**WEEK-2**

**PL/SQL PROGRAMMING HandsOn**

**Superset ID: 6419740**

**TABLE CREATION:**

**Code:**

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),

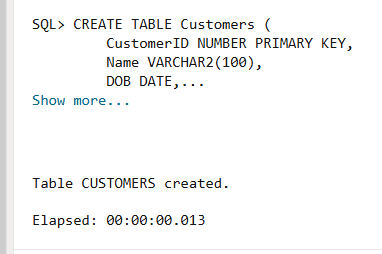
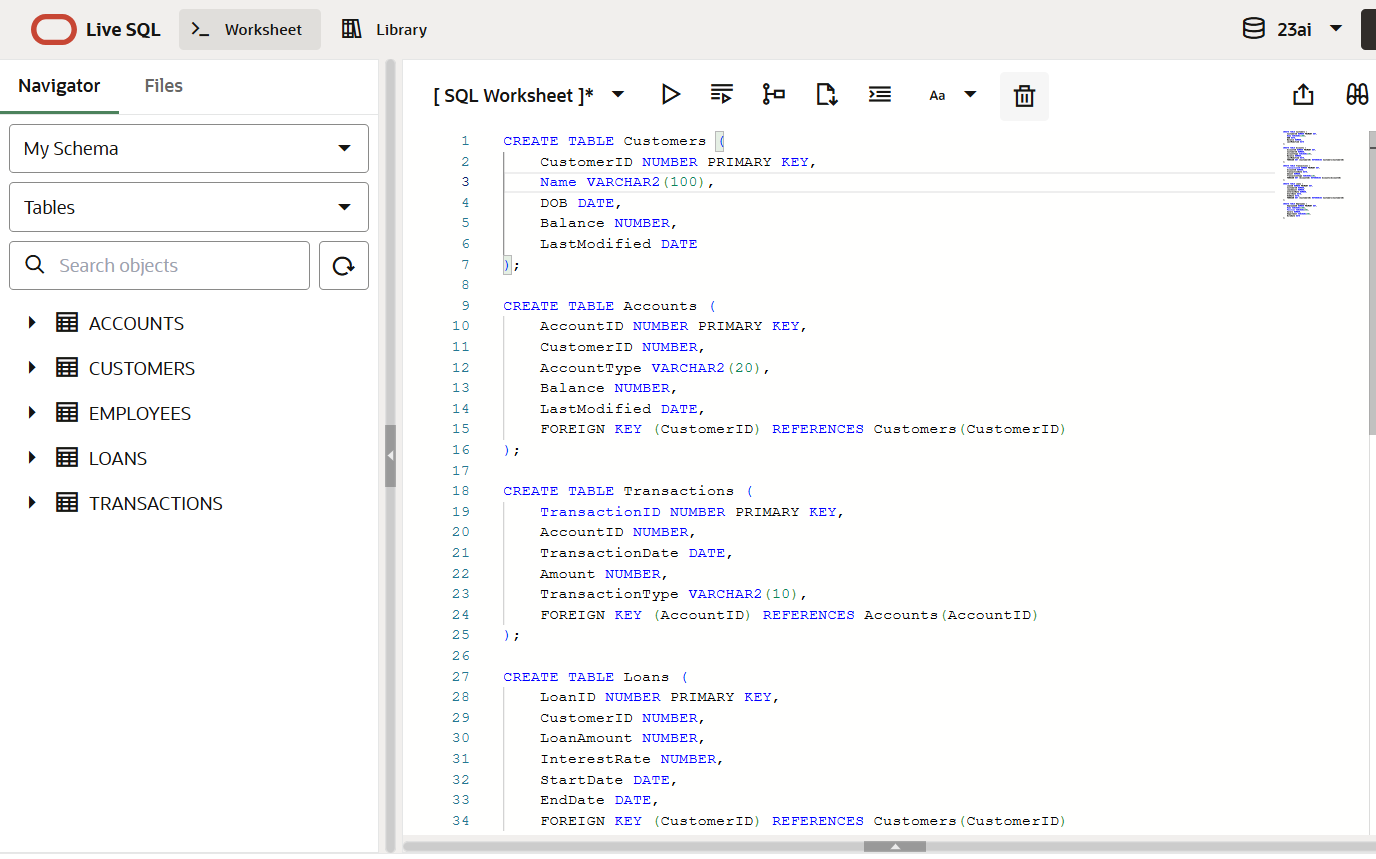
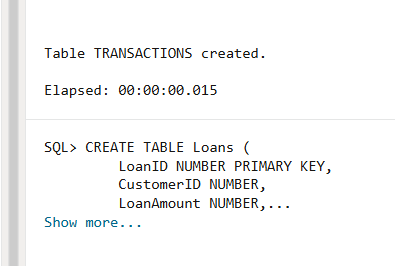
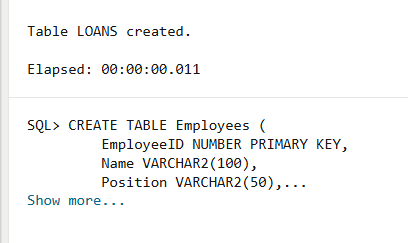
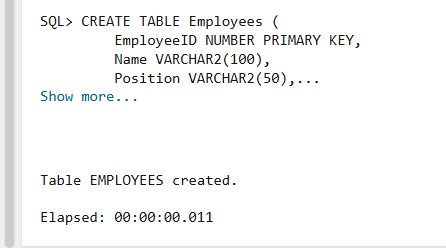
    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

Output:

**INSERTION OF DATA :**

**Code:**

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, 2, 'Checking', 1500, SYSDATE);

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

VALUES (1, 1, SYSDATE, 200, 'Deposit');

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

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

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

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

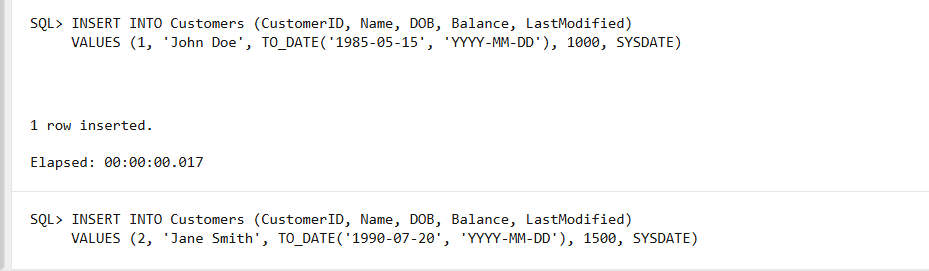
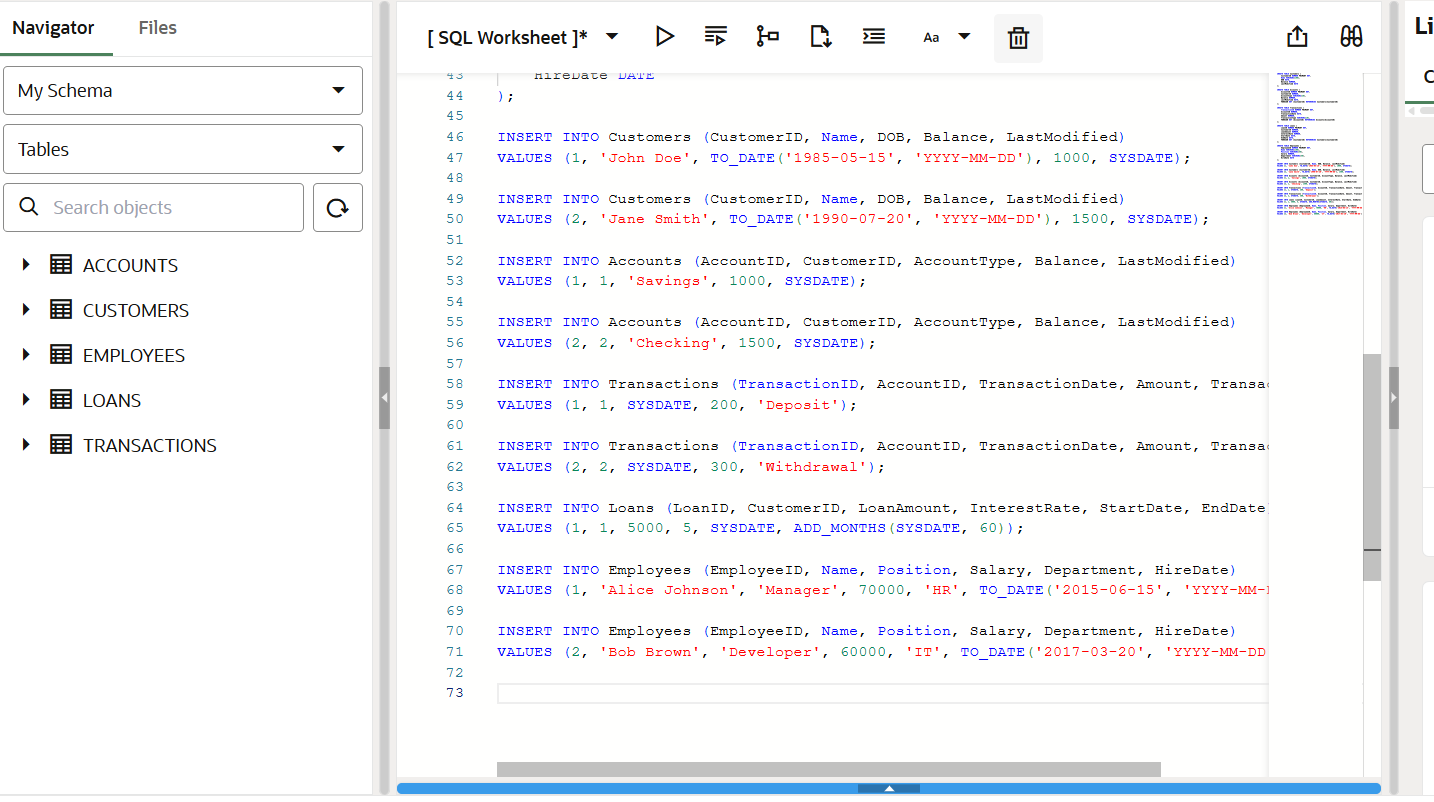
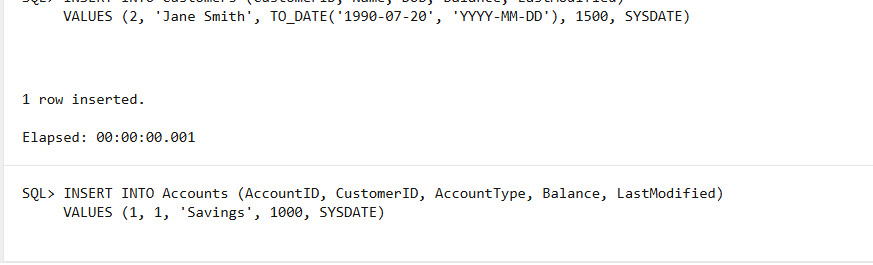
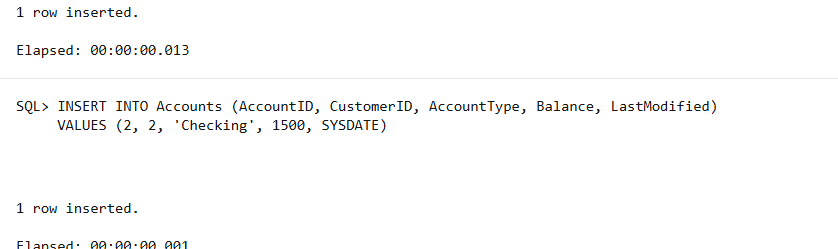
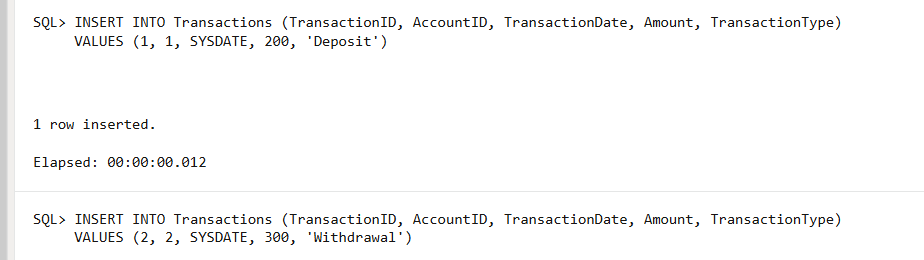
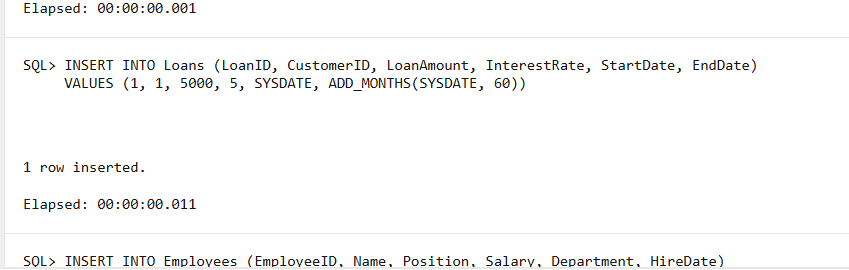
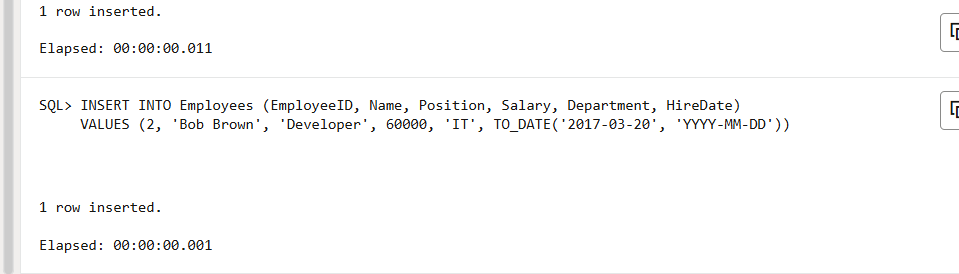
INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

