**Superset ID:6373322**

**COGNITZANT DIGITAL NURTURE 4.0 JAVA FSE**

**WEEK-2:PL/SQL EXERCISES**

**Exercise 1: Control Structures**

**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.

**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.

**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:**

**TABLES CREATION:**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Accounts CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Customers CASCADE CONSTRAINTS';

EXECUTE IMMEDIATE 'DROP TABLE Employees CASCADE CONSTRAINTS';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Create Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE,

IsVIP VARCHAR2(5)

);

-- Create Accounts table

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20),

Balance NUMBER,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Employees table

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Position VARCHAR2(50),

Salary NUMBER,

Department VARCHAR2(50),

HireDate DATE

);

**INSERTING DATA:**

-- Insert Customers

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP)

VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 9000, SYSDATE, NULL);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified, IsVIP)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 15000, SYSDATE, NULL);

-- Insert Accounts

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 Loans

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

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

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

VALUES (2, 2, 3000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 30));

-- Insert Employees

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'));

COMMIT;

**SCENARIO-1:**

BEGIN

FOR c IN (SELECT CustomerID, DOB FROM Customers) LOOP

DECLARE

v\_age NUMBER;

BEGIN

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, c.DOB)/12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = c.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Discount applied to customer ID ' || c.CustomerID);

END IF;

END;

END LOOP;

END;

