**Exercise 1: Control Structures**

**WEEK 2 – PL/SQL**

**HANDS-ON (ALL PROBLEMS COMPLETED)**

**9(**

**CREATE TABLE LOANS (**

**LoanID NUMBER PRIMARY KEY,**

**C\_id NUMBER,**

**DueDate DATE,**

**FOREIGN KEY (C\_id) REFERENCES Customers(C\_id)**

**);**

**CREATE TABLE Customers (**

**c\_id NUMBER PRIMARY KEY,**

**Name VARCHAR2(100),**

**Age NUMBER,**

**Balance NUMBER(10,2),**

**IsVIP CHAR(1) DEFAULT 'N',**

**LoanInterest NUMBER(5,2)**

**);**

**CUSTOMER**

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000.00, 'N', 8.5);

INSERT INTO Customers VALUES (2, 'Bob', 45, 8000.00, 'N', 9.0);

INSERT INTO Customers VALUES (3, 'Charlie', 70, 5000.00, 'N', 7.5);

INSERT INTO Customers VALUES (4, 'Diana', 30, 15000.00, 'N', 8.0);

**LOAN**

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, SYSDATE + 35);

INSERT INTO Loans VALUES (103, 3, SYSDATE + 20);

INSERT INTO Loans VALUES (104, 4, SYSDATE + 5);

**PL/SQL – SCENARIO 1**

BEGIN

    FOR cust IN (SELECT C\_ID, LOANINTEREST FROM Customers WHERE Age>60) LOOP

        UPDATE Customers

        SET LOANINTEREST = LOANINTEREST- 1

        WHERE C\_ID=cust.C\_ID;

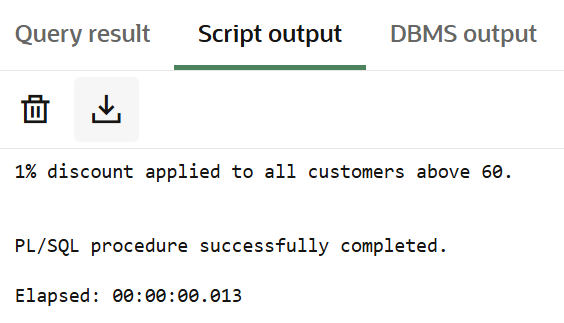
    END LOOP;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('1% discount applied to all customers above 60.');

END;

**OUTPUT:**

****

**PL/SQL – SCENARIO 2**

BEGIN

    FOR cust IN (SELECT C\_ID FROM Customers WHERE Balance > 10000) LOOP

        UPDATE Customers

        SET IsVIP = 'Y'

        WHERE C\_ID = cust.C\_ID;

    END LOOP;

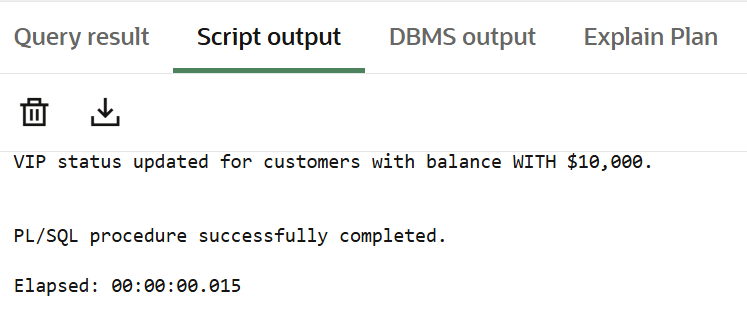
    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('VIP status updated for customers with balance WITH $10,000.');

END;