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'));

**Output:**

**Exercise 1: Control Structures**

**Scenario 1:**

**Coding:**

BEGIN

    FOR c IN (SELECT CustomerID, TRUNC(MONTHS\_BETWEEN(SYSDATE, DOB)/12) AS Age

              FROM Customers)

    LOOP

        IF c.Age > 60 THEN

            UPDATE Loans

            SET InterestRate = InterestRate - 1

            WHERE CustomerID = c.CustomerID;

        END IF;

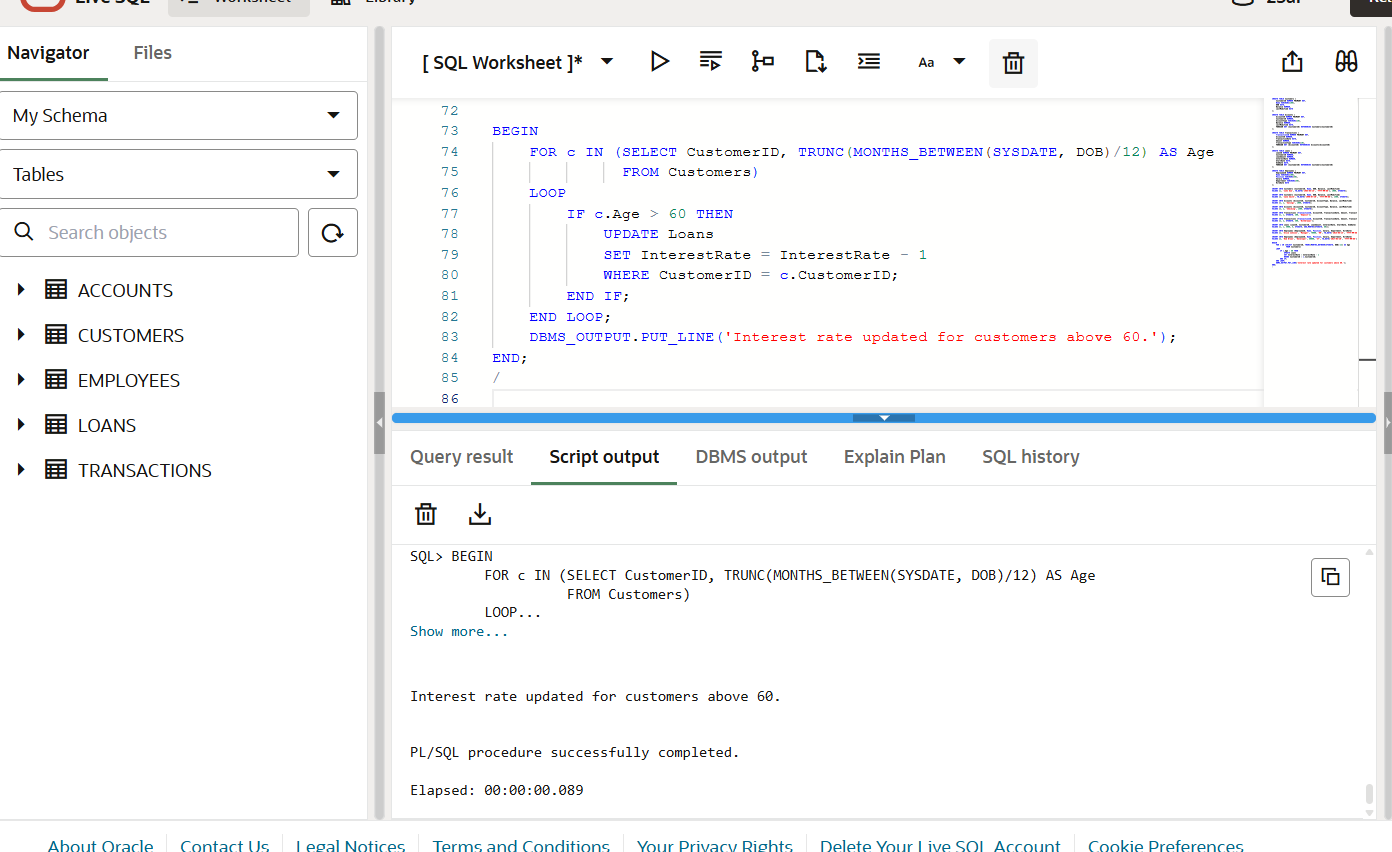
    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE('Interest rate updated for customers above 60.');

END;

/

**Output:**



**Scenario 2:**

**Code:**

ALTER TABLE Customers ADD IsVIP CHAR(1);

BEGIN

    FOR c IN (SELECT CustomerID, Balance FROM Customers)

    LOOP

        IF c.Balance > 10000 THEN

            UPDATE Customers

            SET IsVIP = 'T'

            WHERE CustomerID = c.CustomerID;

        ELSE

            UPDATE Customers

            SET IsVIP = 'F'

            WHERE CustomerID = c.CustomerID;

        END IF;

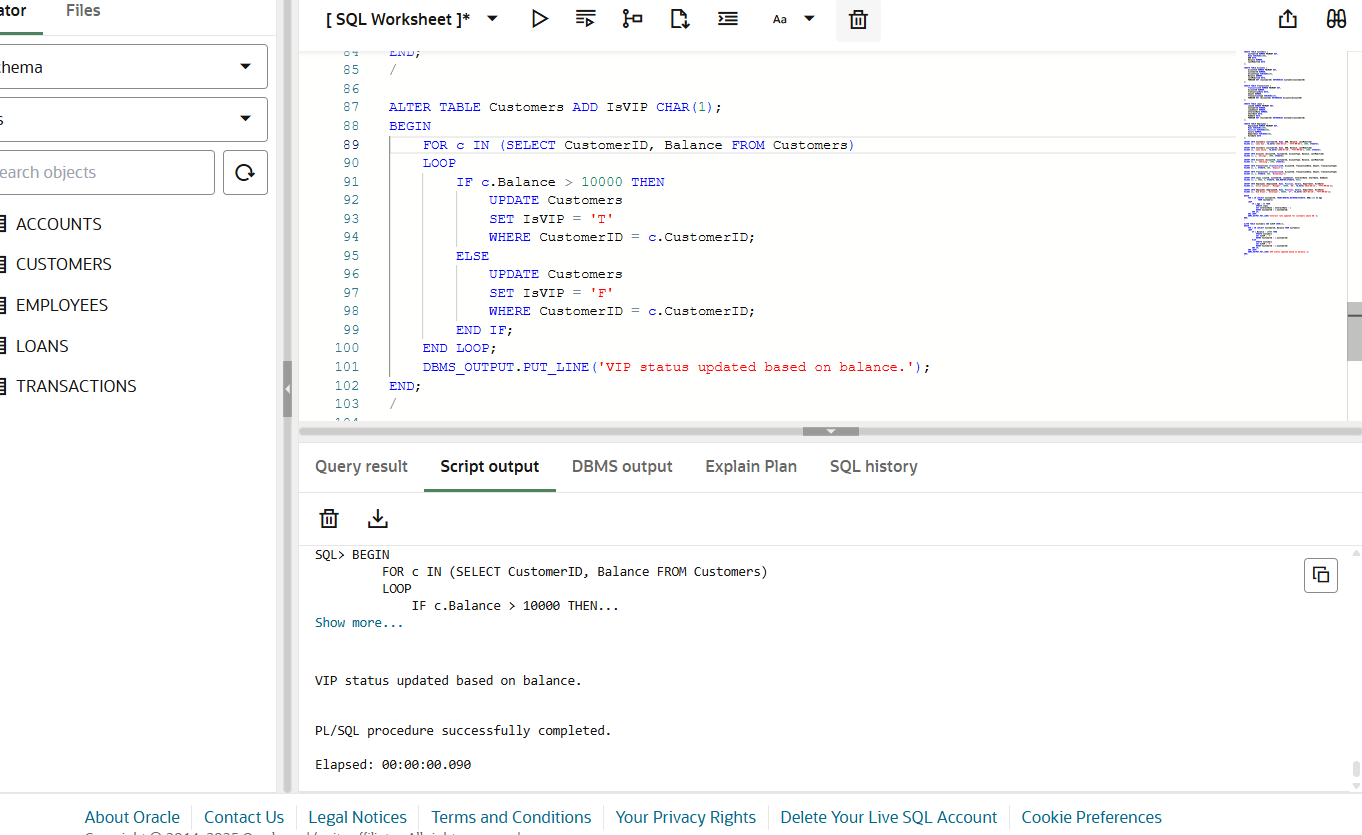
    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE('VIP status updated based on balance.');

END;

/

**Output:**



**Scenario 3:**

**Code:**

UPDATE Loans

SET EndDate = SYSDATE + 10

WHERE LoanID = 1;

COMMIT;

BEGIN

    FOR l IN (

        SELECT LoanID, CustomerID, EndDate

        FROM Loans

        WHERE EndDate <= SYSDATE + 30

    )