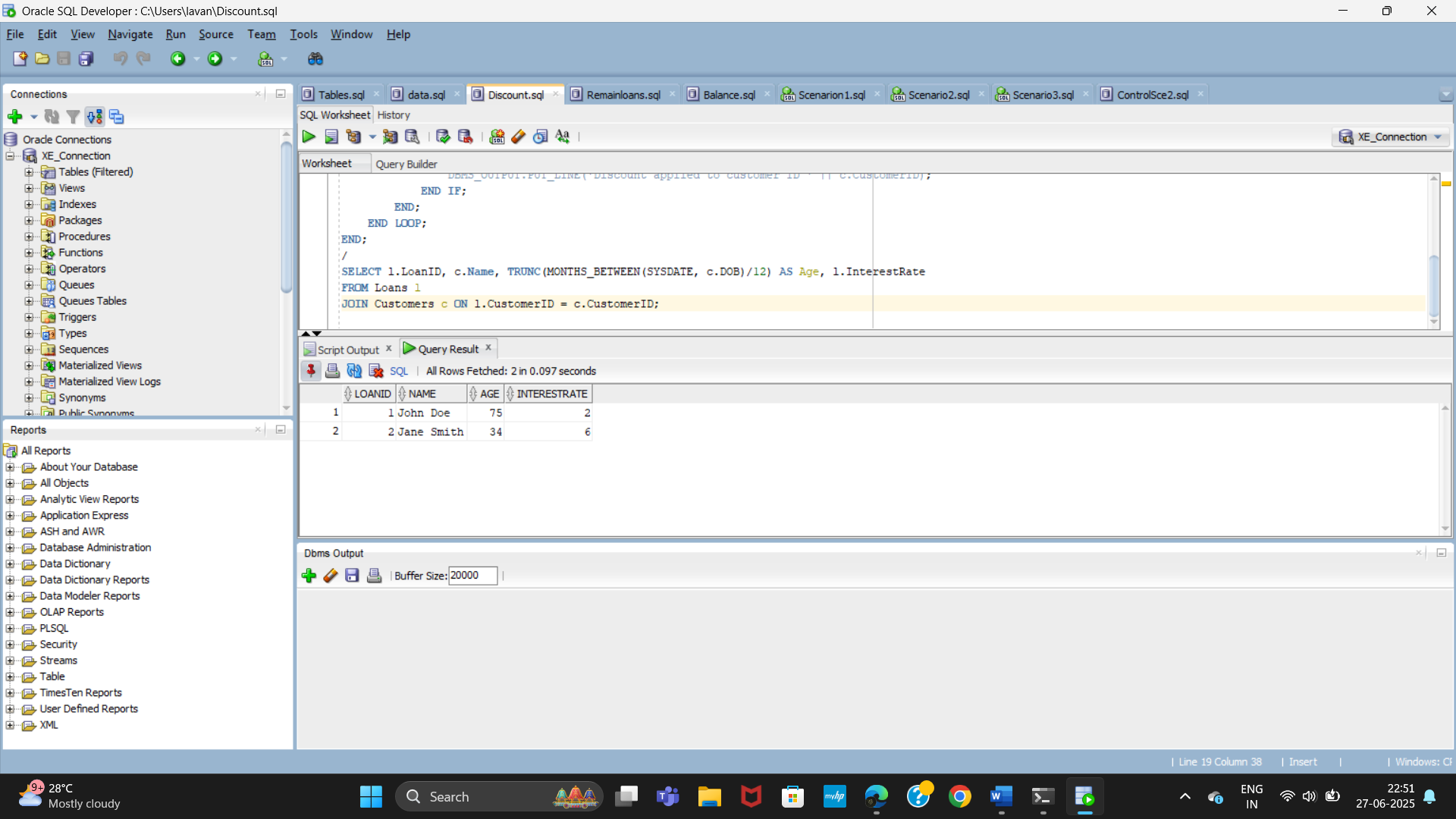
/

SELECT l.LoanID, c.Name, TRUNC(MONTHS\_BETWEEN(SYSDATE, c.DOB)/12) AS Age, l.InterestRate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID;

**OUTPUT:**



**SCENARIO-2:**

BEGIN

FOR c IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF c.