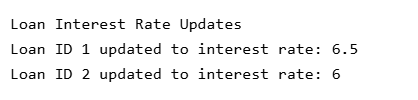
  END LOOP;

  COMMIT;

END;

/

**Output**



**Exercise 7: Packages**

**Scenario 1:**

**Question:** Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

CREATE OR REPLACE PACKAGE CustomerManagement AS

  PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER);

  PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_balance NUMBER);

  FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

  PROCEDURE AddCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_dob DATE, p\_balance NUMBER) IS

  BEGIN

    INSERT INTO Customers VALUES (p\_id, p\_name, p\_dob, p\_balance, SYSDATE);

DBMS\_OUTPUT.PUT\_LINE('Customer added.');

EXCEPTION

    WHEN DUP\_VAL\_ON\_INDEX THEN

      DBMS\_OUTPUT.PUT\_LINE('Customer already exists.');

  END;

  PROCEDURE UpdateCustomer(p\_id NUMBER, p\_name VARCHAR2, p\_balance NUMBER) IS

  BEGIN

    UPDATE Customers

    SET Name = p\_name,

        Balance = p\_balance,

        LastModified = SYSDATE

    WHERE CustomerID = p\_id;

    DBMS\_OUTPUT.PUT\_LINE('Customer updated.');

  END;

  FUNCTION GetCustomerBalance(p\_id NUMBER) RETURN NUMBER IS

    v\_balance NUMBER;

  BEGIN

    SELECT Balance INTO v\_balance FROM Customers WHERE CustomerID = p\_id;

    RETURN v\_balance;

  EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

      RETURN -1;

  END;

END CustomerManagement;

/

SET SERVEROUTPUT ON;

BEGIN

  CustomerManagement.AddCustomer(2, 'Mary Jane', TO\_DATE('1990-02-15','YYYY-MM-DD'), 3000);

  DBMS\_OUTPUT.PUT\_LINE('Balance: ' || CustomerManagement.GetCustomerBalance(2));

END

/

**Output**



**Scenario 2:**

**Question:** Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

CREATE OR REPLACE PACKAGE EmployeeManagement AS

  PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2);

  PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER);

  FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

  PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_pos VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2) IS

  BEGIN

    INSERT INTO Employees VALUES (p\_id, p\_name, p\_pos, p\_salary, p\_dept, SYSDATE);

    DBMS\_OUTPUT.PUT\_LINE('Employee hired.');

  EXCEPTION

    WHEN DUP\_VAL\_ON\_INDEX THEN

      DBMS\_OUTPUT.PUT\_LINE('Employee already exists.');

  END;

  PROCEDURE UpdateEmployee(p\_id NUMBER, p\_salary NUMBER) IS

  BEGIN

    UPDATE Employees SET Salary = p\_salary WHERE EmployeeID = p\_id;

    DBMS\_OUTPUT.PUT\_LINE('Salary updated.');

  END;

  FUNCTION GetAnnualSalary(p\_id NUMBER) RETURN NUMBER IS

    v\_salary NUMBER;

  BEGIN

    SELECT Salary INTO v\_salary FROM Employees WHERE EmployeeID = p\_id;

    RETURN v\_salary \* 12;

  EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

      RETURN -1;

  END;

END EmployeeManagement;

/

SET SERVEROUTPUT ON;

BEGIN

EmployeeManagement.HireEmployee(2, 'Bob', 'Developer', 50000, 'IT');

  DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || EmployeeManagement.GetAnnualSalary(2));

END

/

**Output**



**Scenario 3:**

**Question:** Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

CREATE OR REPLACE PACKAGE AccountOperations AS

  PROCEDURE OpenAccount(p\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER);

  PROCEDURE CloseAccount(p\_id NUMBER);

  FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

  PROCEDURE OpenAccount(p\_id NUMBER, p\_cust\_id NUMBER, p\_type VARCHAR2, p\_balance NUMBER) IS

  BEGIN

    INSERT INTO Accounts VALUES (p\_id, p\_cust\_id, p\_type, p\_balance, SYSDATE);

    DBMS\_OUTPUT.PUT\_LINE('Account opened.');

  EXCEPTION

    WHEN DUP\_VAL\_ON\_INDEX THEN

      DBMS\_OUTPUT.PUT\_LINE('Account already exists.');

  END;

  PROCEDURE CloseAccount(p\_id NUMBER) IS

  BEGIN

    DELETE FROM Accounts WHERE AccountID = p\_id;

    DBMS\_OUTPUT.PUT\_LINE('Account closed.');

  END;

  FUNCTION GetTotalBalance(p\_cust\_id NUMBER) RETURN NUMBER IS

    v\_total NUMBER := 0;

  BEGIN

    SELECT SUM(Balance) INTO v\_total FROM Accounts WHERE CustomerID = p\_cust\_id;

    RETURN NVL(v\_total, 0);

  END;

END AccountOperations;

/

SET SERVEROUTPUT ON;

BEGIN

AccountOperations.OpenAccount(2, 2, 'Checking', 2000);

  DBMS\_OUTPUT.PUT\_LINE('Total Balance for Customer 2: ' || AccountOperations.GetTotalBalance(2));

END;

/

**Output**