    LOOP

        DECLARE

            custName VARCHAR2(100);

        BEGIN

            SELECT Name INTO custName FROM Customers WHERE CustomerID = l.CustomerID;

            DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || l.LoanID || ' for customer ' || custName || ' is due on ' || TO\_CHAR(l.EndDate, 'YYYY-MM-DD'));

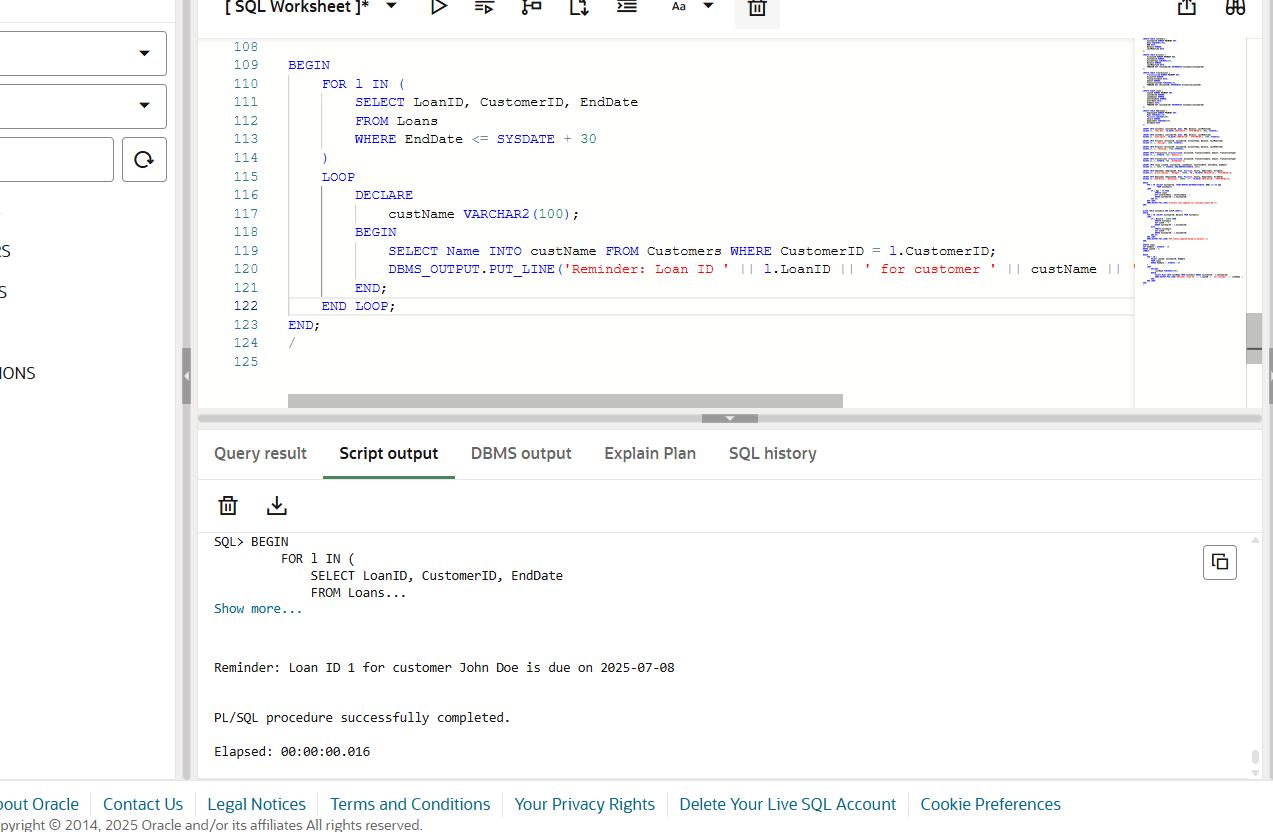
        END;

    END LOOP;

END;

/

**Output:**



**Exercise 2: Error Handling**

**Scenario 1:**

**Code:**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

    p\_from\_acc\_id IN NUMBER,

    p\_to\_acc\_id IN NUMBER,

    p\_amount IN NUMBER

) IS

    v\_balance NUMBER;

BEGIN

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

    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\_acc\_id;

    UPDATE Accounts

    SET Balance = Balance + p\_amount

    WHERE AccountID = p\_to\_acc\_id;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ₹' || p\_amount || ' transferred from Account ' || p\_from\_acc\_id || ' to ' || p\_to\_acc\_id);

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        ROLLBACK;

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

    WHEN OTHERS THEN

        ROLLBACK;

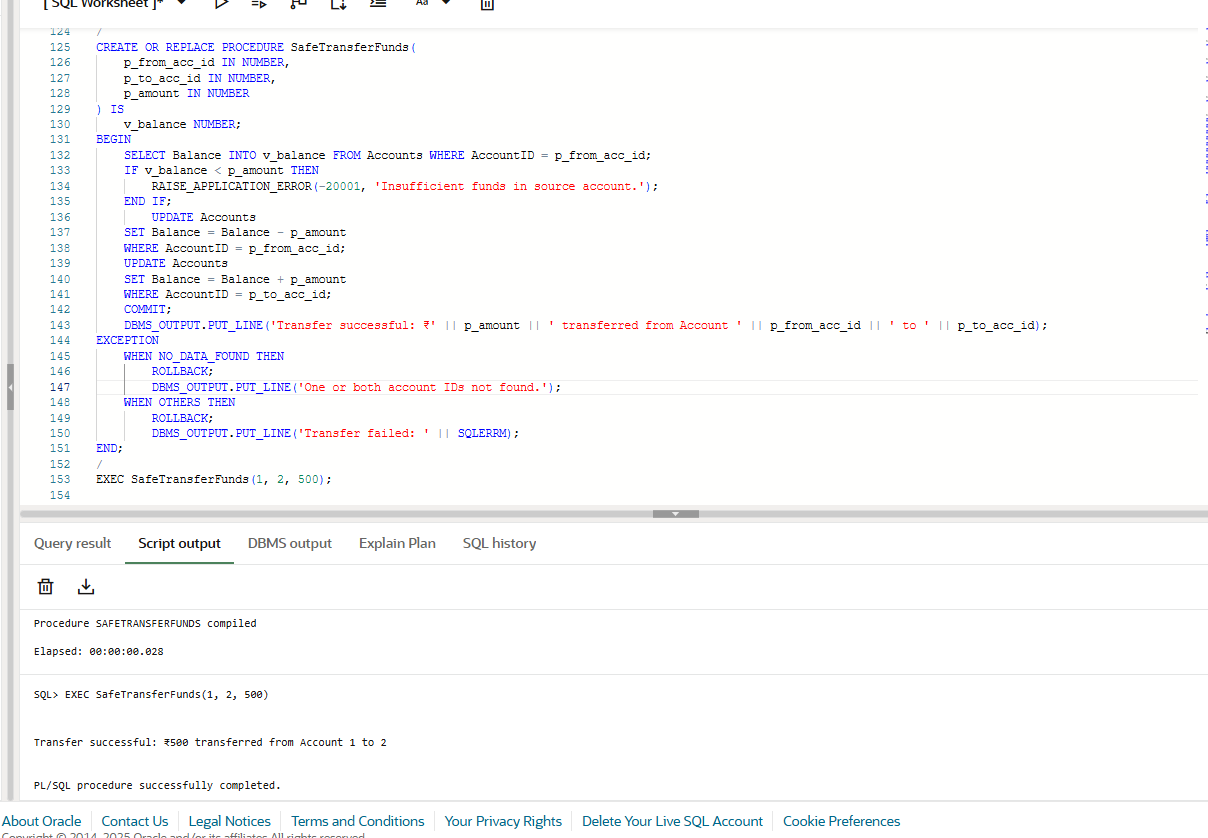
        DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;

/

EXEC SafeTransferFunds(1, 2, 500);

**Output:**



**Scenario 2:**

**Code:**

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 for Employee ID: ' || p\_emp\_id);

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

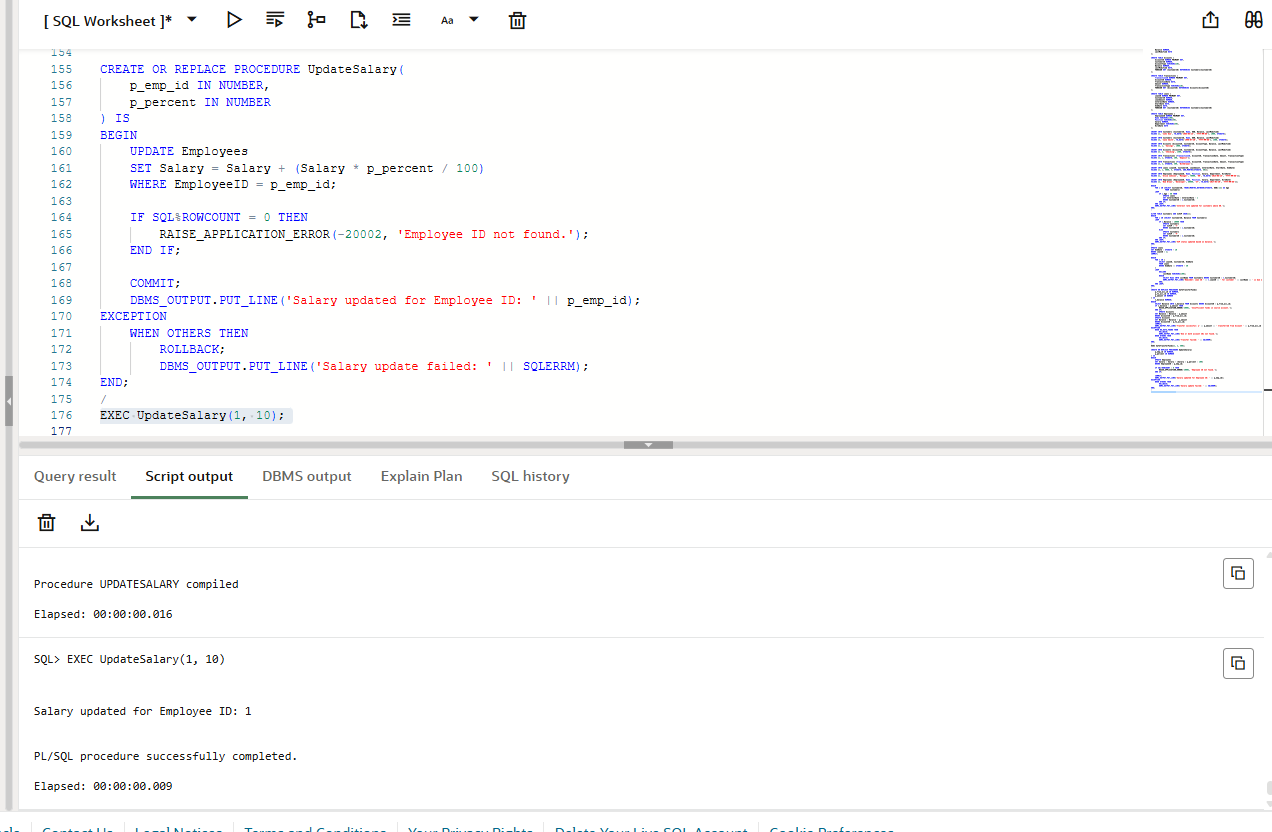
        DBMS\_OUTPUT.PUT\_LINE('Salary update failed: ' || SQLERRM);