/

**OUTPUT**

****

**PL/SQL – SCENARIO 3**

BEGIN

    FOR loan IN (

        SELECT L.LoanID, L.DueDate, C.Name, C.C\_ID

        FROM Loans L

        JOIN Customers C ON L.C\_ID = C.C\_ID

        WHERE L.DueDate BETWEEN SYSDATE AND SYSDATE + 30

    ) LOOP

        DBMS\_OUTPUT.PUT\_LINE(

            'Reminder: Dear ' || loan.Name ||

            ', your loan (ID: ' || loan.LoanID ||

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

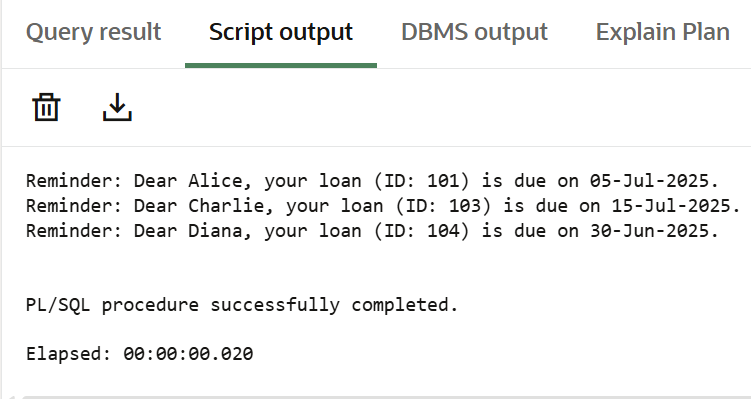
        );

    END LOOP;

END;

/

**OUTPUT**

****

**--------------------------------------------------------------------------------------------------------------------------------------**

**Exercise 2: Error Handling**

**PL/SQL – SCENARIO 1**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_FromAcct IN NUMBER,

p\_ToAcct IN NUMBER,

p\_Amount IN NUMBER) IS

BEGIN

DECLARE

v\_Balance NUMBER;

BEGIN

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

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\_FromAcct;

UPDATE Accounts SET Balance = Balance + p\_Amount WHERE AccountID = p\_ToAcct;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer completed successfully.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

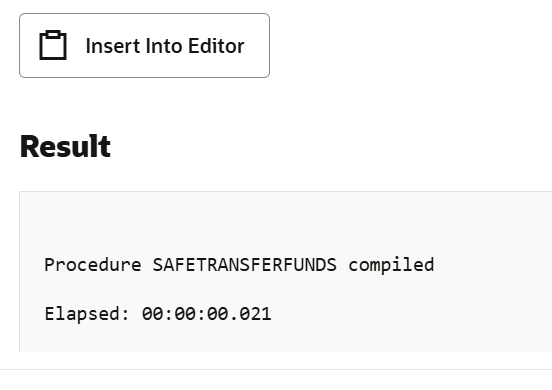
DBMS\_OUTPUT.PUT\_LINE('Error during fund transfer: ' || SQLERRM);

END;

END;

/

**OUTPUT**

****

**PL/SQL SCENARIO 2**

CREATE OR REPLACE PROCEDURE UpdateSalary(

    p\_EmpID IN NUMBER,

    p\_Percent IN NUMBER) IS

BEGIN

    UPDATE Employees

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

    WHERE EmpID = p\_EmpID;

    IF SQL%ROWCOUNT = 0 THEN

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

    END IF;

    COMMIT;

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

EXCEPTION

    WHEN OTHERS THEN

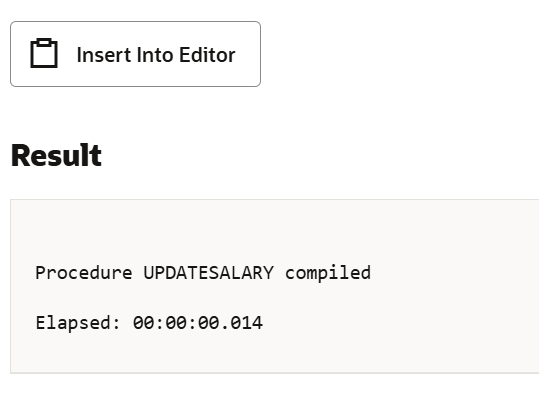
        ROLLBACK;

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

END;

/

**OUTPUT**

****

**PL/SQL SCENARIO 3**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_C\_ID IN NUMBER,

p\_Name IN VARCHAR2,

p\_Age IN NUMBER,

p\_Balance IN NUMBER,

p\_InterestRate IN NUMBER

)

IS

BEGIN

INSERT INTO Customers (C\_ID, Name, Age, Balance, IsVIP, LOANINTEREST)

