**WEEK 2 HANDS ON EXERCISE – PL/SQL PROGRAMMING**

**Exercise 1: Control Structures**

**PROBLEM STATEMENT :**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

* + **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.

**CODE :**

DECLARE

    CURSOR cur\_senior\_customers IS

        SELECT c.CustomerID, l.LoanID, l.InterestRate

        FROM Customers c

        JOIN Loans l ON c.CustomerID = l.CustomerID

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

BEGIN

    FOR rec IN cur\_senior\_customers LOOP

        UPDATE Loans

        SET InterestRate = InterestRate - 1

        WHERE LoanID = rec.LoanID;

        DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount to CustomerID ' || rec.CustomerID ||

                             ', LoanID ' || rec.LoanID ||

                             '. New InterestRate = ' || (rec.InterestRate - 1));

    END LOOP;

    COMMIT;

END;

/

SELECT

    c.CustomerID,

    c.Name,

    TRUNC(MONTHS\_BETWEEN(SYSDATE, c.DOB) / 12) AS Age,

    l.LoanID,

    l.LoanAmount,

    l.InterestRate,

    l.StartDate,

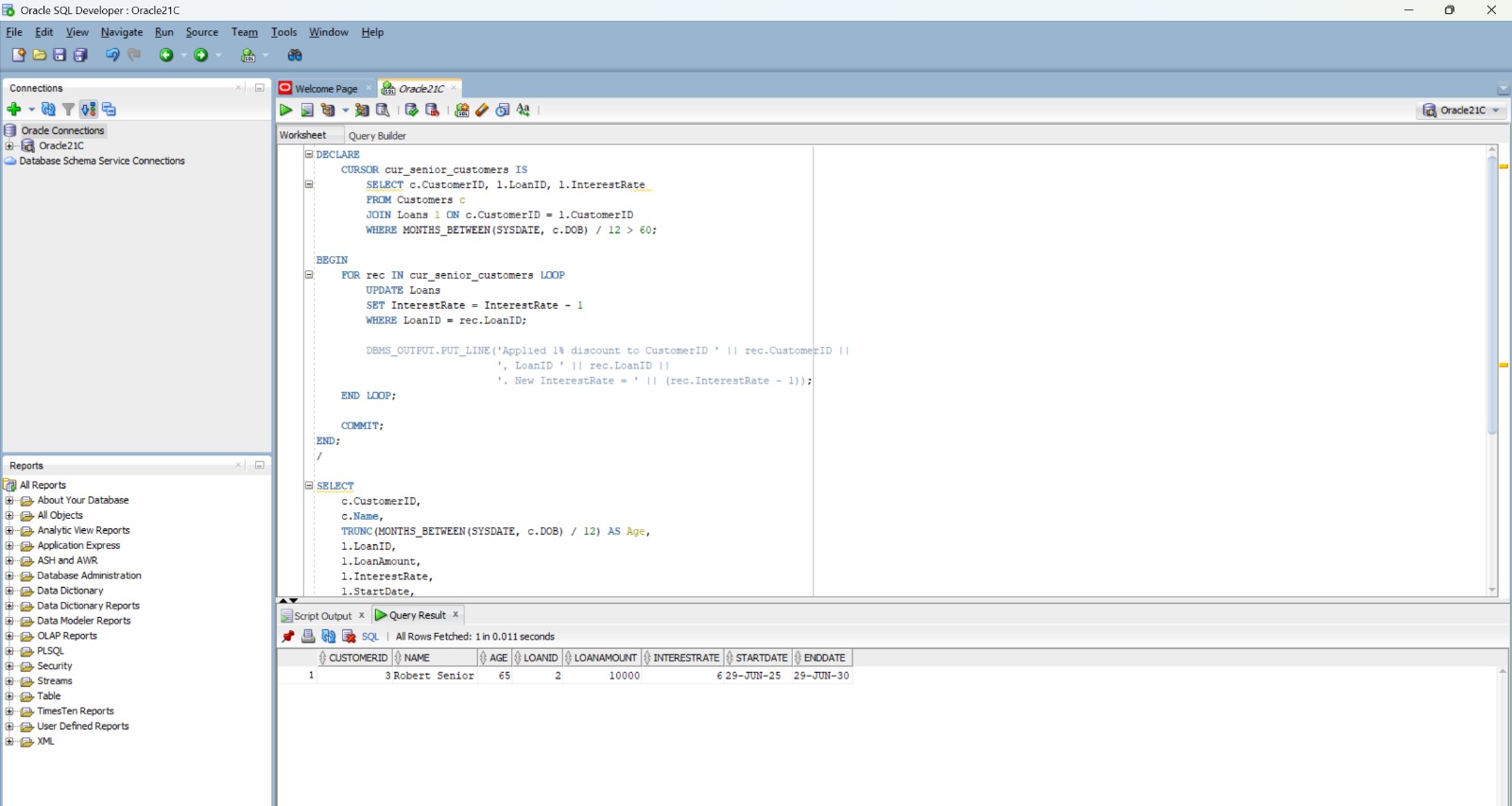
    l.EndDate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE TRUNC(MONTHS\_BETWEEN(SYSDATE, c.DOB) / 12) > 60;