END;

/

EXEC UpdateSalary(1, 10);

**Output:**



**Scenario 3:**

**Code:**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

    p\_cust\_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\_cust\_id, p\_name, p\_dob, p\_balance, SYSDATE);

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('New customer added: ' || p\_name);

EXCEPTION

    WHEN DUP\_VAL\_ON\_INDEX THEN

        DBMS\_OUTPUT.PUT\_LINE('Insertion failed: Customer ID ' || p\_cust\_id || ' already exists.');

    WHEN OTHERS THEN

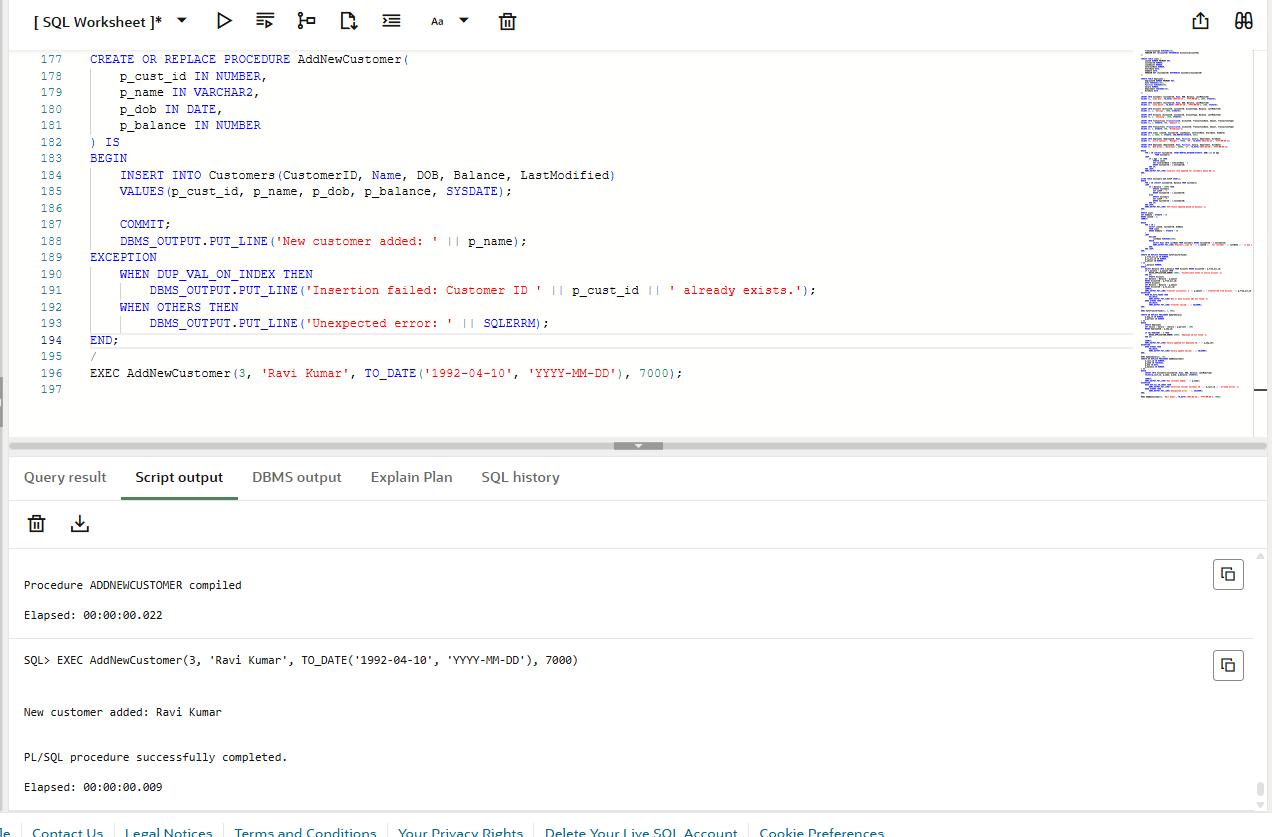
        DBMS\_OUTPUT.PUT\_LINE('Unexpected error: ' || SQLERRM);

END;

/

EXEC AddNewCustomer(3, 'Ravi Kumar', TO\_DATE('1992-04-10', 'YYYY-MM-DD'), 7000);

**Output:**



**Exercise 3: Stored Procedures**

**Scenario 1:**

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

    FOR acc IN (

        SELECT AccountID, Balance

        FROM Accounts

        WHERE AccountType = 'Savings'

    ) LOOP        UPDATE Accounts

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

            LastModified = SYSDATE

        WHERE AccountID = acc.AccountID;

        DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID ' || acc.AccountID ||

                             ': New Balance = ₹' || TO\_CHAR(acc.Balance \* 1.01, '999999.99'));

    END LOOP;

    COMMIT;

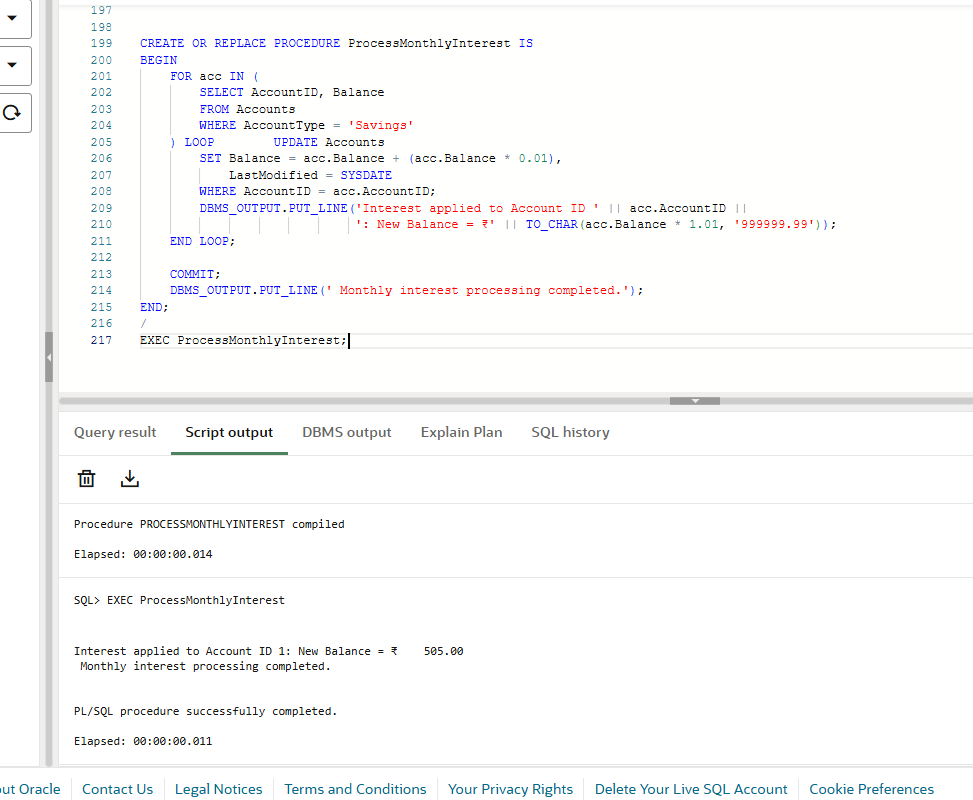
    DBMS\_OUTPUT.PUT\_LINE(' Monthly interest processing completed.');

END;

/

EXEC ProcessMonthlyInterest;

**Output:**



**Scenario 2:**

**Code:**

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

        DBMS\_OUTPUT.PUT\_LINE(' Bonus applied to ' || SQL%ROWCOUNT ||

                             ' employee(s) in ' || p\_department || ' department.');

    ELSE

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

    END IF;

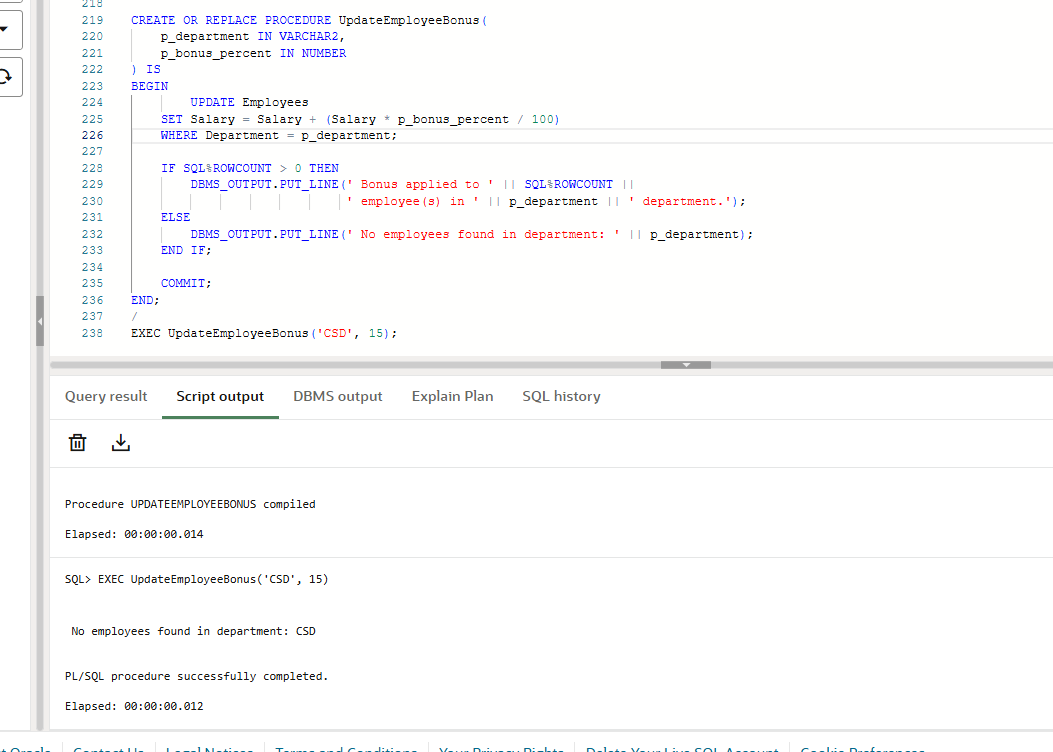
    COMMIT;

END;

/

EXEC UpdateEmployeeBonus('CSD', 15);

**Output:**



**Scenario 3:**

**Code:**

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;

    IF v\_balance < p\_amount THEN

        DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient funds in Account ID ' || p\_from\_account);

        RETURN;

    END IF;

    UPDATE Accounts

    SET Balance = Balance - p\_amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_from\_account;

    UPDATE Accounts

    SET Balance = Balance + p\_amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_to\_account;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('$' || p\_amount || ' transferred from Account ID ' || p\_from\_account ||

                         ' to Account ID ' || p\_to\_account);