VALUES (p\_C\_ID, p\_Name, p\_Age, p\_Balance, 'N', p\_InterestRate);

COMMIT;

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

EXCEPTION

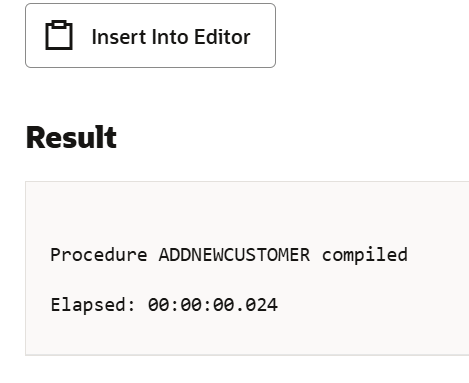
WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer ID already exists. Cannot insert duplicate.');

WHEN OTHERS THEN

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

END;

****

**--------------------------------------------------------------------------------------------------------------------------------------**

**Exercise 3: Stored Procedures**

**PL/SQL SCENARIO 1**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest is

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

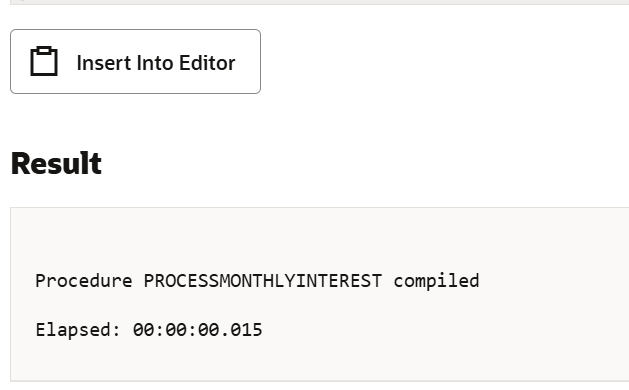
COMMIT;

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

END;

/

**OUTPUT**



**PL/SQL SCENARIO 2**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonusByID(

    p\_EmpID IN NUMBER,

    p\_BonusPercent IN NUMBER

)

IS

BEGIN

    UPDATE Employees

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

    WHERE EmpID = p\_EmpID;

    IF SQL%ROWCOUNT > 0 THEN

        COMMIT;

        DBMS\_OUTPUT.PUT\_LINE('Bonus applied to employee ID: ' || p\_EmpID);

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('No employee found with ID: ' || p\_EmpID);

    END IF;

EXCEPTION

    WHEN OTHERS THEN

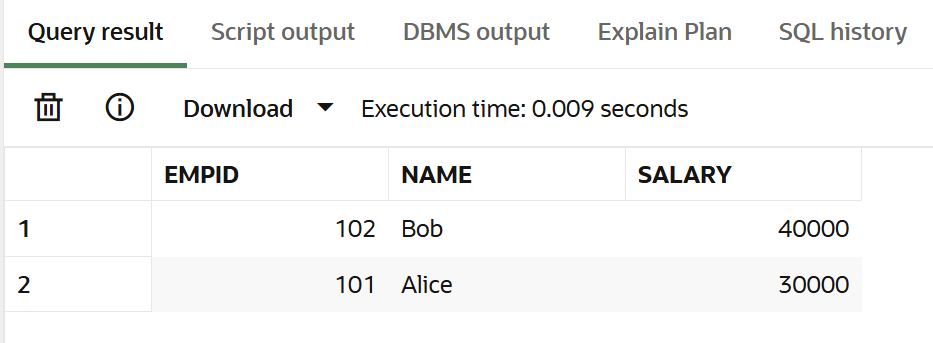
        ROLLBACK;

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

END;

/

**OUTPUT**



**PL/SQL SCENARIO 3**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAcct IN NUMBER,

p\_ToAcct IN NUMBER,

p\_Amount IN NUMBER

v\_Balance NUMBER)IS

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts2 WHERE AccountID = p\_FromAcct;

IF v\_Balance < p\_Amount THEN

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

END IF;

UPDATE Accounts2

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAcct;

