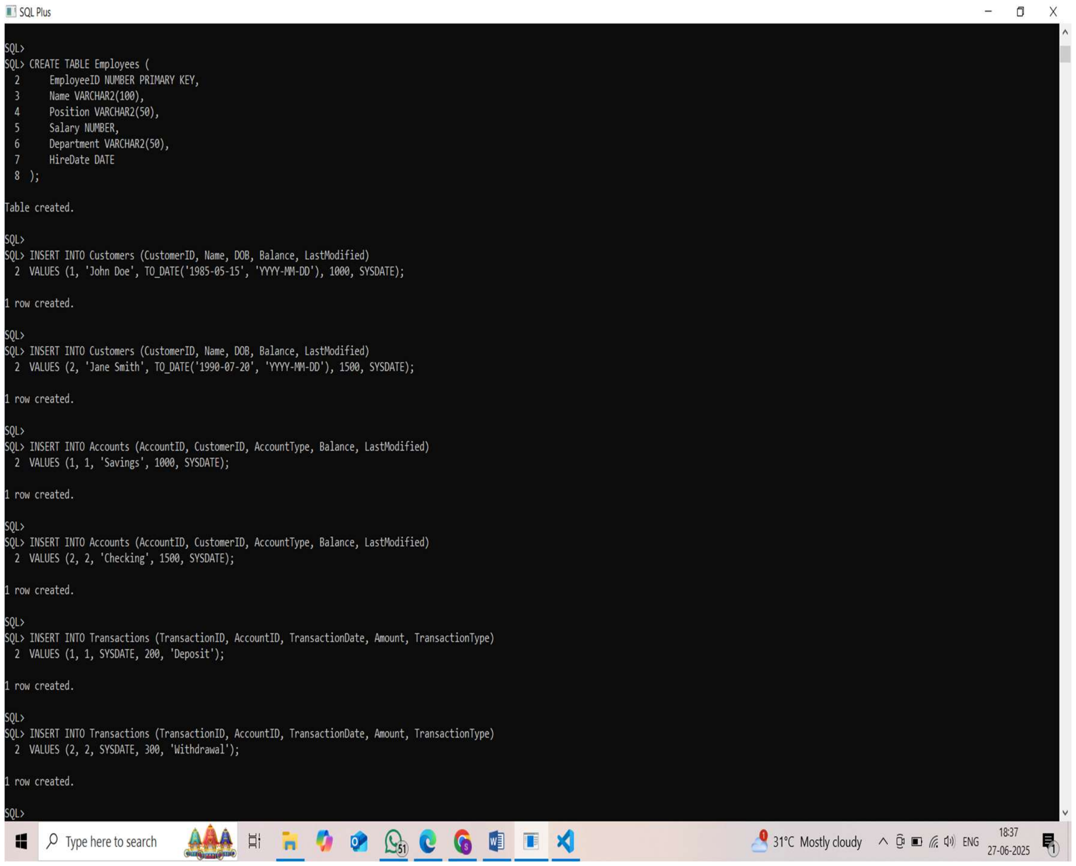
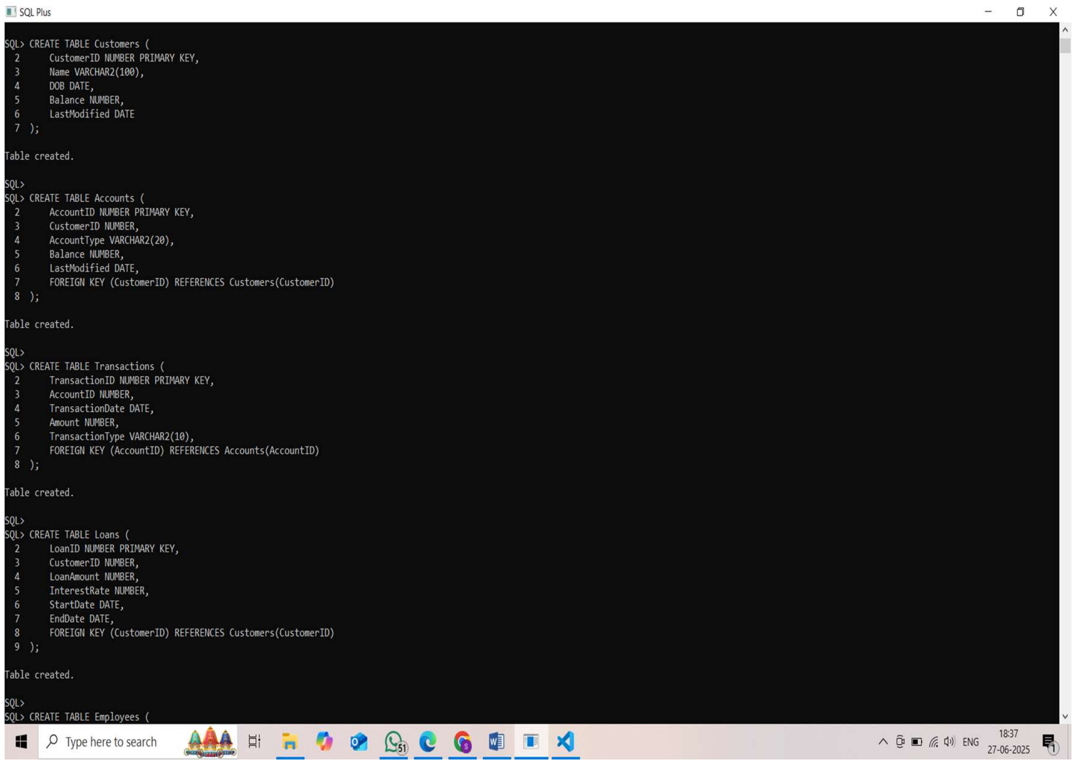
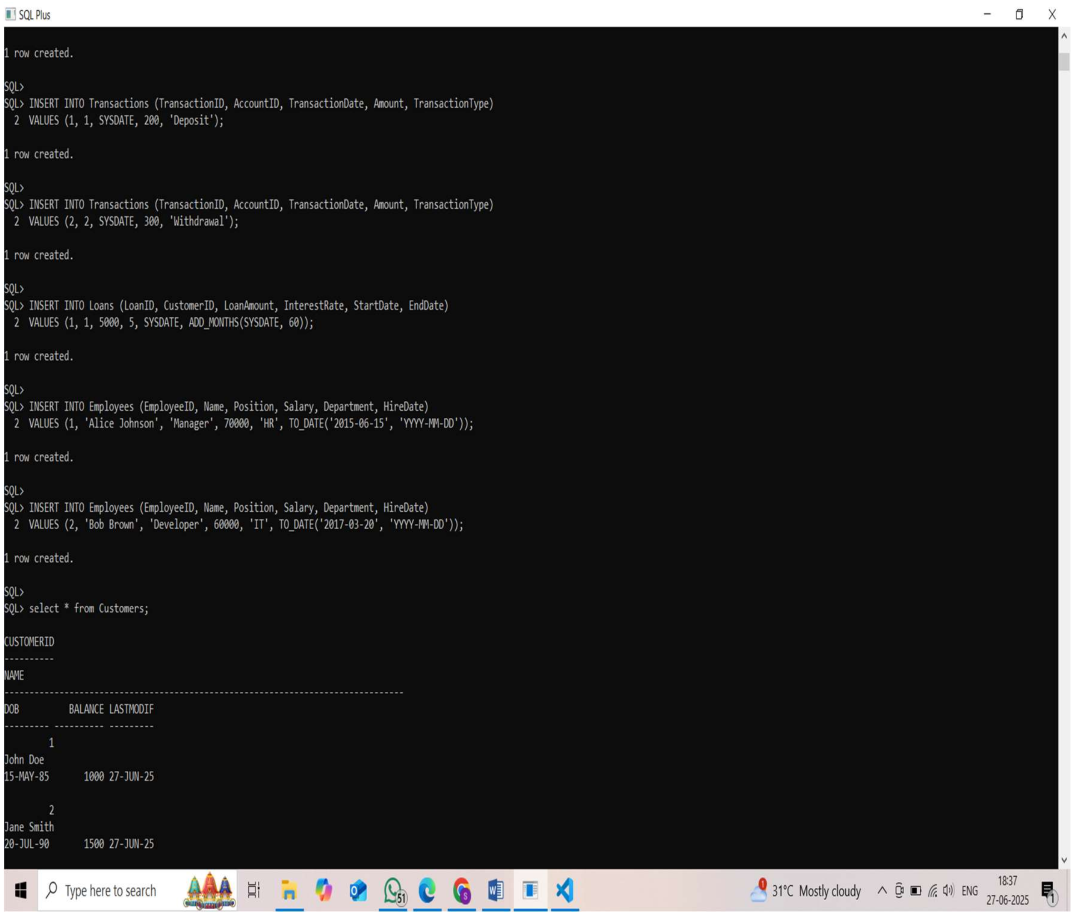
WEEK 2



```
SQL>
SQL> CREATE TABLE Employees (
    EmployeeID NUMBER PRIMARY KEY,
    Name VARCHAR2(100),
    Position VARCHAR2(50),
    Salary NUMBER,
    Department VARCHAR2(50),
    HireDate DATE
);

Table created.

SQL>
SQL> INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
VALUES (1, 'John Doe', TO_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

1 row created.

SQL>
SQL> INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
VALUES (2, 'Jane Smith', TO_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

1 row created.

SQL>
SQL> INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)
VALUES (1, 1, 'Savings', 1000, SYSDATE);

1 row created.

SQL>
SQL> INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)
VALUES (2, 2, 'Checking', 1500, SYSDATE);

1 row created.

SQL>
SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (1, 1, SYSDATE, 200, 'Deposit');

1 row created.

SQL>
SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

1 row created.

SQL>
```



```
SQL> CREATE TABLE Customers (
    CustomerID NUMBER PRIMARY KEY,
    Name VARCHAR2(100),
    DOB DATE,
    Balance NUMBER,
    LastModified DATE
);

Table created.