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

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

    WHEN OTHERS THEN

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

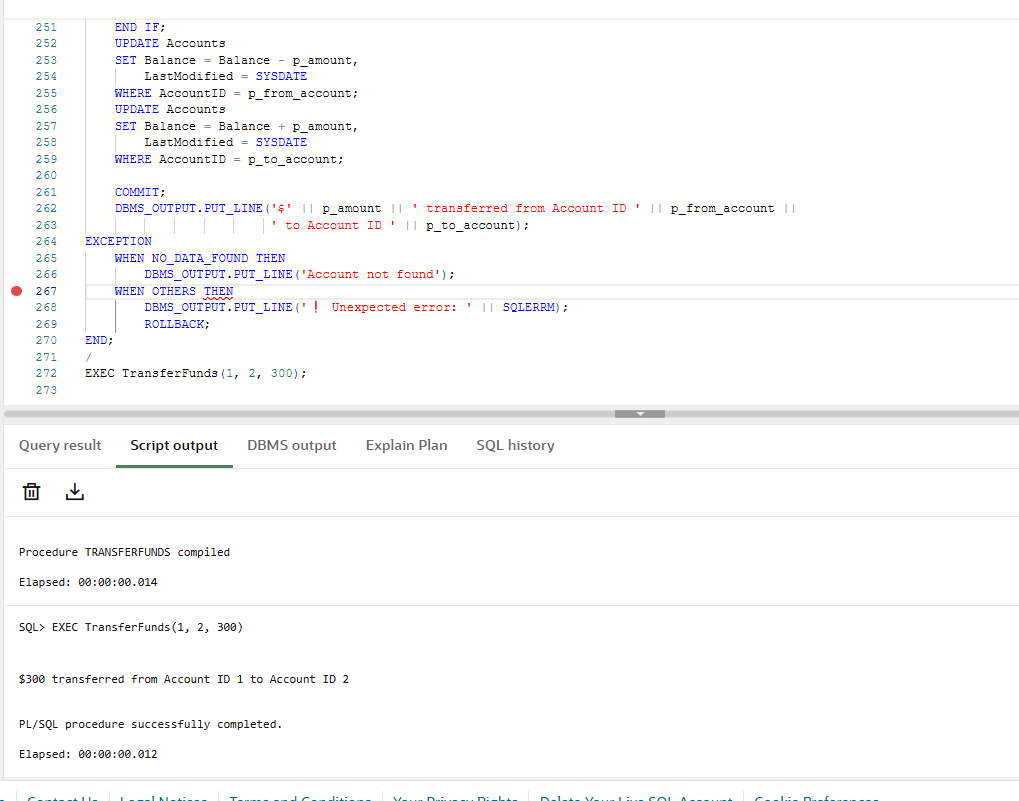
        ROLLBACK;

END;

/

EXEC TransferFunds(1, 2, 300);

**Output:**



**Exercise 4: Functions**

**Scenario 1:**

**Code:**

CREATE OR REPLACE FUNCTION CalculateAge(

    p\_dob IN DATE

) RETURN NUMBER IS

    v\_age NUMBER;

BEGIN

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

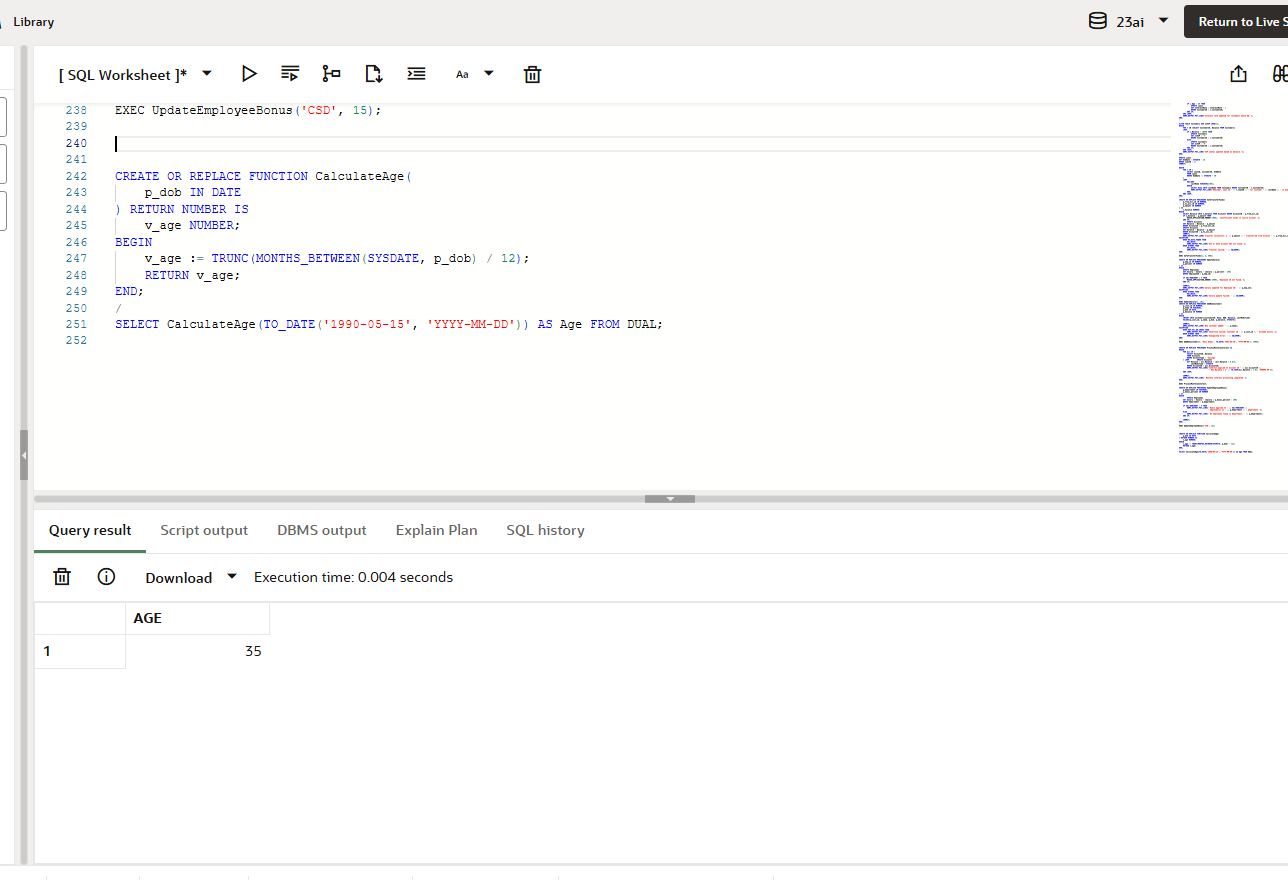
    RETURN v\_age;

END;

/

SELECT CalculateAge(TO\_DATE('1990-05-15', 'YYYY-MM-DD')) AS Age FROM DUAL;

**Output:**



**Scenario 2:**

**Code:**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

    p\_loan\_amount IN NUMBER,

    p\_interest\_rate IN NUMBER,

    p\_years IN NUMBER

) RETURN NUMBER IS

    v\_monthly\_rate NUMBER;

    v\_months NUMBER;

    v\_emi NUMBER;

BEGIN

    v\_monthly\_rate := p\_interest\_rate / 12 / 100;

    v\_months := p\_years \* 12;

    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;

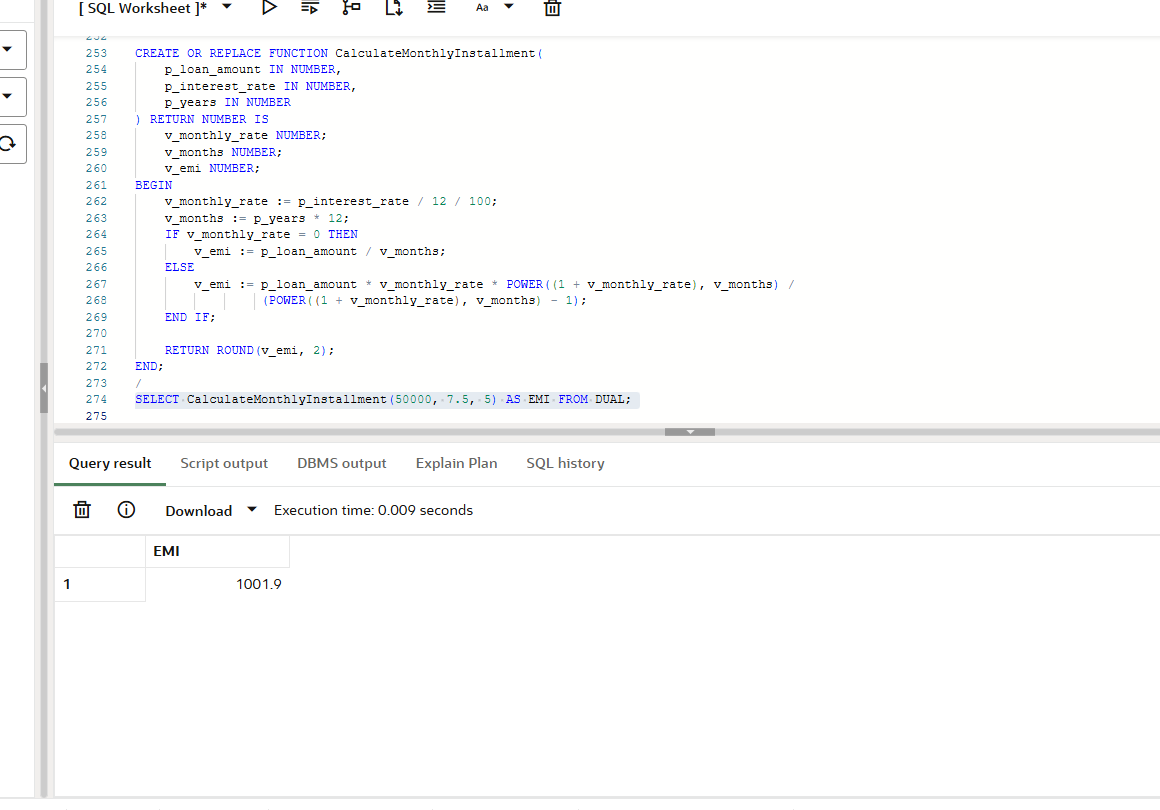
    RETURN ROUND(v\_emi, 2);

END;

/

SELECT CalculateMonthlyInstallment(50000, 7.5, 5) AS EMI FROM DUAL;

**Output:**



**Scenario 3:**

**Code:**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

    p\_account\_id IN NUMBER,

    p\_required\_amount IN NUMBER

) RETURN BOOLEAN IS

    v\_balance NUMBER;

BEGIN

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

    IF v\_balance >= p\_required\_amount THEN

        RETURN TRUE;

    ELSE

        RETURN FALSE;

    END IF;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RETURN FALSE;

END;

/

DECLARE

    v\_ok BOOLEAN;

BEGIN

    v\_ok := HasSufficientBalance(1, 1000);