UPDATE Accounts2

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAcct;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful from account ' || p\_FromAcct || ' to account ' || p\_ToAcct);

EXCEPTION

WHEN OTHERS THEN

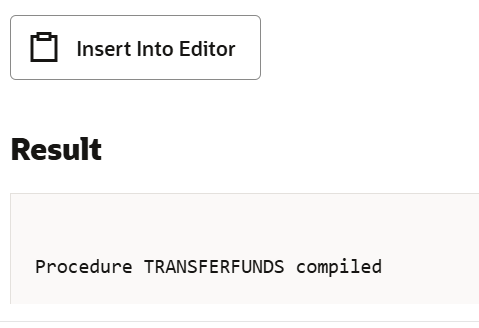
ROLLBACK;

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

END;

/

**OUTPUT**



**--------------------------------------------------------------------------------------------------------------------------------------**

**Exercise 4: Functions**

**PL/SQL SCENARIO 1**

CREATE OR REPLACE FUNCTION CalculateAge(p\_DOB DATE) RETURN NUMBER IS

v\_Age NUMBER;

BEGIN

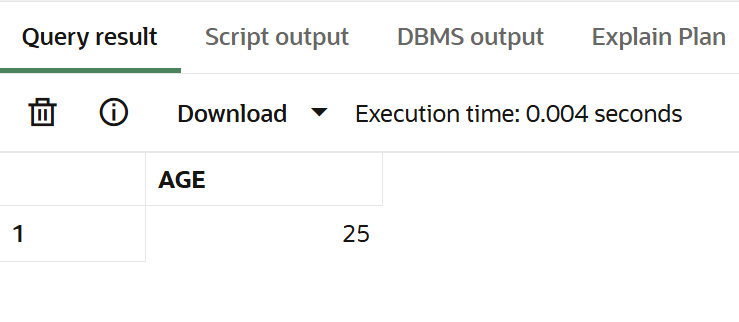
v\_Age := TRUNC(MONTHS\_BETWEEN(SYSDATE, p\_DOB) / 12);

RETURN v\_Age;

END;

/

**OUTPUT**



**PL/SQL SCENARIO 2**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

    p\_LoanAmount NUMBER,

    p\_AnnualRate NUMBER,

    p\_DurationYears NUMBER

) RETURN NUMBER

IS

    v\_MonthlyRate NUMBER := p\_AnnualRate / 12 / 100;

    v\_TotalMonths NUMBER := p\_DurationYears \* 12;

    v\_EMI NUMBER;

BEGIN

    IF v\_MonthlyRate = 0 THEN

        v\_EMI := p\_LoanAmount / v\_TotalMonths;

    ELSE

        v\_EMI := p\_LoanAmount \* v\_MonthlyRate \* POWER(1 + v\_MonthlyRate, v\_TotalMonths) /

                 (POWER(1 + v\_MonthlyRate, v\_TotalMonths) - 1);

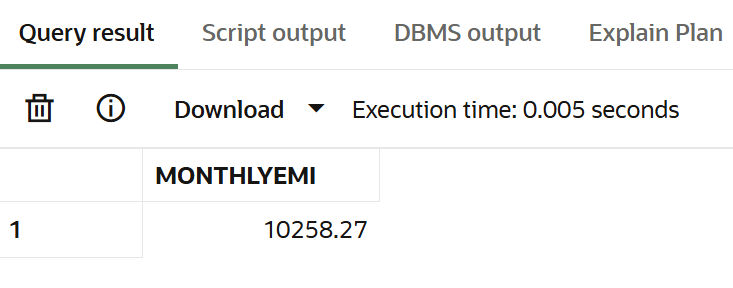
    END IF;

    RETURN ROUND(v\_EMI, 2);

END;

/

**OUTPUT**



**PL/SQL SCENARIO 3**

CREATE OR REPLACE FUNCTION HasSufficientBalance( p\_AccountID NUMBER, p\_Amount NUMBER

) RETURN BOOLEAN IS

    v\_Balance NUMBER;

BEGIN

    SELECT Balance INTO v\_Balance FROM Accounts2 WHERE AccountID = p\_AccountID;

