EXERCISE 1:

--SCENARIO 2

```
BEGIN
   FOR cust IN (SELECT CustomerID, Name, DOB FROM Customers) LOOP
      IF MONTHS_BETWEEN(SYSDATE, cust.DOB) / 12 > 60 THEN
         UPDATE Loans
         SET InterestRate = InterestRate - 1
         WHERE CustomerID = cust.CustomerID;
         DBMS_OUTPUT.PUT_LINE('Interest discounted for: ' || cust.Name);
      END IF;
   END LOOP;
   COMMIT;
END;
/
ALTER TABLE Customers ADD IsVIP VARCHAR2(5);
```

--SCENARIO 2

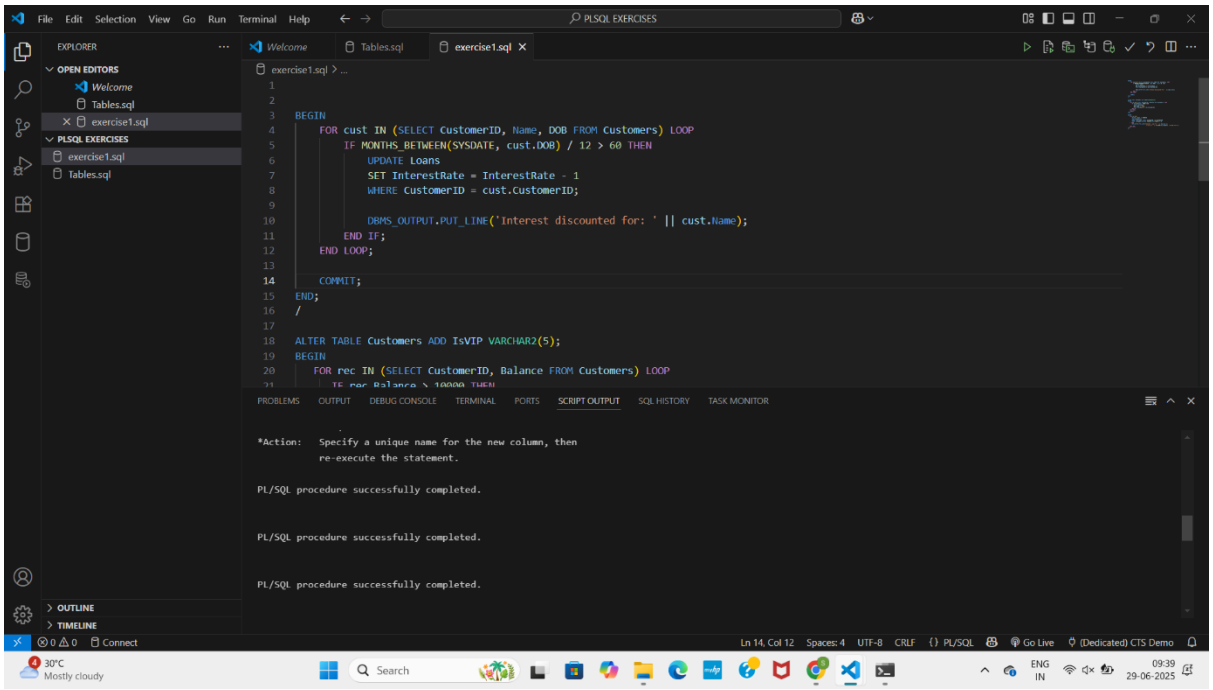
```
BEGIN
  FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP
    IF rec.Balance > 10000 THEN
      UPDATE Customers
      SET IsVIP = 'TRUE'
      WHERE CustomerID = rec.CustomerID;
    END IF;
  END LOOP;
  COMMIT;
END;
```

--SCENARIO 2

```
BEGIN
  FOR rec IN (
    SELECT c.Name, l.EndDate
    FROM Customers c
```

```
    INNER JOIN Loans l ON c.CustomerID = l.CustomerID
    WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30
  ) LOOP
    DBMS_OUTPUT.PUT_LINE('Reminder: Loan for ' || rec.Name ||
                         ' is due on ' || TO_CHAR(rec.EndDate, 'DD-MON-YYYY'));
  END LOOP;
END;
/
```

OUTPUT:



EXERCISE 2:

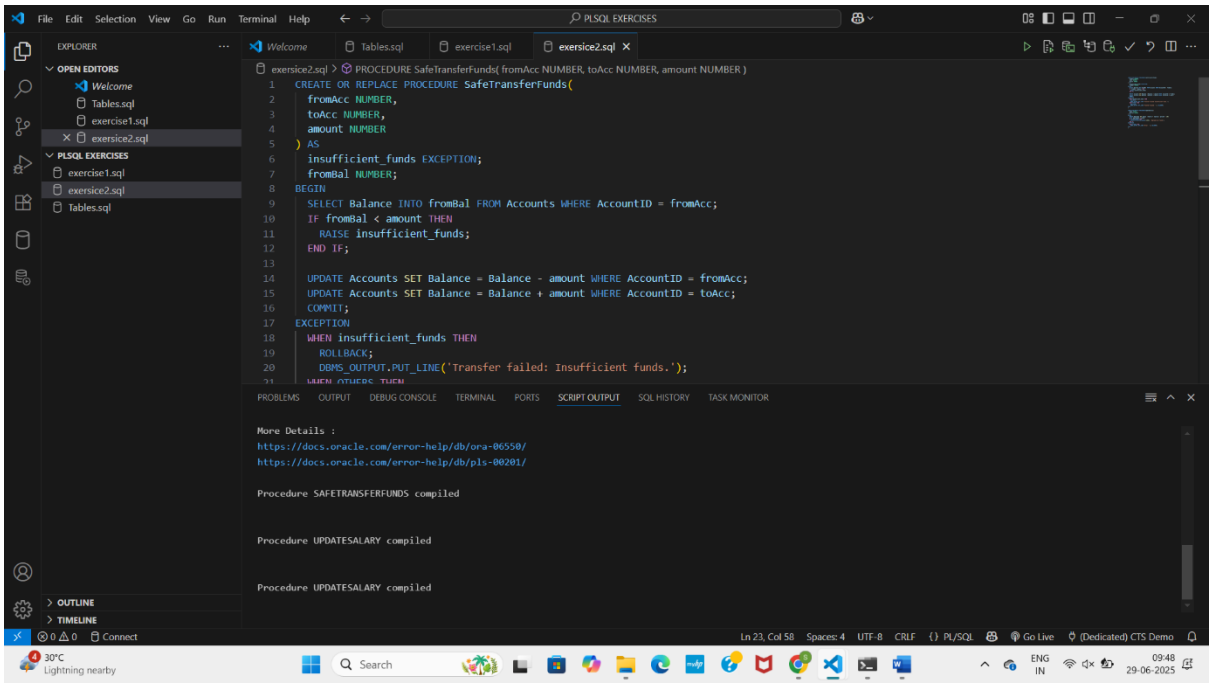
--SCENARIO 1

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds(
  fromAcc NUMBER,
  toAcc NUMBER,
  amount NUMBER
) AS
  insufficient_funds EXCEPTION;
  fromBal NUMBER;
```

```
 --SCENARIO 2
BEGIN
  SELECT Balance INTO fromBal FROM Accounts WHERE AccountID = fromAcc;
  IF fromBal < amount THEN
    RAISE insufficient_funds;
  END IF;
  UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAcc;
  UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = toAcc;
  COMMIT;
EXCEPTION
  WHEN insufficient_funds THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Transfer failed: Insufficient funds.');
  WHEN OTHERS THEN
    ROLLBACK;
    DBMS_OUTPUT.PUT_LINE('Transfer failed: ' || SQLERRM);
END;
/
--SCENARIO 3
CREATE OR REPLACE PROCEDURE UpdateSalary(
  empID NUMBER,
  percent NUMBER
) AS
BEGIN
  UPDATE Employees SET Salary = Salary + (Salary * percent / 100)
  WHERE EmployeeID = empID;
  IF SQL%NOTFOUND THEN
    RAISE_APPLICATION_ERROR(-20001, 'Employee not found');
  END IF;
EXCEPTION
```

```
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
/
```

OUTPUT:



EXERCISE 3:

--SCENARIO 1

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
  UPDATE Accounts
  SET Balance = Balance + (Balance * 0.01)
  WHERE AccountType = 'Savings';
END;
/
```