    IF v\_ok THEN

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

    ELSE

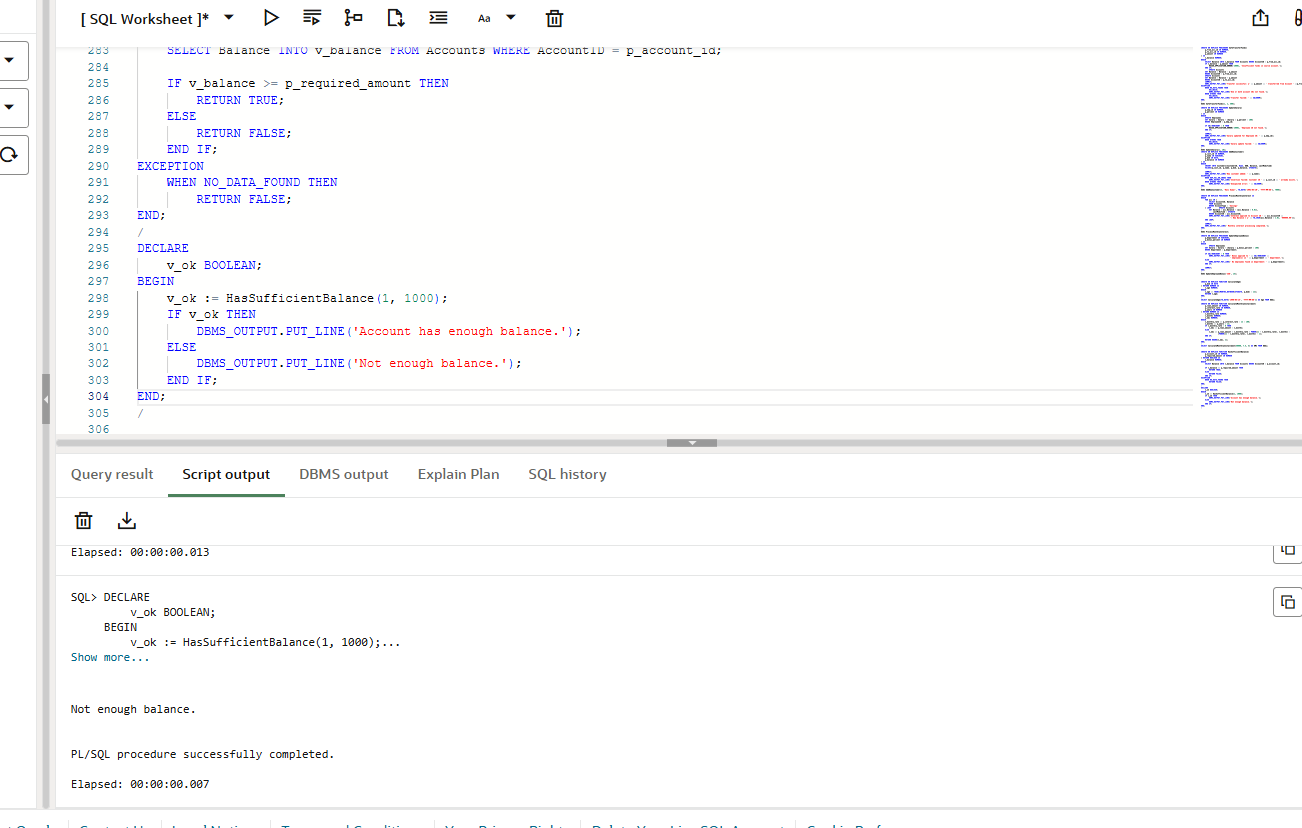
        DBMS\_OUTPUT.PUT\_LINE('Not enough balance.');

    END IF;

END;

/

**Output:**



**Exercise 5: Triggers**

**Scenario 1:**

**Code:**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

    :NEW.LastModified := SYSDATE;

END;

/

**Output:**



**Scenario 2:**

**Code:**

CREATE TABLE AuditLog (

    LogID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    TransactionID NUMBER,

    AccountID NUMBER,

    Amount NUMBER,

    TransactionType VARCHAR2(20),

    ActionTime TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

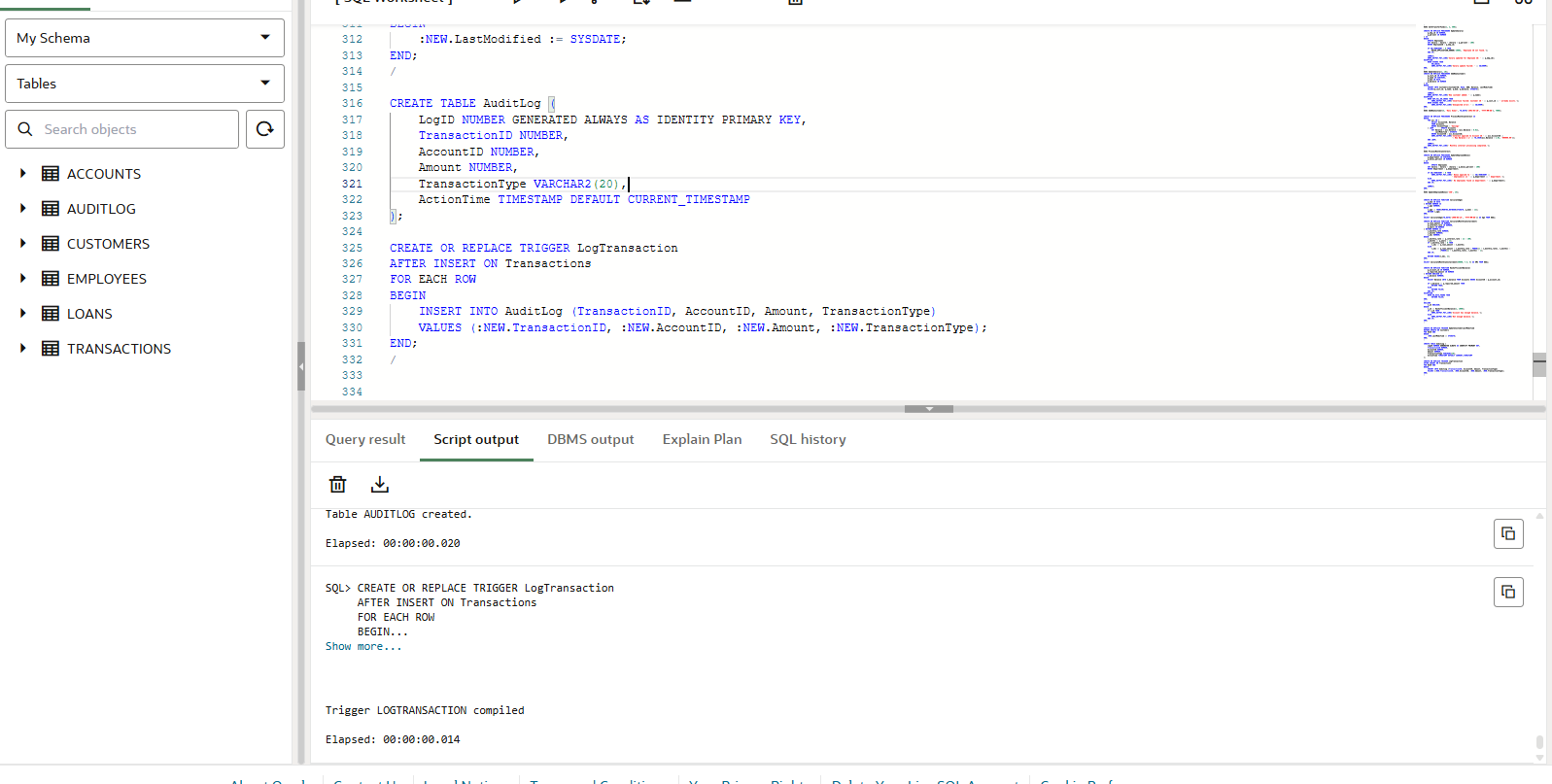
    INSERT INTO AuditLog (TransactionID, AccountID, Amount, TransactionType)

    VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.Amount, :NEW.TransactionType);

END;

/

**Output:**



**Scenario 3:**

**Code:**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

    v\_balance NUMBER;

BEGIN

    -- Check current balance

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

    IF :NEW.TransactionType = 'Withdrawal' THEN

        IF :NEW.Amount > v\_balance THEN

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

        END IF;

    ELSIF :NEW.TransactionType = 'Deposit' THEN

        IF :NEW.Amount <= 0 THEN

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

        END IF;

    ELSE

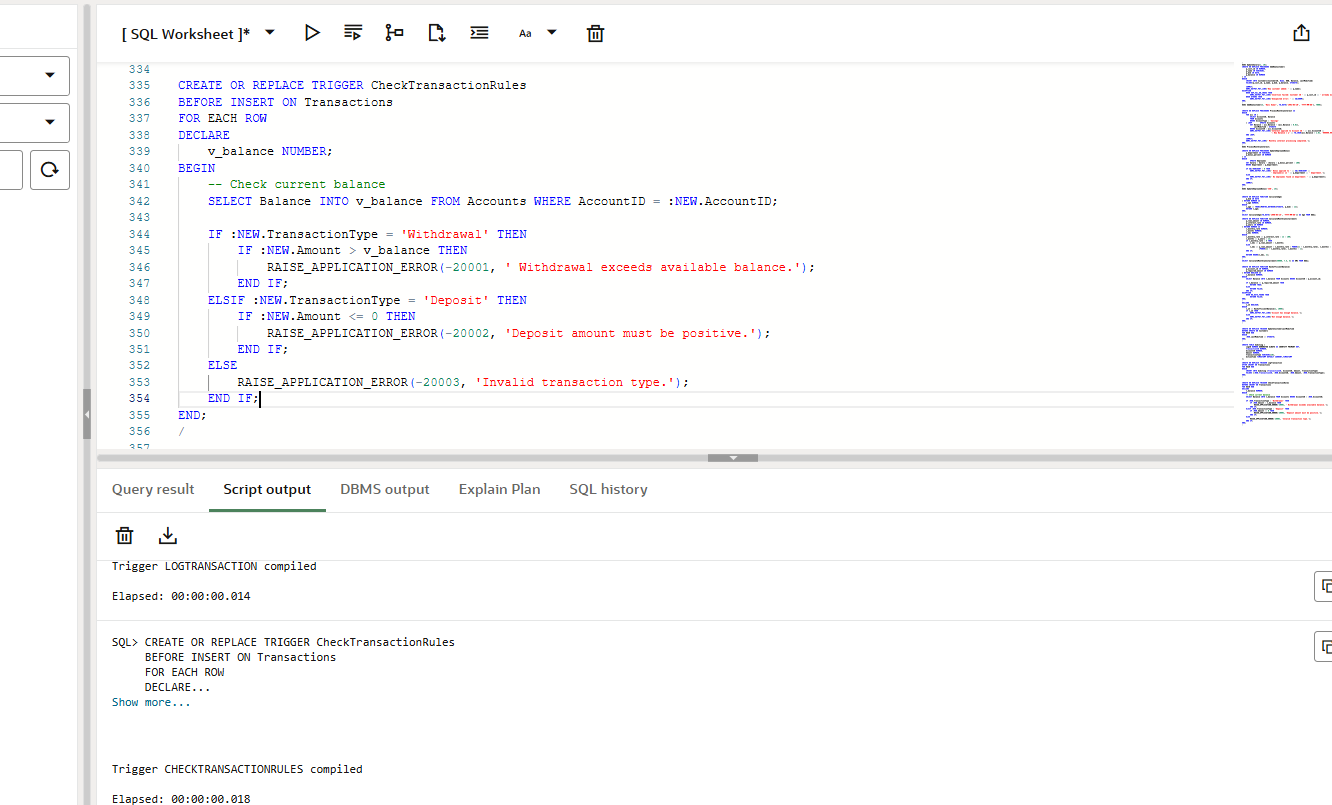
        RAISE\_APPLICATION\_ERROR(-20003, 'Invalid transaction type.');

    END IF;

END;

/

**Output:**



**Exercise 6: Cursors**

**Scenario 1:**

**Code:**

DECLARE

    CURSOR trans\_cursor IS

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

        FROM Transactions t

        JOIN Accounts a ON t.AccountID = a.AccountID

        JOIN Customers c ON a.CustomerID = c.CustomerID

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

    v\_trans trans\_cursor%ROWTYPE;

BEGIN

    OPEN trans\_cursor;

    LOOP

        FETCH trans\_cursor INTO v\_trans;

        EXIT WHEN trans\_cursor%NOTFOUND;