    RETURN v\_Balance >= p\_Amount;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN FALSE;

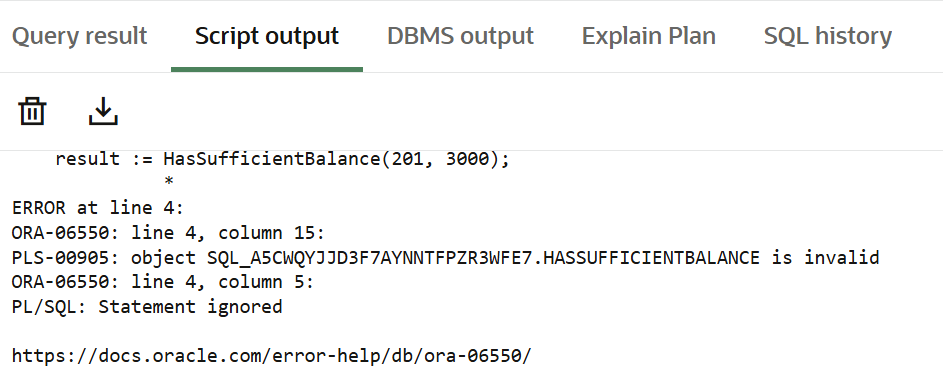
    WHEN OTHERS THEN

        RETURN FALSE;

END;

/

**OUTPUT**

****

**--------------------------------------------------------------------------------------------------------------------------------------**

**Exercise 5: Triggers**

**PL/SQL SCENARIO 1**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

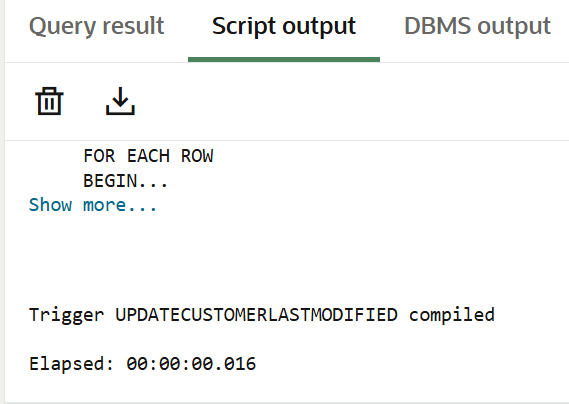
BEGIN

    :NEW.LastModified := SYSDATE;

END;

/

**OUTPUT**

****

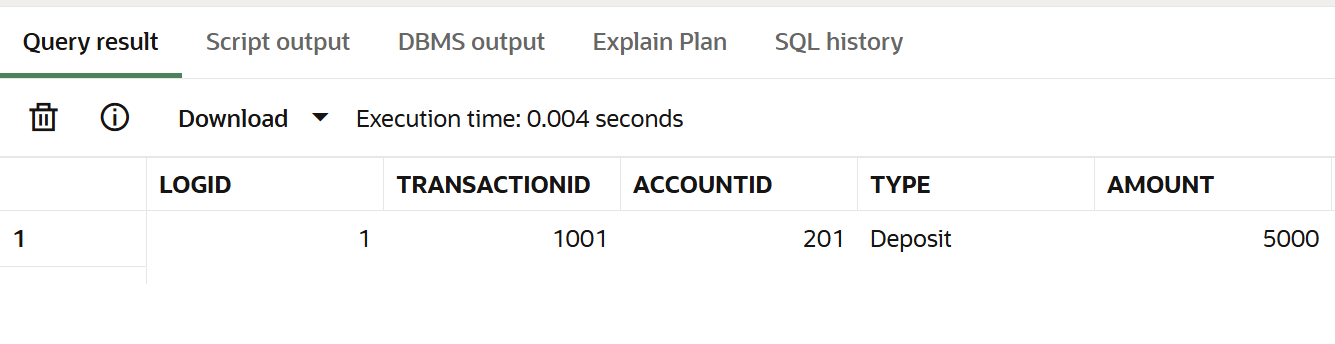
**PL/SQL SCENARIO 2**

INSERT INTO Transactions2 (TransactionID, AccountID, Type, Amount, TransactionDate)