--SCENARIO 2

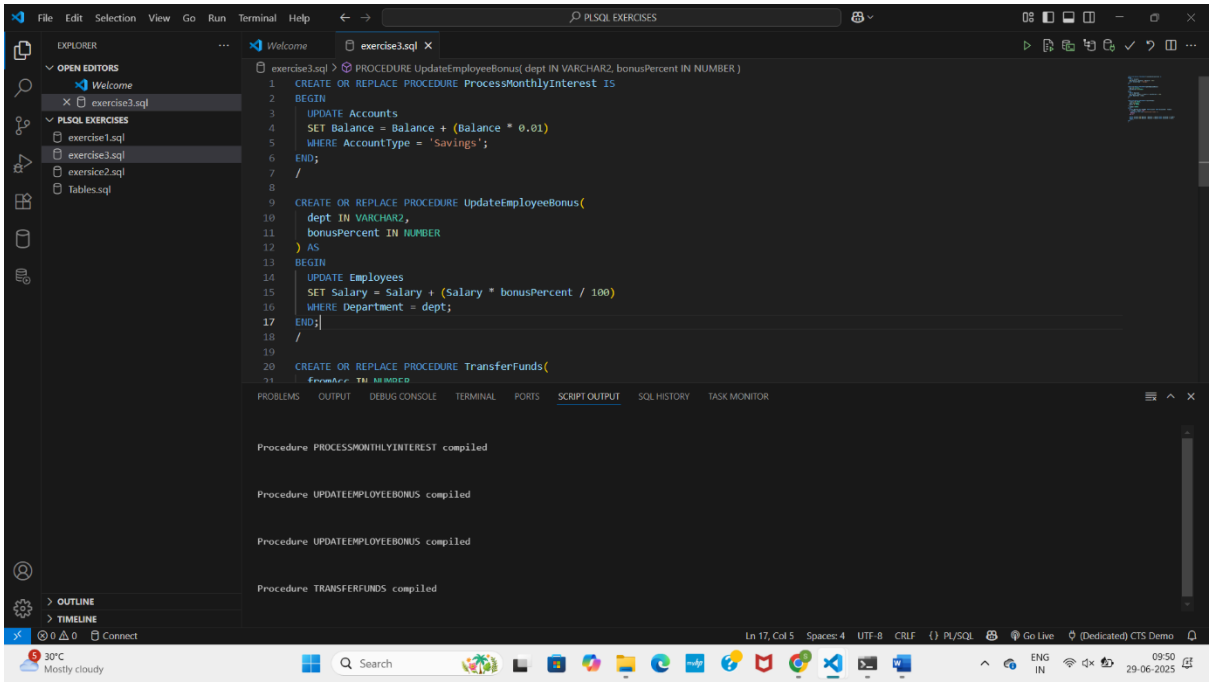
```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(
  dept IN VARCHAR2,
  bonusPercent IN NUMBER
```

```
) AS
BEGIN
  UPDATE Employees
  SET Salary = Salary + (Salary * bonusPercent / 100)
  WHERE Department = dept;
END;
/
```

--SCENARIO 3

```
CREATE OR REPLACE PROCEDURE TransferFunds(
  fromAcc IN NUMBER,
  toAcc IN NUMBER,
  amount IN NUMBER
) AS
  fromBal NUMBER;
BEGIN
  SELECT Balance INTO fromBal FROM Accounts WHERE AccountID = fromAcc;
  IF fromBal < amount THEN
    DBMS_OUTPUT.PUT_LINE('Insufficient funds.');
    RETURN;
  END IF;

  UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAcc;
  UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = toAcc;
  COMMIT;
END;
/
```

OUTPUT:



EXERCISE 4:

-- Scenario 1

```
CREATE OR REPLACE FUNCTION CalculateAge(dob DATE) RETURN NUMBER IS
BEGIN
  RETURN FLOOR(MONTHS_BETWEEN(SYSDATE, dob) / 12);
END;
/
```