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = c.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('VIP status set for Customer ID: ' || c.CustomerID);

ELSE

UPDATE Customers

SET IsVIP = 'FALSE'

WHERE CustomerID = c.CustomerID;

END IF;

END LOOP;

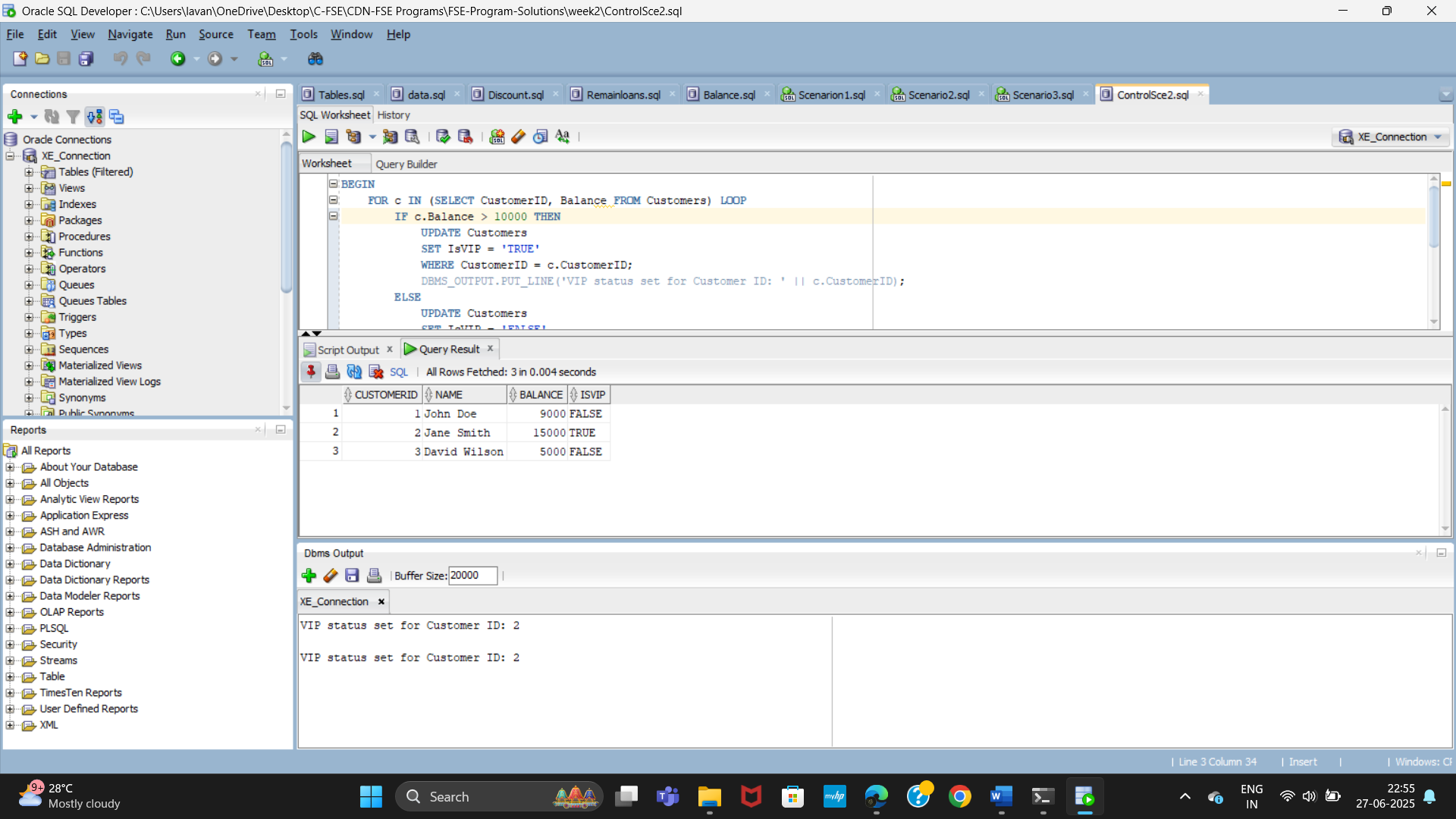
END;