VALUES (1001, 201, 'Deposit', 5000, SYSDATE);

SELECT \* FROM AuditLog;

**OUTPUT**

****

**PL/SQL SCENARIO 3**

CREATE OR REPLACE TRIGGER CheckTransactionRules2

BEFORE INSERT ON Transactions2

FOR EACH ROW

DECLARE

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance FROM Accounts2 WHERE AccountID = :NEW.AccountID;

IF :NEW.Type = 'Withdrawal' THEN

IF :NEW.Amount > v\_Balance THEN

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

ELSIF :NEW.Amount <= 0 THEN

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

END IF;

ELSIF :NEW.Type = 'Deposit' THEN

IF :NEW.Amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20003, 'Deposit amount must be greater than 0.');

END IF;

ELSE

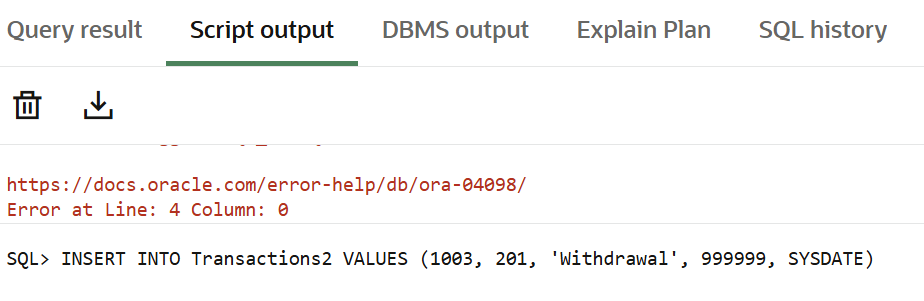
RAISE\_APPLICATION\_ERROR(-20004, 'Invalid transaction type.');

END IF;

END;

/

**OUTPUT**

****

**--------------------------------------------------------------------------------------------------------------------------------------**

**Exercise 6: Cursors**

**PL/SQL SCENARIO 1**

DECLARE

    CURSOR txn\_cur IS

        SELECT AccountID, Type, Amount, TransactionDate

        FROM Transactions2

        WHERE TO\_CHAR(TransactionDate, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY');

    v\_acctID Transactions2.AccountID%TYPE;

    v\_type   Transactions2.Type%TYPE;

    v\_amt    Transactions2.Amount%TYPE;

    v\_date   Transactions2.TransactionDate%TYPE;

BEGIN

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

    OPEN txn\_cur;

    LOOP

        FETCH txn\_cur INTO v\_acctID, v\_type, v\_amt, v\_date;

        EXIT WHEN txn\_cur%NOTFOUND;

        DBMS\_OUTPUT.PUT\_LINE('Account: ' || v\_acctID ||   ' | ' || v\_type ||  ' | Amount: ' ' | Date: ' || TO\_CHAR(v\_date, 'DD-Mon-YYYY'));

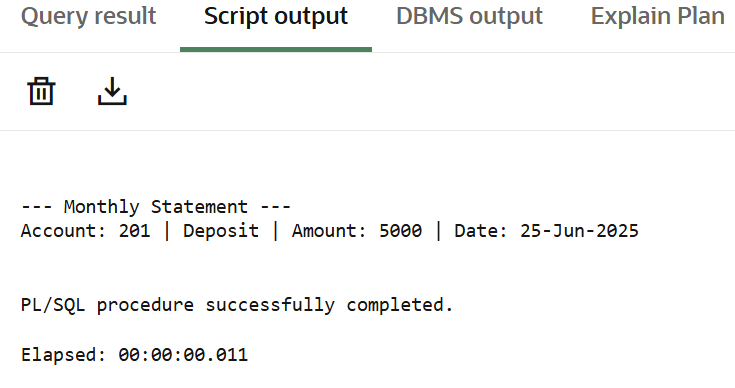
    END LOOP;

    CLOSE txn\_cur;

END;