-- Scenario 2

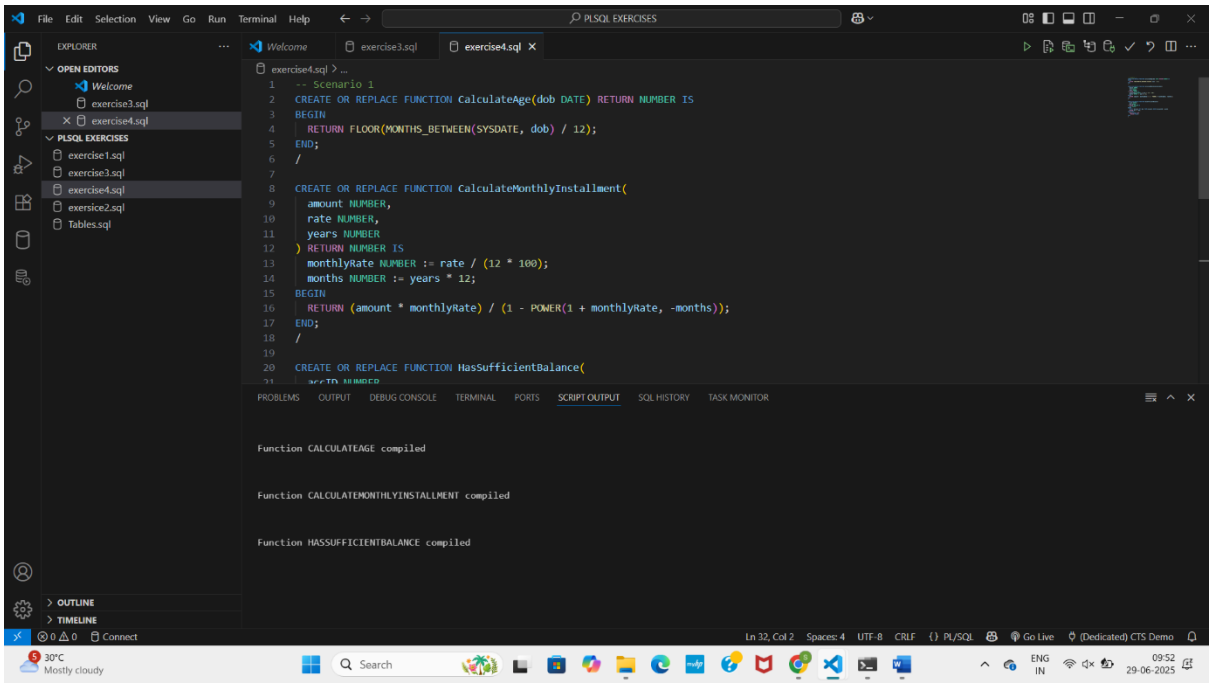
```
CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(
  amount NUMBER,
  rate NUMBER,
  years NUMBER
) RETURN NUMBER IS
  monthlyRate NUMBER := rate / (12 * 100);
  months NUMBER := years * 12;
BEGIN
  RETURN (amount * monthlyRate) / (1 - POWER(1 + monthlyRate, -months));
END;
```

/

-- Scenario 3

```
CREATE OR REPLACE FUNCTION HasSufficientBalance(
  accID NUMBER,
  amt NUMBER
) RETURN BOOLEAN IS
  bal NUMBER;
BEGIN
  SELECT Balance INTO bal FROM Accounts WHERE AccountID = accID;
  RETURN bal >= amt;
EXCEPTION
  WHEN OTHERS THEN
    RETURN FALSE;
END;
/
```

OUTPUT:



EXERCISE 5:

--Scenario 1

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers
FOR EACH ROW
BEGIN
  :NEW.LastModified := SYSDATE;
END;
/
```

--Scenario 2

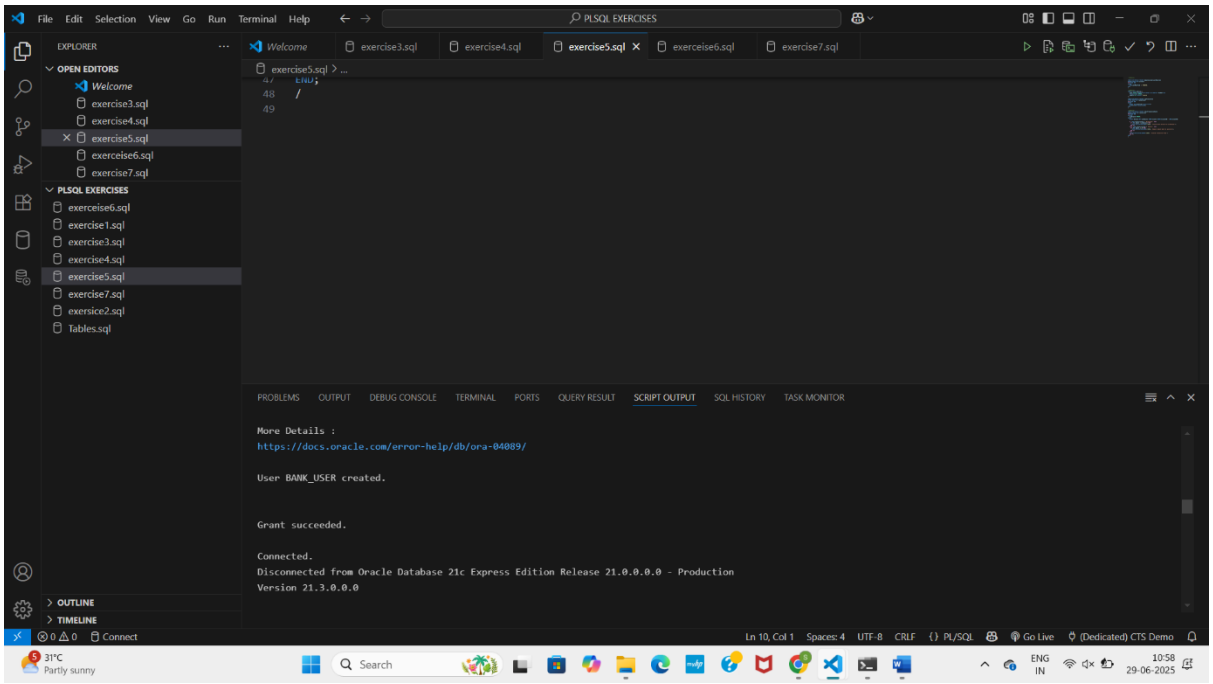
```
CREATE TABLE AuditLog (
  LogID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  TransactionID NUMBER,
  LogDate DATE DEFAULT SYSDATE
);
CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
  INSERT INTO AuditLog(TransactionID)
  VALUES(:NEW.TransactionID);
END;
/
```

--SCENARIO 3