**OUTPUT :**



**PROBLEM STATEMENT :**

**Scenario 2**:A customer can be promoted to VIP status based on their balance.

* + 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.

**CODE :**

ALTER TABLE Customers ADD IsVIP CHAR(1) DEFAULT 'N';

BEGIN

    FOR cust IN (

        SELECT CustomerID, Balance

        FROM Customers

    ) LOOP

        IF cust.Balance > 10000 THEN

            UPDATE Customers

            SET IsVIP = 'Y'

            WHERE CustomerID = cust.CustomerID;

        ELSE

            UPDATE Customers

            SET IsVIP = 'N'

            WHERE CustomerID = cust.CustomerID;

        END IF;

    END LOOP;

    COMMIT;

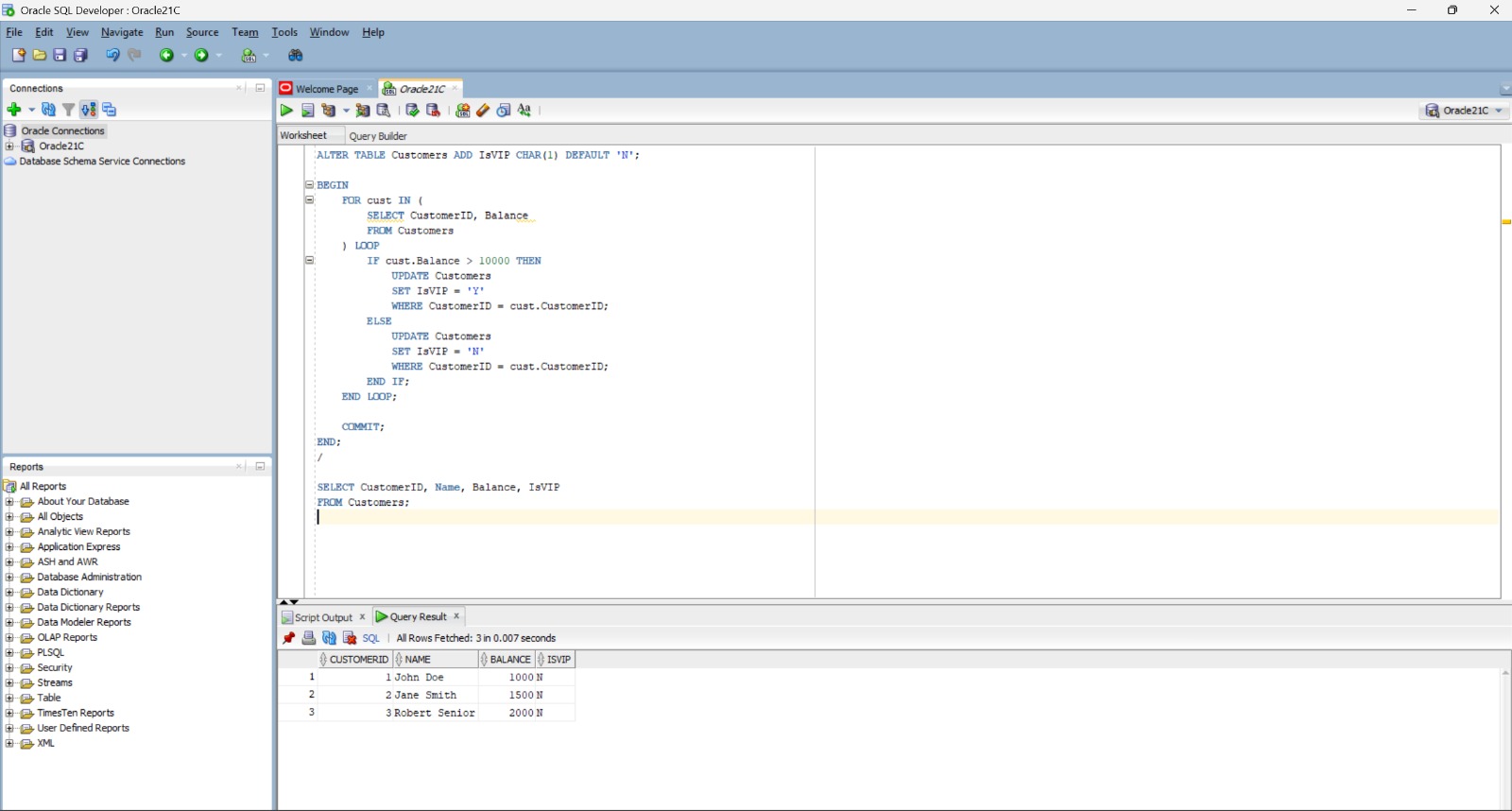
END;