/

**OUTPUT**

****

**PL/SQL SCENARIO 2**

DECLARE

CURSOR txn\_cur IS

SELECT AccountID, Type, Amount, TransactionDate

FROM Transactions2

WHERE TO\_CHAR(TransactionDate, 'MMYYYY') = TO\_CHAR(SYSDATE, 'MMYYYY');

v\_acctID Transactions2.AccountID%TYPE;

v\_type Transactions2.Type%TYPE;

v\_amt Transactions2.Amount%TYPE;

v\_date Transactions2.TransactionDate%TYPE;

BEGIN

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

OPEN txn\_cur;

LOOP

FETCH txn\_cur INTO v\_acctID, v\_type, v\_amt, v\_date;

EXIT WHEN txn\_cur%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Account: ' || v\_acctID || ' | ' || v\_type || ' | Amount: ' || v\_amt ||

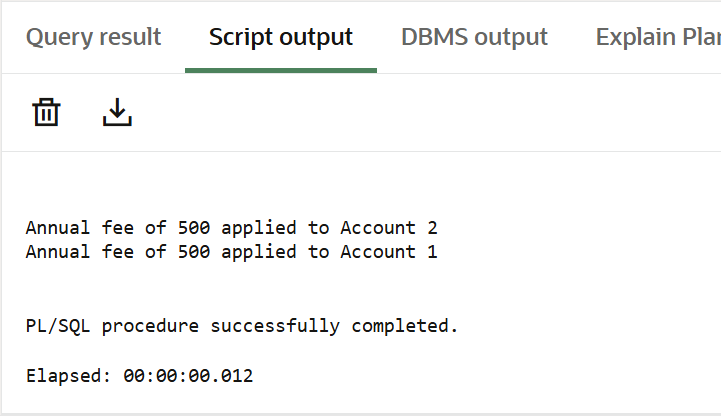
' | Date: ' || TO\_CHAR(v\_date, 'DD-Mon-YYYY'));

END LOOP;

CLOSE txn\_cur;

END; /

**OUTPUT**



**PL/SQL SCENARIO 3**

DECLARE

CURSOR loan\_cur IS

SELECT LoanID, LoanInterestRate FROM Loans;

v\_LoanID Loans.LoanID%TYPE;

v\_Rate Loans.LoanInterestRate%TYPE;

BEGIN

OPEN loan\_cur;

LOOP

FETCH loan\_cur INTO v\_LoanID, v\_Rate;

EXIT WHEN loan\_cur%NOTFOUND;

IF v\_Rate > 10 THEN

UPDATE Loans

SET LoanInterestRate = v\_Rate - 1

WHERE LoanID = v\_LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ' || v\_LoanID || ': Interest reduced to ' || (v\_Rate - 1));

ELSE

UPDATE Loans

SET LoanInterestRate = v\_Rate + 0.5

WHERE LoanID = v\_LoanID;

DBMS\_OUTPUT.PUT\_LINE('Loan ' || v\_LoanID || ': Interest increased to ' || (v\_Rate + 0.5));

END IF;

END LOOP;

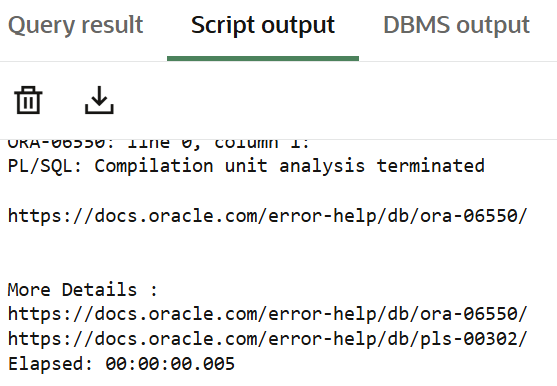
CLOSE loan\_cur;

COMMIT;

END;

/

**OUTPUT**

****

**------------------------------------------------------------------------------------------------------------------------------------**

**Exercise 7: Packages**

**PL/SQL SCENARIO 1**

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_ID NUMBER, p\_Name VARCHAR2, p\_Balance NUMBER) IS

BEGIN

INSERT INTO Customers (CustomerID, Name, Balance)

VALUES (p\_ID, p\_Name, p\_Balance);

END;