/

SELECT CustomerID, Name, Balance, IsVIP

FROM Customers;

**OUTPUT**:



**SCENARIO-3:**

BEGIN

FOR rec IN (

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

FROM Loans l

JOIN Customers c ON c.CustomerID = l.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID ||

' for ' || rec.Name ||

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

END LOOP;

END;

/

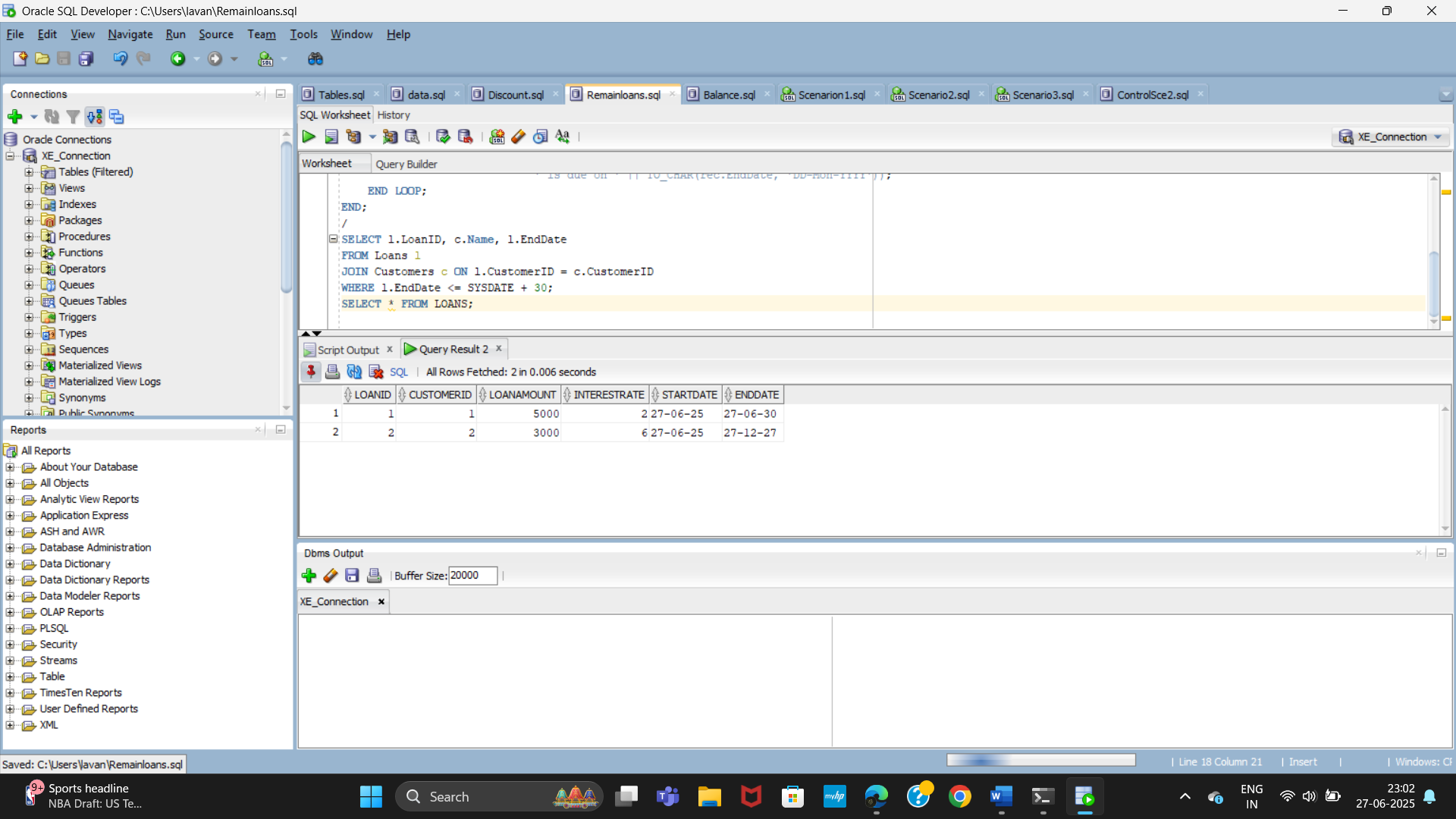
SELECT l.LoanID, c.Name, l.EndDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30;

OUTPUT:



**Exercise 3: Stored Procedures**

**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.

**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.

**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:**

**SCENARIO-1:**

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)

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('Interest added for AccountID: ' || acc.AccountID);

END LOOP;

COMMIT;

END;