/

SELECT CustomerID, Name, Balance, IsVIP

FROM Customers;

**OUTPUT:**



**PROBLEM STATEMENT :**

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

* + Question: Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**CODE :**

SET SERVEROUTPUT ON;

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

VALUES (4, 1, 8000, 4.5, SYSDATE - 335, SYSDATE + 15);

COMMIT;

BEGIN

    FOR loan\_rec IN (

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

        FROM Customers c

        JOIN Loans l ON c.CustomerID = l.CustomerID

        WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

    ) LOOP

        DBMS\_OUTPUT.PUT\_LINE(

            'Reminder: Loan ID ' || loan\_rec.LoanID || ' for customer ' ||

            loan\_rec.Name || ' (Customer ID: ' || loan\_rec.CustomerID ||

            ') is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-MON-YYYY')

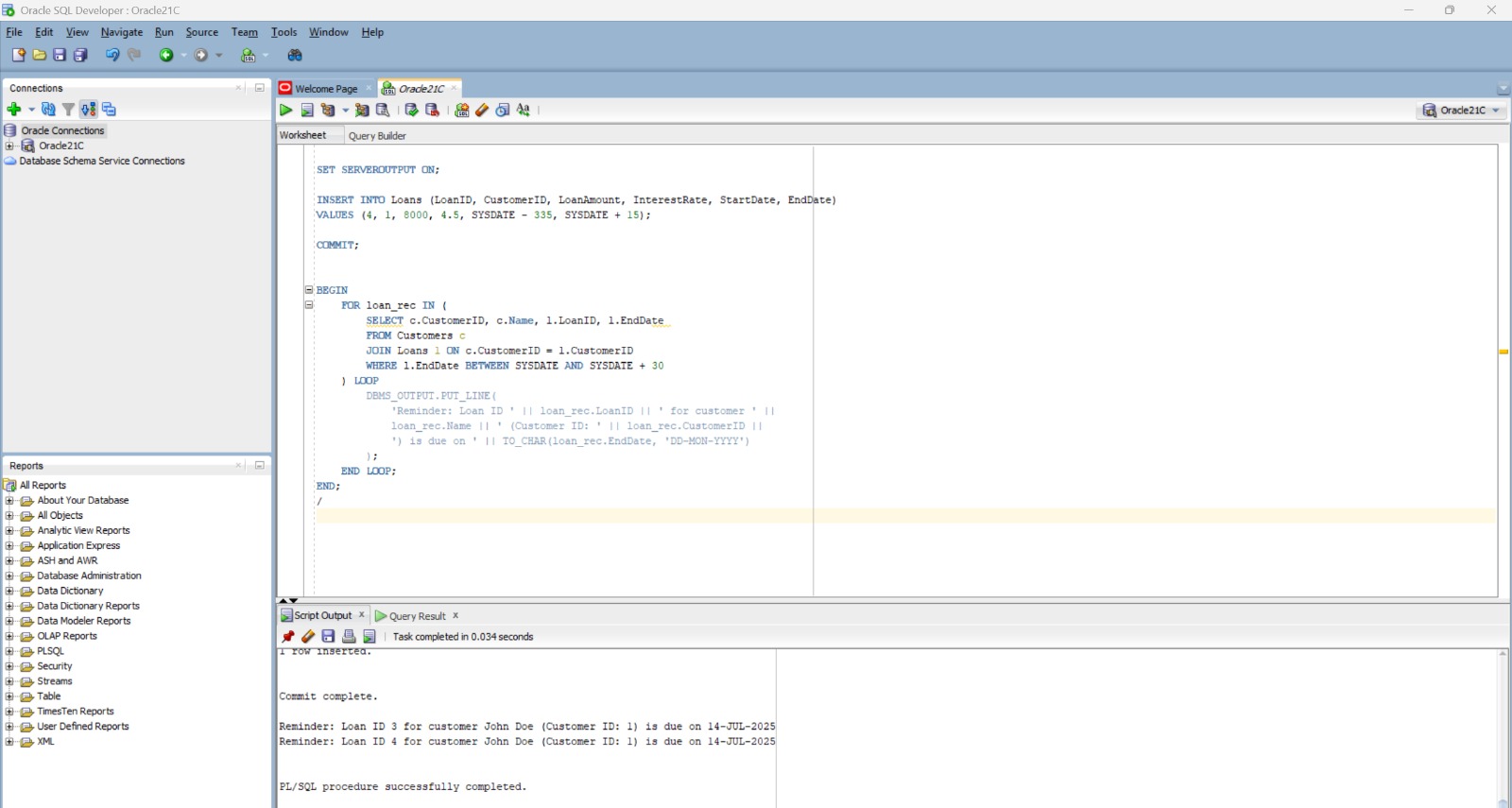
        );