```
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
  accBalance NUMBER;
BEGIN
  SELECT Balance INTO accBalance FROM Accounts WHERE AccountID = :NEW.AccountID;
  IF :NEW.TransactionType = 'Withdrawal' THEN
    IF :NEW.Amount > accBalance THEN
```

```
    RAISE_APPLICATION_ERROR(-20002, 'Insufficient balance for withdrawal');
  END IF;
 ELSIF :NEW.TransactionType = 'Deposit' THEN
  IF :NEW.Amount <= 0 THEN
    RAISE_APPLICATION_ERROR(-20003, 'Deposit amount must be positive');
  END IF;
 ELSE
  RAISE_APPLICATION_ERROR(-20004, 'Invalid transaction type');
 END IF;
END;
/
```

OUTPUT:



EXERCISE 6:

-- Scenario 1

```
DECLARE
 CURSOR cur IS
  SELECT c.CustomerID, c.Name, t.Amount, t.TransactionDate
  FROM Customers c
  JOIN Accounts a ON c.CustomerID = a.CustomerID
```

```
    JOIN Transactions t ON a.AccountID = t.AccountID
    WHERE TO_CHAR(t.TransactionDate, 'MMYYYY') = TO_CHAR(SYSDATE, 'MMYYYY');
BEGIN
  FOR rec IN cur LOOP
    DBMS_OUTPUT.PUT_LINE('Customer: ' || rec.Name || ' - ' || rec.TransactionDate || ' - Amount: '
|| rec.Amount);
  END LOOP;
END;
/
```

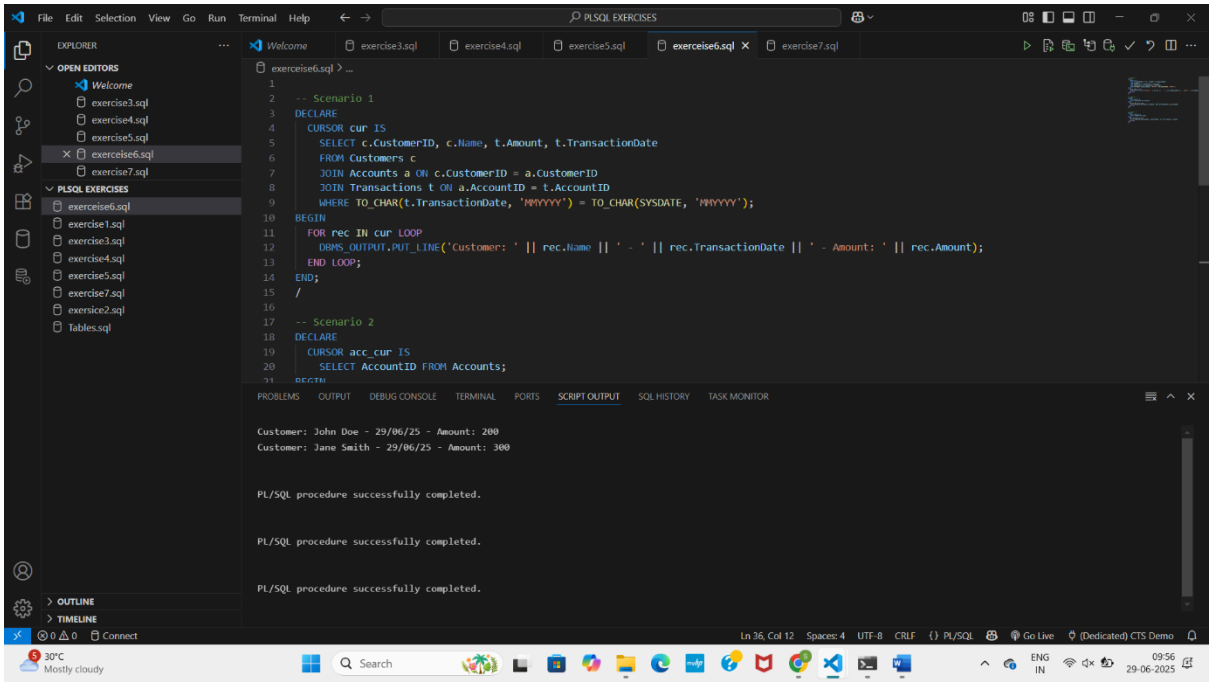
-- Scenario 2