PROCEDURE UpdateCustomer(p\_ID NUMBER, p\_Name VARCHAR2) IS

BEGIN

UPDATE Customers

SET Name = p\_Name

WHERE CustomerID = p\_ID;

END;

FUNCTION GetCustomerBalance(p\_ID NUMBER) RETURN NUMBER IS

v\_Balance NUMBER;

BEGIN

SELECT Balance INTO v\_Balance

FROM Customers

WHERE CustomerID = p\_ID;

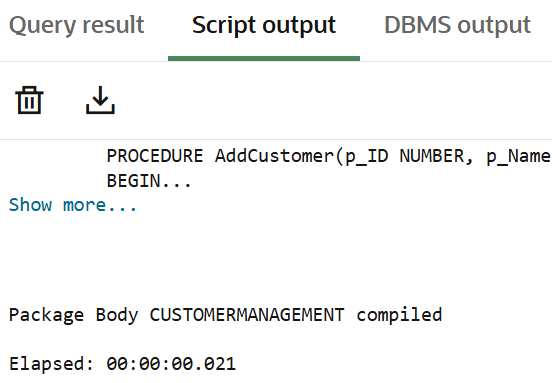
RETURN v\_Balance;

END;

END CustomerManagement;

/

**OUTPUT**

****

**PL/SQL SCENARIO 2**

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_ID NUMBER, p\_Name VARCHAR2, p\_Salary NUMBER) IS

BEGIN

INSERT INTO Employees (EmpID, EmpName, Salary)

VALUES (p\_ID, p\_Name, p\_Salary);

END;

PROCEDURE UpdateEmployee(p\_ID NUMBER, p\_Name VARCHAR2) IS

BEGIN

UPDATE Employees

SET EmpName = p\_Name

WHERE EmpID = p\_ID;

END;

FUNCTION CalculateAnnualSalary(p\_ID NUMBER) RETURN NUMBER IS

v\_Salary NUMBER;

BEGIN

SELECT Salary INTO v\_Salary

FROM Employees

WHERE EmpID = p\_ID;

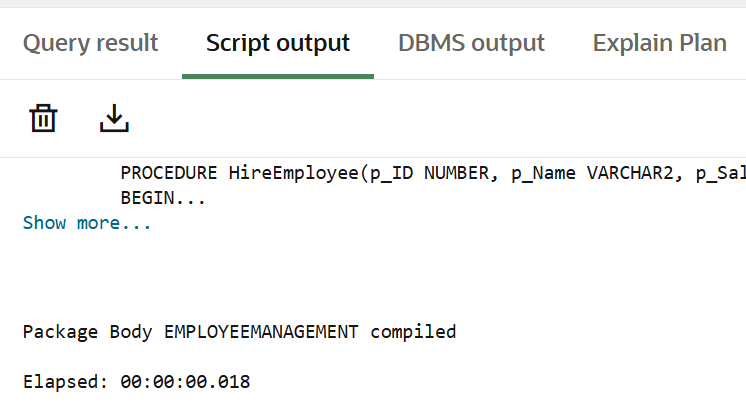
RETURN v\_Salary \* 12;

END;

END EmployeeManagement;

/

**OUTPUT**



**PL/SQL SCENARIO 3**

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

    PROCEDURE OpenAccount(

        p\_AccID NUMBER,

        p\_CustID NUMBER,

        p\_Type VARCHAR2,

        p\_Balance NUMBER) IS

    BEGIN

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

        VALUES (p\_AccID, p\_CustID, p\_Type, p\_Balance);

    END;

    PROCEDURE CloseAccount(p\_AccID NUMBER) IS

    BEGIN

        DELETE FROM Accounts

  WHERE AccountID = p\_AccID;

    END;

    FUNCTION GetTotalBalance(p\_CustID NUMBER) RETURN NUMBER IS

        v\_Total NUMBER;

    BEGIN

        SELECT NVL(SUM(Balance), 0)

        INTO v\_Total

        FROM Accounts

        WHERE CustomerID = p\_CustID;

        RETURN v\_Total;

    END;

END AccountOperations;

/

**OUTPUT**