    END LOOP;

END;

/

**OUTPUT :**



**Exercise 3: Stored Procedure**

**PROBLEM STATEMENT :**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

* + 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**.**

**CODE :**

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

VALUES (5, 1, 'Savings', 5000, SYSDATE);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

    FOR acc IN (

        SELECT AccountID, Balance

        FROM Accounts

        WHERE AccountType = 'Savings'

    ) LOOP

        UPDATE Accounts

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

            LastModified = SYSDATE

        WHERE AccountID = acc.AccountID;

    END LOOP;

    COMMIT;

END;

/

BEGIN

    ProcessMonthlyInterest;

END;

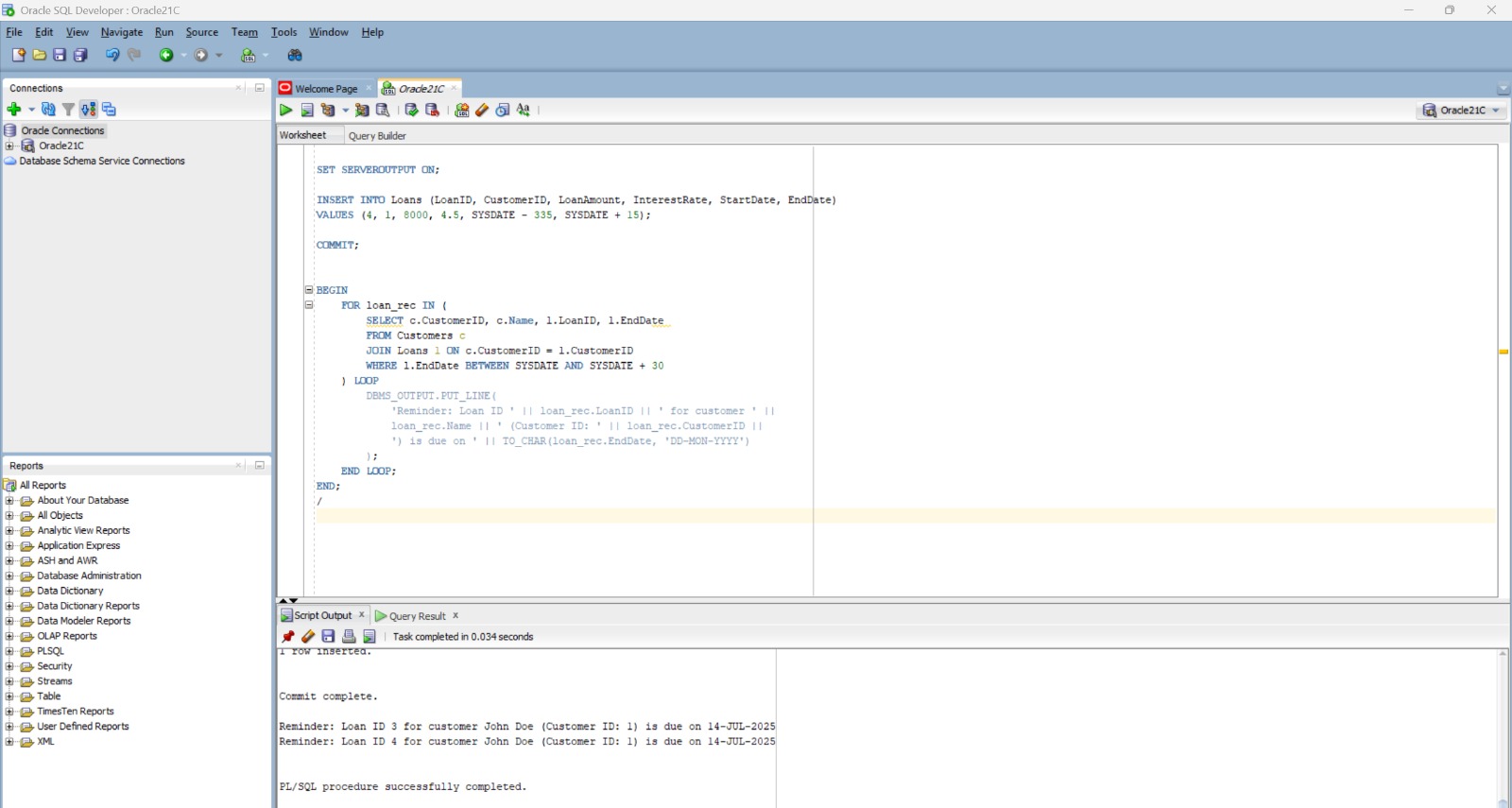
/

SELECT AccountID, CustomerID, AccountType, Balance, LastModified

FROM Accounts

WHERE AccountType = 'Savings';

**OUTPUT :**



**PROBLEM STATEMENT :**

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

* + **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.

**CODE :**

SET SERVEROUTPUT ON;

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

VALUES (3, 'Karthik ECE', 'Engineer', 40000, 'ECE', TO\_DATE('2019-08-01', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_department IN VARCHAR2,

    p\_bonus\_percent IN NUMBER

) IS

    v\_count NUMBER := 0;

BEGIN

    UPDATE Employees

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

    WHERE Department = p\_department;

    v\_count := SQL%ROWCOUNT;

    DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_bonus\_percent || '% applied to ' || v\_count || ' employee(s) in department ' || p\_department);

    COMMIT;

END;

/

BEGIN

    UpdateEmployeeBonus('ECE', 15);

END;

