**Exercise 4: Functions**

**Scenario 1:** Calculate the age of customers for eligibility checks.

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

**Code:**

CREATE OR REPLACE FUNCTION GetCustomerAge (

  p\_customer\_id IN NUMBER

) RETURN NUMBER IS

  v\_dob DATE;

  v\_age NUMBER;

BEGIN

  SELECT DOB INTO v\_dob

  FROM Customers

  WHERE CustomerID = p\_customer\_id;

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

  RETURN v\_age;

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

    DBMS\_OUTPUT.PUT\_LINE('❌ Customer not found.');

    RETURN NULL;

  WHEN OTHERS THEN

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

    RETURN NULL;

END;

/

SET SERVEROUTPUT ON;

DECLARE

  v\_age NUMBER;

BEGIN

  v\_age := GetCustomerAge(1);

  IF v\_age IS NOT NULL THEN

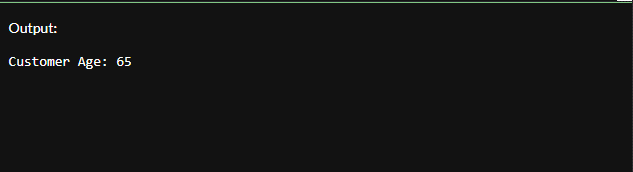
    DBMS\_OUTPUT.PUT\_LINE('Customer Age: ' || v\_age);

  END IF;

END;

/

**Output:**



**Scenario 2:** The bank needs to compute the monthly installment for a loan.

**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.

**Code:**

CREATE TABLE Accounts (

  AccountID NUMBER PRIMARY KEY,

  CustomerID NUMBER,

  AccountType VARCHAR2(20),

  Balance NUMBER,

  LastModified DATE

);

INSERT INTO Accounts VALUES (101, 1, 'Savings', 15000, SYSDATE);

INSERT INTO Accounts VALUES (102, 2, 'Current', 8000, SYSDATE);

COMMIT;

CREATE OR REPLACE FUNCTION GetAccountBalance (

  p\_account\_id IN NUMBER

) RETURN NUMBER IS

  v\_balance NUMBER;

BEGIN

  SELECT Balance INTO v\_balance

  FROM Accounts

  WHERE AccountID = p\_account\_id;

  RETURN v\_balance;

EXCEPTION

  WHEN NO\_DATA\_FOUND THEN

    DBMS\_OUTPUT.PUT\_LINE('❌ Account not found.');

    RETURN NULL;

  WHEN OTHERS THEN

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

    RETURN NULL;

END;

/

SET SERVEROUTPUT ON;

DECLARE

  bal NUMBER;

BEGIN

  bal := GetAccountBalance(101);

  IF bal IS NOT NULL THEN

    DBMS\_OUTPUT.PUT\_LINE('Account Balance: ₹' || bal);

  END IF;

END;



**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

**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.

**Code:**

CREATE OR REPLACE FUNCTION GetTransactionCount (

  p\_account\_id IN NUMBER

) RETURN NUMBER IS

  v\_count NUMBER;

BEGIN

  SELECT COUNT(\*) INTO v\_count

  FROM Transactions

  WHERE AccountID = p\_account\_id;

  RETURN v\_count;

EXCEPTION

  WHEN OTHERS THEN

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

    RETURN NULL;

END;

/

SET SERVEROUTPUT ON;

DECLARE

  t\_count NUMBER;

BEGIN

  t\_count := GetTransactionCount(101);

  IF t\_count IS NOT NULL THEN

    DBMS\_OUTPUT.PUT\_LINE('Transaction Count for Account 101: ' || t\_count);

  END IF;

END;

/