/

BEGIN

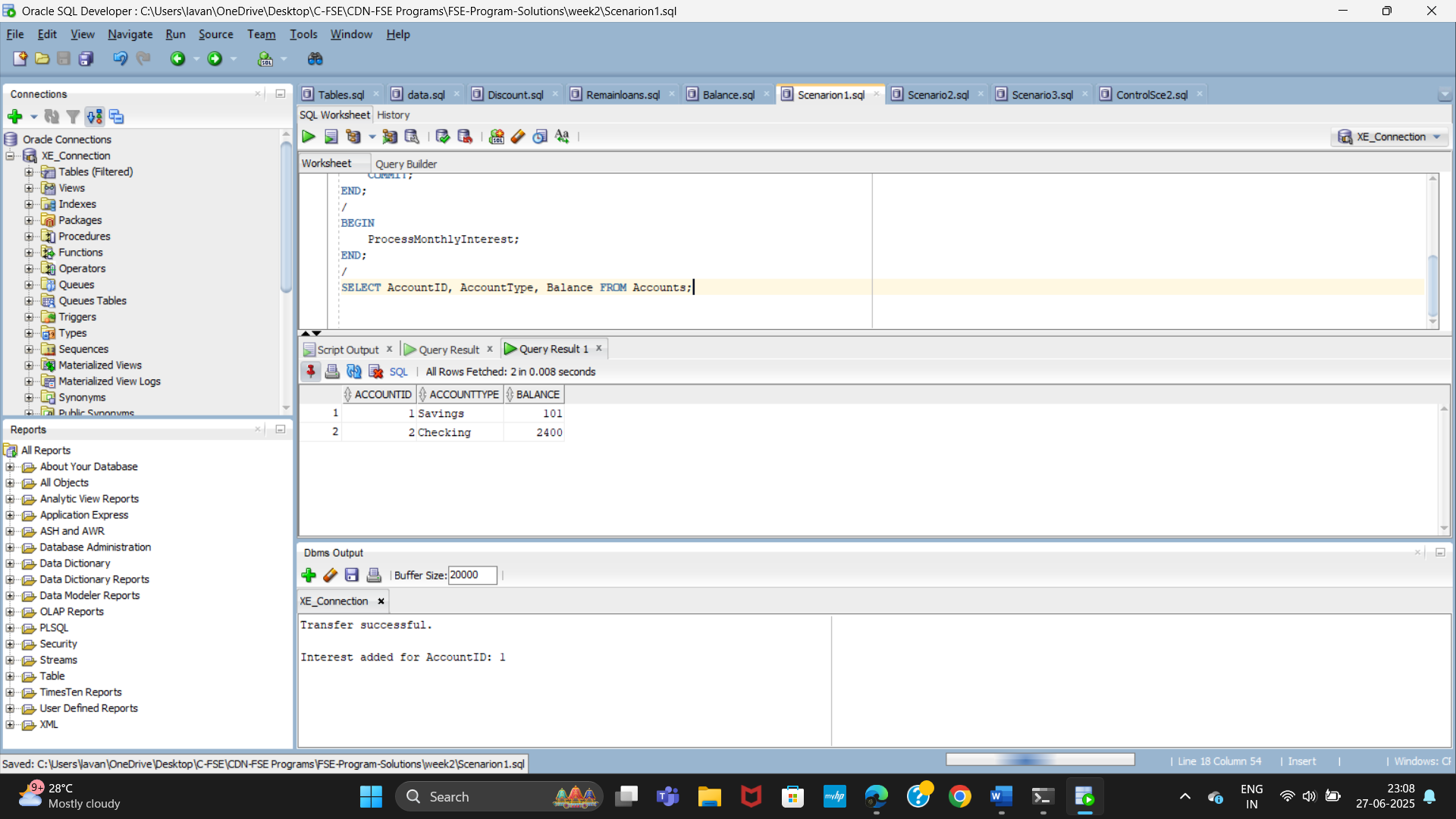
ProcessMonthlyInterest;

END;

/

SELECT AccountID, AccountType, Balance FROM Accounts;

**OUTPUT**:



**SCENARIO-2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_Department IN VARCHAR2,

p\_BonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_Department;

IF SQL%ROWCOUNT = 0 THEN

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

ELSE

DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' employees updated in department: ' || p\_Department);

END IF;

COMMIT;

END;