```
DECLARE
  CURSOR acc_cur IS
    SELECT AccountID FROM Accounts;
BEGIN
  FOR acc IN acc_cur LOOP
    UPDATE Accounts SET Balance = Balance - 100 WHERE AccountID = acc.AccountID;
  END LOOP;
END;
/
```

-- Scenario 3

```
DECLARE
  CURSOR loan_cur IS
    SELECT LoanID FROM Loans;
BEGIN
  FOR l IN loan_cur LOOP
    UPDATE Loans SET InterestRate = InterestRate + 0.5 WHERE LoanID = l.LoanID;
  END LOOP;
END;
/
```

OUTPUT:



EXERCISE 7:

-- Scenario 1

```
CREATE OR REPLACE PACKAGE CustomerManagement AS
  PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, bal NUMBER);
  PROCEDURE UpdateCustomer(id NUMBER, name VARCHAR2);
  FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER;
END;
/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS
  PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, bal NUMBER) IS
  BEGIN
    INSERT INTO Customers(CustomerID, Name, DOB, Balance, LastModified)
    VALUES (id, name, dob, bal, SYSDATE);
  END;

  PROCEDURE UpdateCustomer(id NUMBER, name VARCHAR2) IS
  BEGIN
```

```
    UPDATE Customers SET Name = name, LastModified = SYSDATE WHERE CustomerID = id;
  END;

  FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER IS
    bal NUMBER;
  BEGIN
    SELECT Balance INTO bal FROM Customers WHERE CustomerID = id;
    RETURN bal;
  END;
END;
/
```

-- Scenario 2

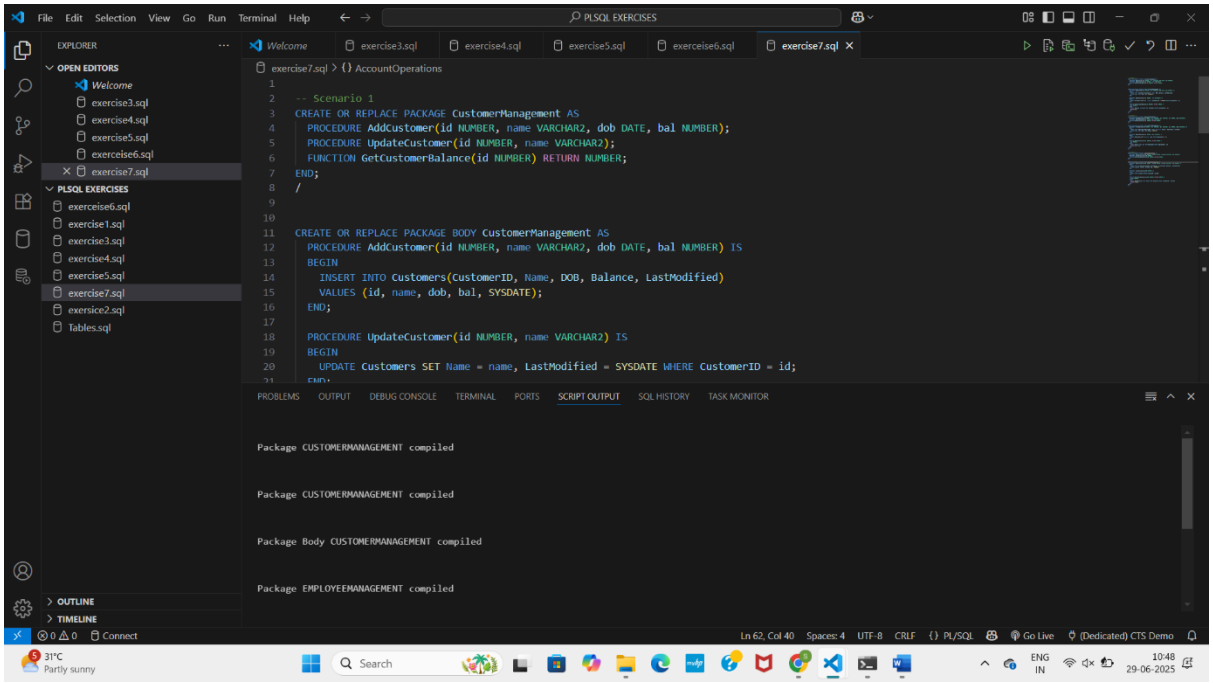
```
CREATE OR REPLACE PACKAGE EmployeeManagement AS
  PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, pos VARCHAR2, sal NUMBER, dept
VARCHAR2);
  PROCEDURE UpdateEmployee(id NUMBER, pos VARCHAR2);
  FUNCTION GetAnnualSalary(id NUMBER) RETURN NUMBER;
END;
/
CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS
  PROCEDURE HireEmployee(id NUMBER, name VARCHAR2, pos VARCHAR2, sal NUMBER, dept
VARCHAR2) IS
  BEGIN
    INSERT INTO Employees(EmployeeID, Name, Position, Salary, Department, HireDate)
    VALUES (id, name, pos, sal, dept, SYSDATE);
  END;
  PROCEDURE UpdateEmployee(id NUMBER, pos VARCHAR2) IS
  BEGIN
    UPDATE Employees SET Position = pos WHERE EmployeeID = id;
  END;
  FUNCTION GetAnnualSalary(id NUMBER) RETURN NUMBER IS
    sal NUMBER;
```

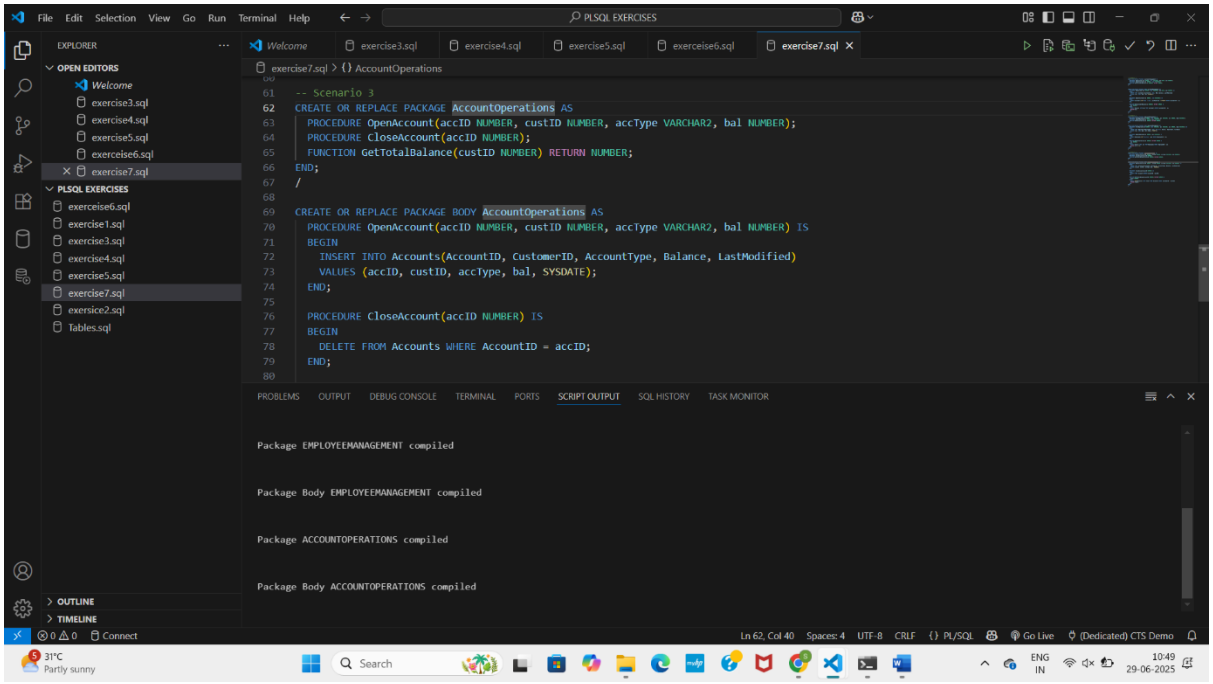
```
  BEGIN
    SELECT Salary INTO sal FROM Employees WHERE EmployeeID = id;
    RETURN sal * 12;
  END;
END;
/
-- Scenario 3
CREATE OR REPLACE PACKAGE AccountOperations AS
  PROCEDURE OpenAccount(accID NUMBER, custID NUMBER, accType VARCHAR2, bal NUMBER);
  PROCEDURE CloseAccount(accID NUMBER);
  FUNCTION GetTotalBalance(custID NUMBER) RETURN NUMBER;
END;
/
CREATE OR REPLACE PACKAGE BODY AccountOperations AS
  PROCEDURE OpenAccount(accID NUMBER, custID NUMBER, accType VARCHAR2, bal NUMBER) IS
  BEGIN
    INSERT INTO Accounts(AccountID, CustomerID, AccountType, Balance, LastModified)
    VALUES (accID, custID, accType, bal, SYSDATE);
  END;