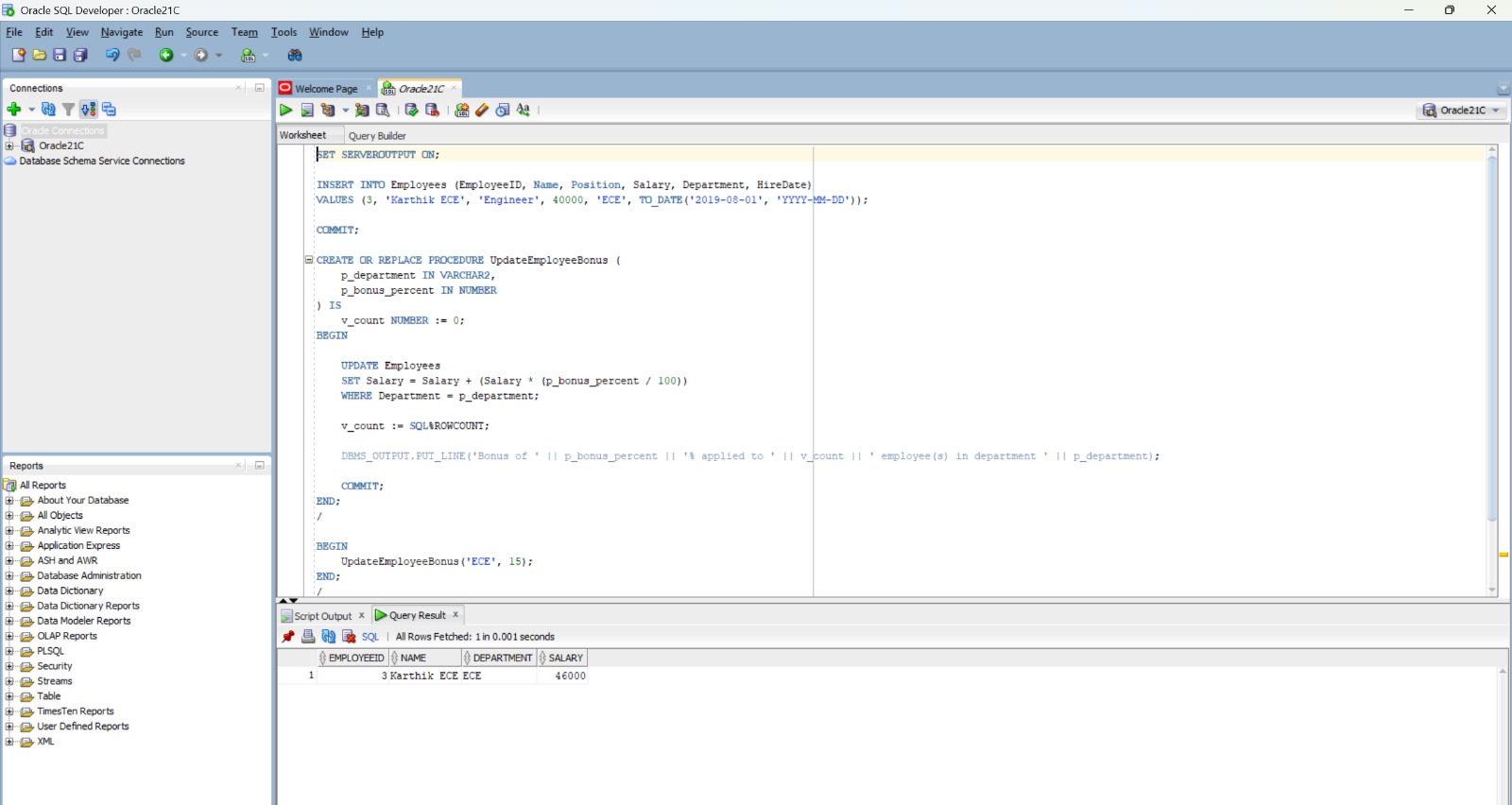
/

SELECT EmployeeID, Name, Department, Salary

FROM Employees

WHERE Department = 'ECE';

**OUTPUT :**



**PROBLEM STATEMENT :**

**Scenario 3:** Customers should be able to transfer funds between their accounts.

* + **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.

**CODE :**

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_source\_account\_id      IN NUMBER,

    p\_destination\_account\_id IN NUMBER,

    p\_amount                 IN NUMBER

) IS

    v\_source\_balance NUMBER;

BEGIN

    SELECT Balance INTO v\_source\_balance

    FROM Accounts

    WHERE AccountID = p\_source\_account\_id

    FOR UPDATE;

    IF v\_source\_balance < p\_amount THEN

        DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance in source account.');

        ROLLBACK;

        RETURN;

    END IF;

    UPDATE Accounts

    SET Balance = Balance - p\_amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_source\_account\_id;

    UPDATE Accounts

    SET Balance = Balance + p\_amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_destination\_account\_id;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Transfer of $' || p\_amount || ' from Account ' ||

                         p\_source\_account\_id || ' to Account ' ||

                         p\_destination\_account\_id || ' completed.');

END;

/

BEGIN

    TransferFunds(1, 2, 500);

END;

/

SELECT AccountID, CustomerID, AccountType, Balance

FROM Accounts

WHERE AccountID IN (1, 2);

**OUTPUT :**