/

BEGIN

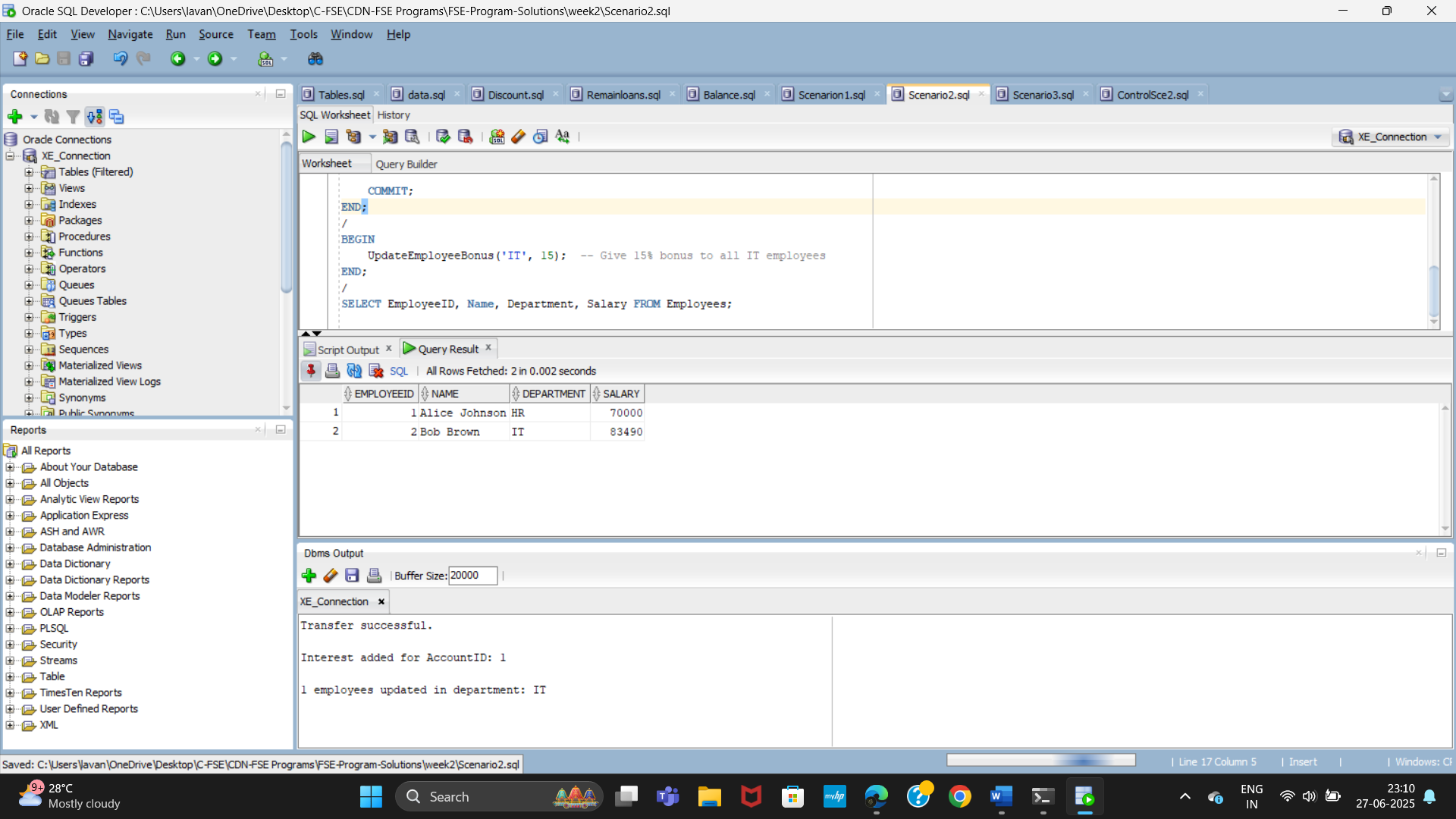
UpdateEmployeeBonus('IT', 15); -- Give 15% bonus to all IT employees

END;

/

SELECT EmployeeID, Name, Department, Salary FROM Employees;

**OUTPUT:**



**SCENARIO-3:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAccount IN NUMBER,

p\_ToAccount IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

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

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\_FromAccount;

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

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_Amount || ' from Account ' || p\_FromAccount || ' to Account ' || p\_ToAccount);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

BEGIN

TransferFunds(2, 1, 200); -- Example: transfer 200 from Account 2 to Account 1

END;

/

SELECT AccountID, CustomerID, Balance FROM Accounts ORDER BY AccountID;

**OUTPUT:**