        DBMS\_OUTPUT.PUT\_LINE('📄 Statement for: ' || v\_trans.Name);

        DBMS\_OUTPUT.PUT\_LINE('   Transaction ID: ' || v\_trans.TransactionID);

        DBMS\_OUTPUT.PUT\_LINE('   Type: ' || v\_trans.TransactionType || ' | Amount: ₹' || v\_trans.Amount ||

                             ' | Date: ' || TO\_CHAR(v\_trans.TransactionDate, 'YYYY-MM-DD'));

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

    END LOOP;

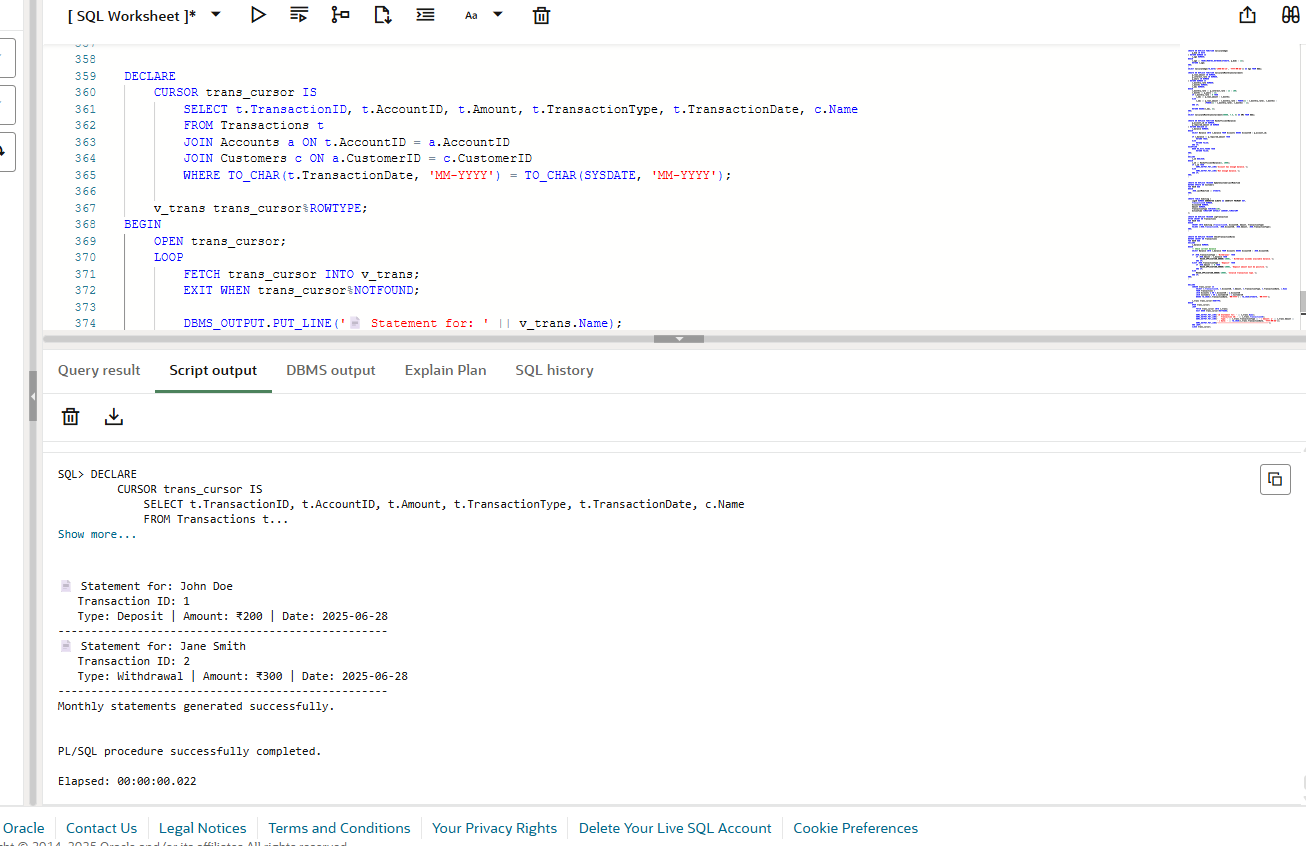
    CLOSE trans\_cursor;

    DBMS\_OUTPUT.PUT\_LINE('Monthly statements generated successfully.');

END;

/

**Output:**



Scenario 2:

**Code:**

DECLARE

    CURSOR account\_cursor IS

        SELECT AccountID, Balance FROM Accounts;

    v\_account account\_cursor%ROWTYPE;

    v\_fee NUMBER := 100;

BEGIN

    OPEN account\_cursor;

    LOOP

        FETCH account\_cursor INTO v\_account;

        EXIT WHEN account\_cursor%NOTFOUND;

        UPDATE Accounts

        SET Balance = Balance - v\_fee,

            LastModified = SYSDATE

        WHERE AccountID = v\_account.AccountID;

        DBMS\_OUTPUT.PUT\_LINE('$' || v\_fee || ' annual fee deducted from Account ID: ' || v\_account.AccountID);

    END LOOP;

    CLOSE account\_cursor;

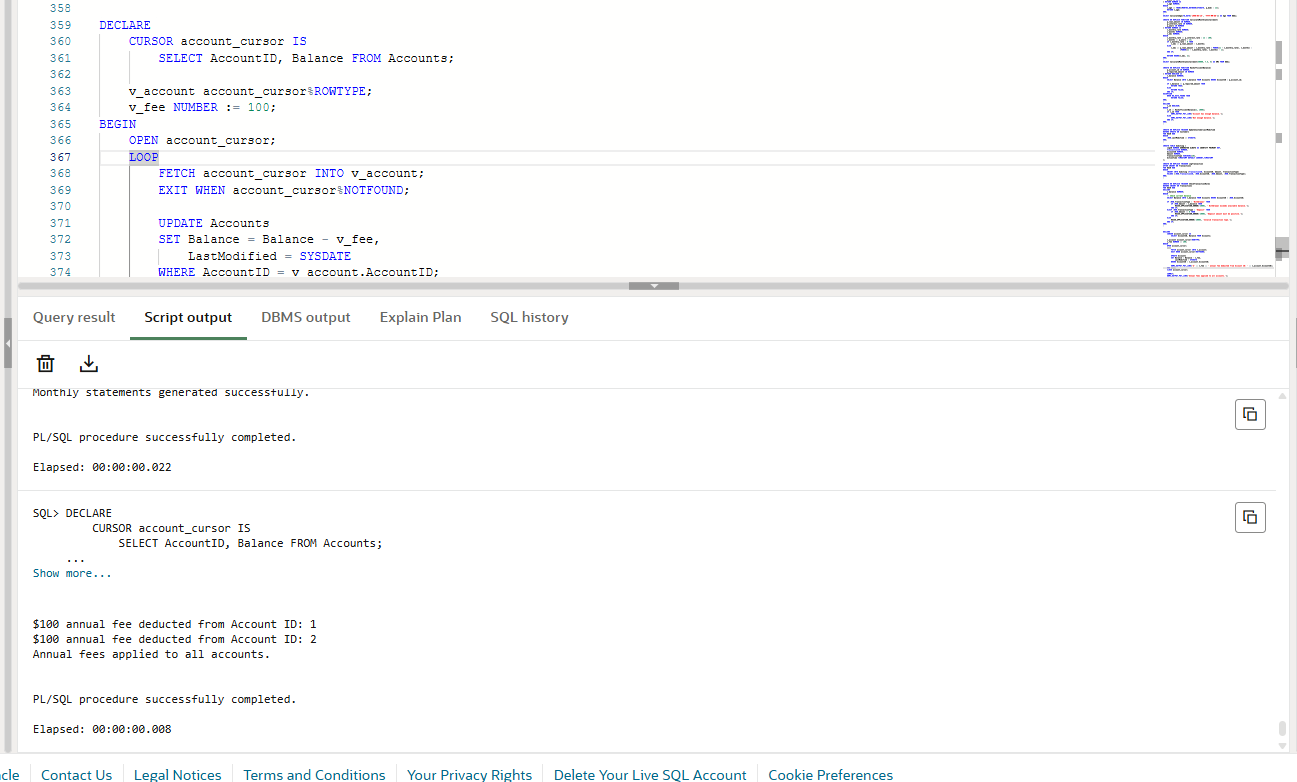
    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Annual fees applied to all accounts.');

END;

/

**Output:**



**Scenario 3:**

**Code:**

DECLARE

    CURSOR loan\_cursor IS

        SELECT LoanID, InterestRate, LoanAmount FROM Loans;

    v\_loan loan\_cursor%ROWTYPE;

    v\_new\_rate NUMBER;

BEGIN

    OPEN loan\_cursor;

    LOOP

        FETCH loan\_cursor INTO v\_loan;

        EXIT WHEN loan\_cursor%NOTFOUND;

        IF v\_loan.LoanAmount < 10000 THEN

            v\_new\_rate := v\_loan.InterestRate + 0.5;

        ELSE

            v\_new\_rate := v\_loan.InterestRate + 1;

        END IF;

        UPDATE Loans

        SET InterestRate = v\_new\_rate

        WHERE LoanID = v\_loan.LoanID;

        DBMS\_OUTPUT.PUT\_LINE('Loan ID ' || v\_loan.LoanID || ' interest updated to ' || v\_new\_rate || '%');

    END LOOP;

    CLOSE loan\_cursor;

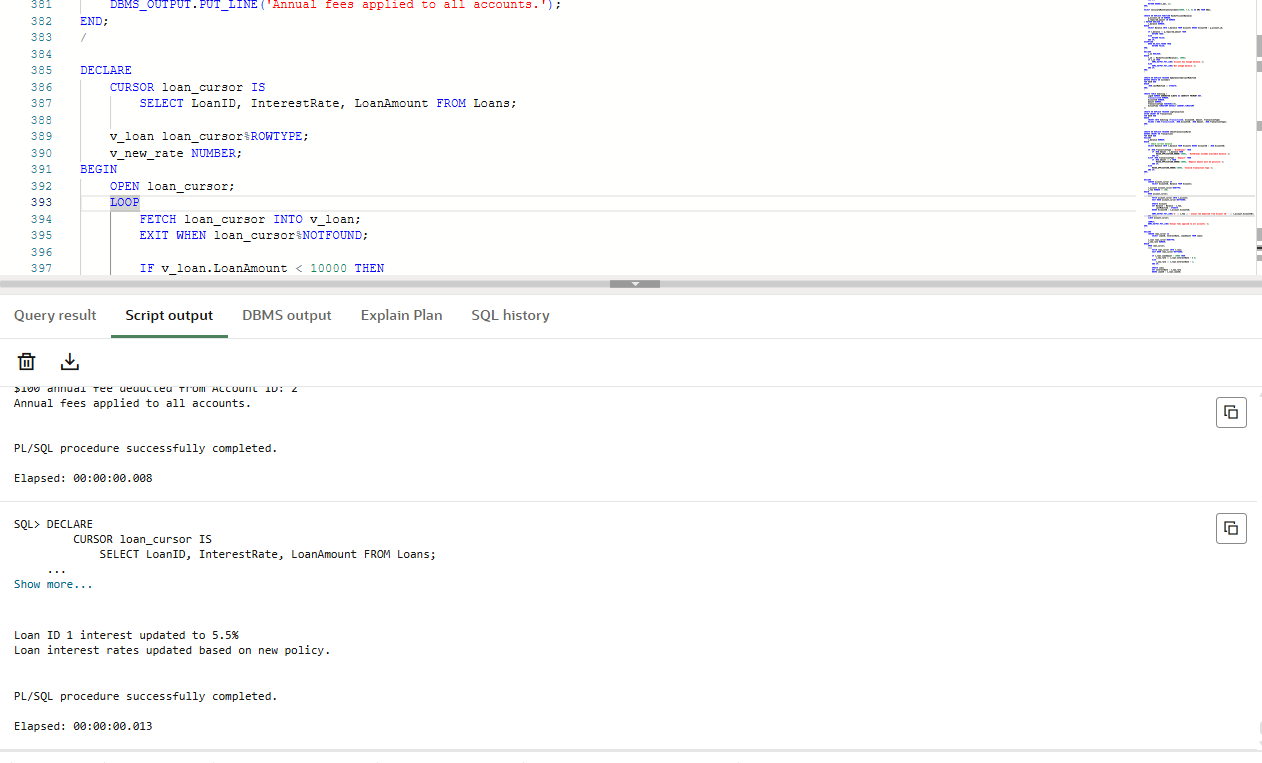
    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Loan interest rates updated based on new policy.');