  PROCEDURE CloseAccount(accID NUMBER) IS
  BEGIN
    DELETE FROM Accounts WHERE AccountID = accID;
  END;

  FUNCTION GetTotalBalance(custID NUMBER) RETURN NUMBER IS
    total NUMBER;
  BEGIN
    SELECT SUM(Balance) INTO total FROM Accounts WHERE CustomerID = custID;
    RETURN total;
  END;
```

END;

/

OUTPUT:


exercise7.sql > {} AccountOperations
-- Scenario 1
CREATE OR REPLACE PACKAGE CustomerManagement AS
PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, bal NUMBER);
PROCEDURE UpdateCustomer(id NUMBER, name VARCHAR2);
FUNCTION GetCustomerBalance(id NUMBER) RETURN NUMBER;
END;
/
CREATE OR REPLACE PACKAGE BODY CustomerManagement AS
PROCEDURE AddCustomer(id NUMBER, name VARCHAR2, dob DATE, bal NUMBER) IS
BEGIN
INSERT INTO Customers(CustomerID, Name, DOB, Balance, LastModified)
VALUES (id, name, dob, bal, SYSDATE);
END;
PROCEDURE UpdateCustomer(id NUMBER, name VARCHAR2) IS
BEGIN
UPDATE Customers SET Name = name, LastModified = SYSDATE WHERE CustomerID = id;
SCRIPT OUTPUT
Package CUSTOMERMANAGEMENT compiled
Package CUSTOMERMANAGEMENT compiled
Package Body CUSTOMERMANAGEMENT compiled
Package EMPLOYEEMANAGEMENT compiled


exercise7.sql > {} AccountOperations
-- Scenario 3
CREATE OR REPLACE PACKAGE AccountOperations AS
PROCEDURE OpenAccount(accID NUMBER, custID NUMBER, accType VARCHAR2, bal NUMBER);
PROCEDURE CloseAccount(accID NUMBER);
FUNCTION GetTotalBalance(custID NUMBER) RETURN NUMBER;
END;
/
CREATE OR REPLACE PACKAGE BODY AccountOperations AS
PROCEDURE OpenAccount(accID NUMBER, custID NUMBER, accType VARCHAR2, bal NUMBER) IS
BEGIN
INSERT INTO Accounts(AccountID, CustomerID, AccountType, Balance, LastModified)
VALUES (accID, custID, accType, bal, SYSDATE);
END;
PROCEDURE CloseAccount(accID NUMBER) IS
BEGIN
DELETE FROM Accounts WHERE AccountID = accID;
END;
SCRIPT OUTPUT
Package EMPLOYEEMANAGEMENT compiled
Package Body EMPLOYEEMANAGEMENT compiled
Package ACCOUNTOPERATIONS compiled
Package Body ACCOUNTOPERATIONS compiled