END;

/

**Output:**



**Exercise 7: Packages**

**Scenario 1:**

**Code:**

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\_dob DATE);

    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(CustomerID, Name, DOB, Balance, LastModified)

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

        DBMS\_OUTPUT.PUT\_LINE('Customer added: ' || p\_name);

    EXCEPTION

        WHEN DUP\_VAL\_ON\_INDEX THEN

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

    END;

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

    BEGIN

        UPDATE Customers

        SET Name = p\_name, DOB = p\_dob, LastModified = SYSDATE

        WHERE CustomerID = p\_id;

        IF SQL%ROWCOUNT = 0 THEN

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

        ELSE

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

        END IF;

    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

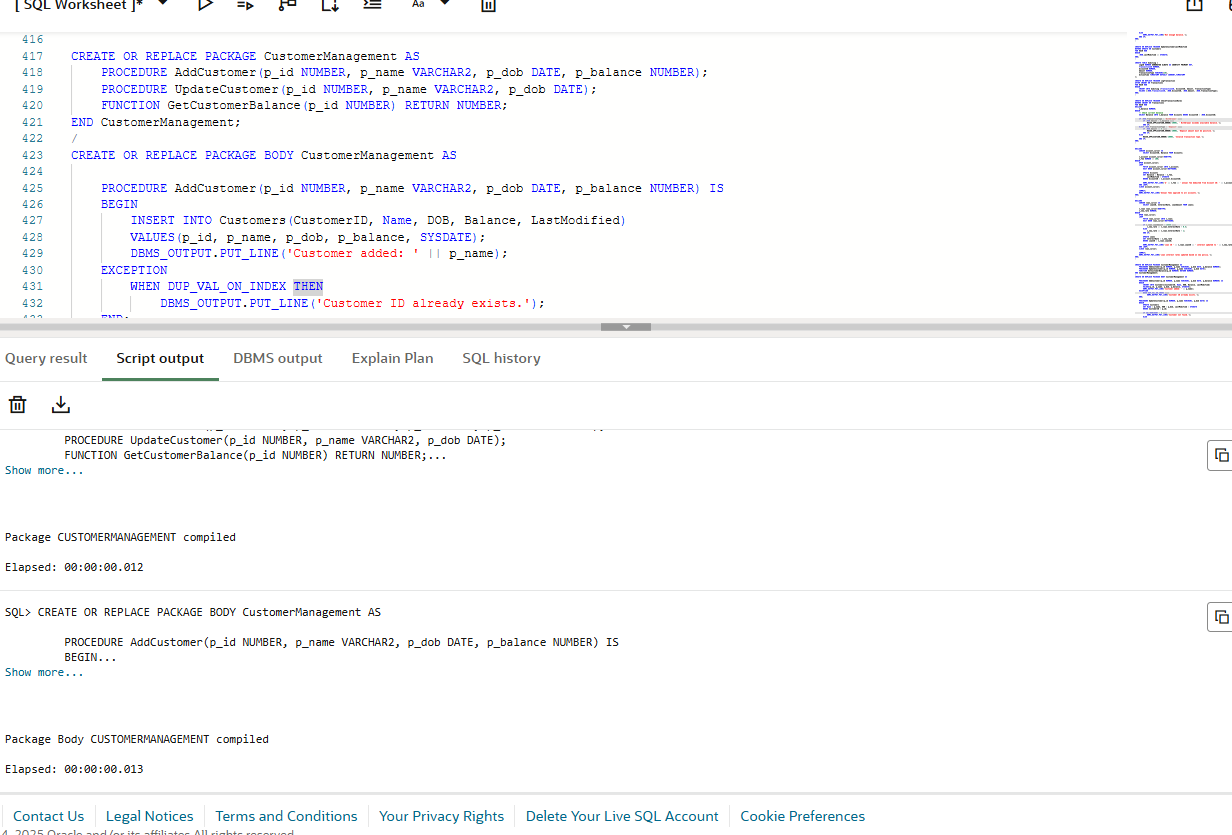
            RETURN NULL;

    END;

END CustomerManagement;

/

**Output:**



**Scenario 2:**

**Code:**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

    PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_hiredate DATE);

    PROCEDURE UpdateEmployee(p\_id NUMBER, p\_position VARCHAR2, p\_salary NUMBER);

    FUNCTION CalculateAnnualSalary(p\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

    PROCEDURE HireEmployee(p\_id NUMBER, p\_name VARCHAR2, p\_position VARCHAR2, p\_salary NUMBER, p\_dept VARCHAR2, p\_hiredate DATE) IS

    BEGIN

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

        VALUES(p\_id, p\_name, p\_position, p\_salary, p\_dept, p\_hiredate);

        DBMS\_OUTPUT.PUT\_LINE('Employee hired: ' || p\_name);

    EXCEPTION

        WHEN DUP\_VAL\_ON\_INDEX THEN

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

    END;

    PROCEDURE UpdateEmployee(p\_id NUMBER, p\_position VARCHAR2, p\_salary NUMBER) IS

    BEGIN

        UPDATE Employees

        SET Position = p\_position, Salary = p\_salary

        WHERE EmployeeID = p\_id;

        IF SQL%ROWCOUNT = 0 THEN

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

        ELSE

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

        END IF;

    END;

    FUNCTION CalculateAnnualSalary(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

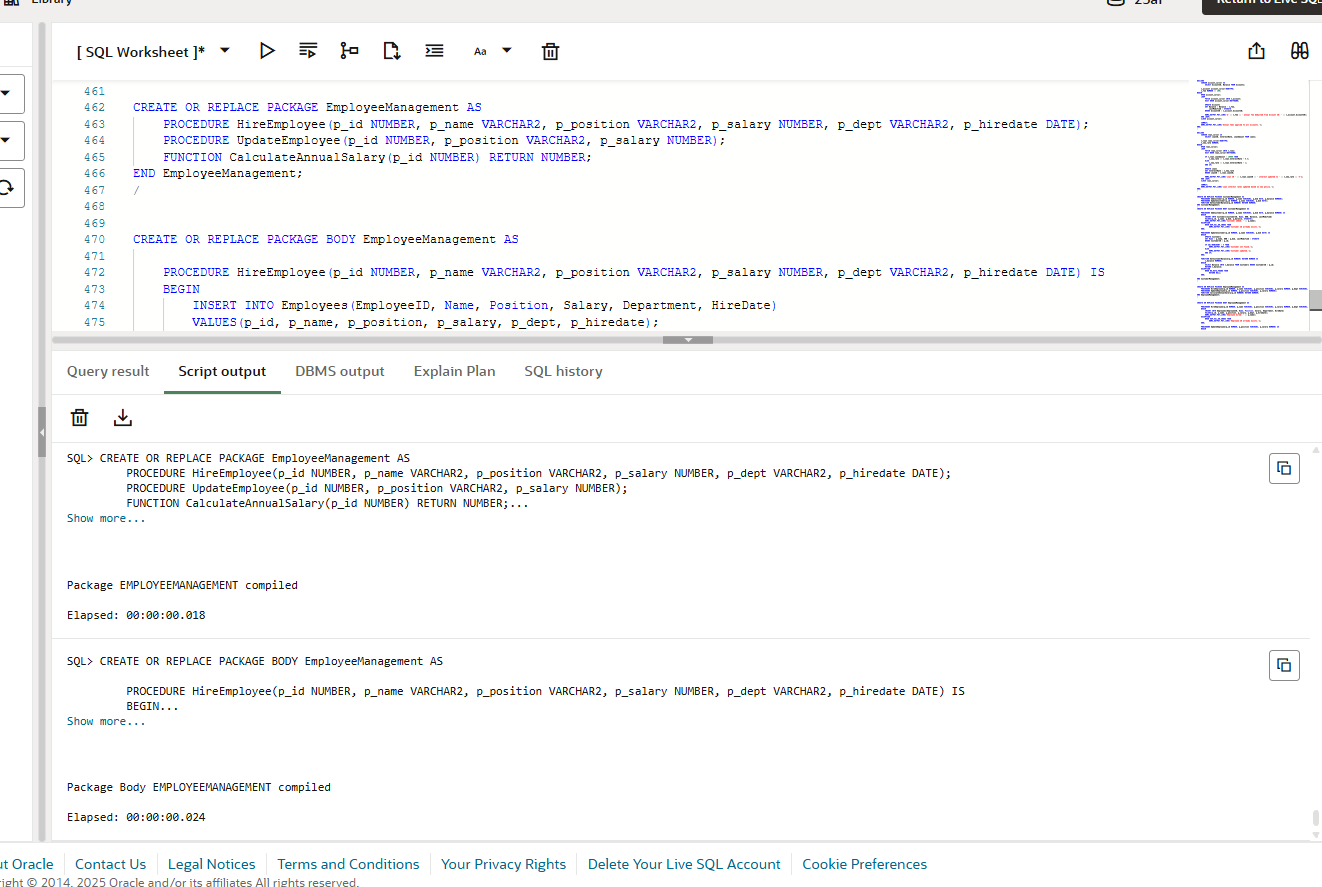
            RETURN NULL;

    END;

END EmployeeManagement;

/

**Output:**



**Scenario 3:**

**Code:**  
CREATE OR REPLACE PACKAGE AccountOperations AS

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

    PROCEDURE CloseAccount(p\_acc\_id NUMBER);

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

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

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

    BEGIN

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

        VALUES(p\_acc\_id, p\_cust\_id, p\_type, p\_balance, SYSDATE);

        DBMS\_OUTPUT.PUT\_LINE('Account opened for Customer ID: ' || p\_cust\_id);

    EXCEPTION

        WHEN DUP\_VAL\_ON\_INDEX THEN

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

        WHEN OTHERS THEN

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

    END;

    PROCEDURE CloseAccount(p\_acc\_id NUMBER) IS

    BEGIN

        DELETE FROM Accounts WHERE AccountID = p\_acc\_id;

        IF SQL%ROWCOUNT = 0 THEN

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

        ELSE

            DBMS\_OUTPUT.PUT\_LINE('Account closed: ' || p\_acc\_id);

        END IF;

    END;

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

        v\_total NUMBER;

    BEGIN

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

        RETURN NVL(v\_total, 0);

    EXCEPTION

        WHEN NO\_DATA\_FOUND THEN

            RETURN 0;

    END;

END AccountOperations;

/

**Output:**