SQL>
SQL> CREATE TABLE Accounts (
    AccountID NUMBER PRIMARY KEY,
    CustomerID NUMBER,
    AccountType VARCHAR2(20),
    Balance NUMBER,
    LastModified DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

Table created.

SQL>
SQL> CREATE TABLE Transactions (
    TransactionID NUMBER PRIMARY KEY,
    AccountID NUMBER,
    TransactionDate DATE,
    Amount NUMBER,
    TransactionType VARCHAR2(10),
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);

Table created.

SQL>
SQL> CREATE TABLE Loans (
    LoanID NUMBER PRIMARY KEY,
    CustomerID NUMBER,
    LoanAmount NUMBER,
    InterestRate NUMBER,
    StartDate DATE,
    EndDate DATE,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);

Table created.

SQL>
SQL> CREATE TABLE Employees (
```



```
1 row created.

SQL>
SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
  2  VALUES (1, 1, SYSDATE, 200, 'Deposit');

1 row created.

SQL>
SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
  2  VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

1 row created.

SQL>
SQL> INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)
  2  VALUES (1, 1, 5000, 5, SYSDATE, ADD_MONTHS(SYSDATE, 60));

1 row created.

SQL>
SQL> INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)
  2  VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO_DATE('2015-06-15', 'YYYY-MM-DD'));

1 row created.

SQL>
SQL> INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)
  2  VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO_DATE('2017-03-20', 'YYYY-MM-DD'));

1 row created.

SQL>
SQL> select * from Customers;

CUSTOMERID
----------
NAME
--------------------------------------------------------------------------------
DOB          BALANCE LASTMODIF
--------- ---------- ---------
         1
John Doe
15-MAY-85       1000 27-JUN-25

         2
Jane Smith
20-JUL-90       1500 27-JUN-25
```

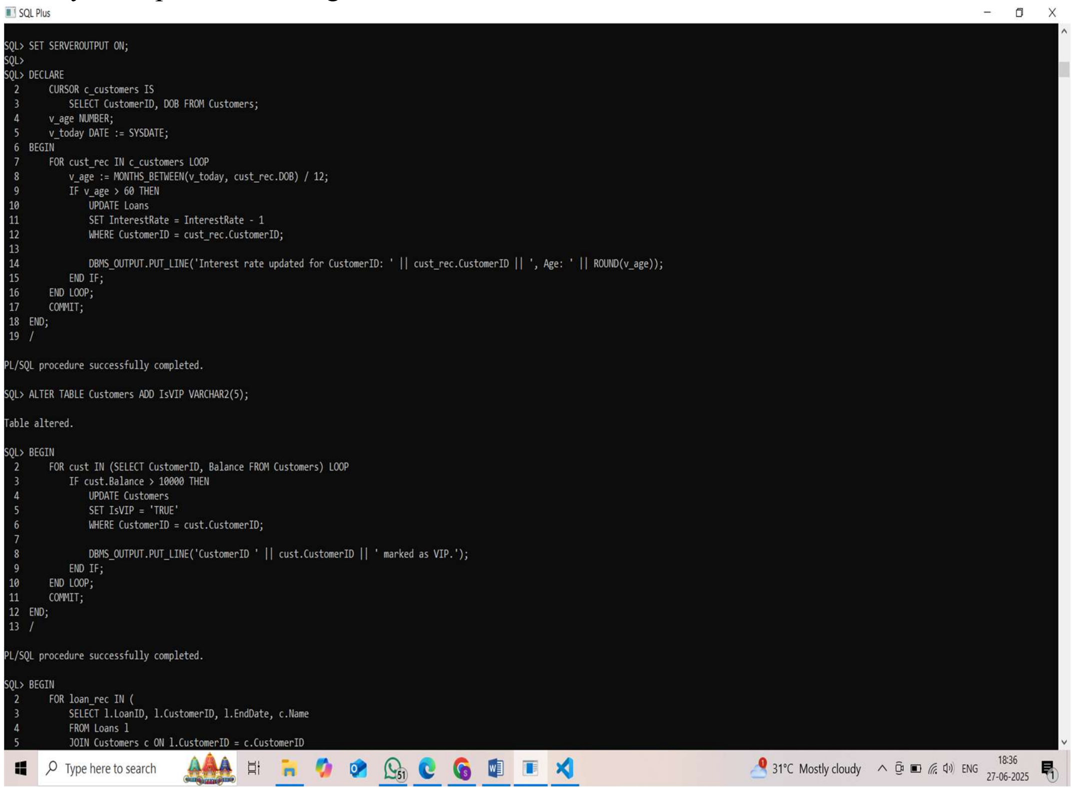
Exercise 1: Control Structures

Scenario 1: The bank wants to apply a 1% discount to the loan interest rate for all customers who are above 60 years old.
Question: Write a PL/SQL block that loops through all customers, calculates their age from DOB, and if the age is over 60, reduce the interest rate by 1% for their loans.

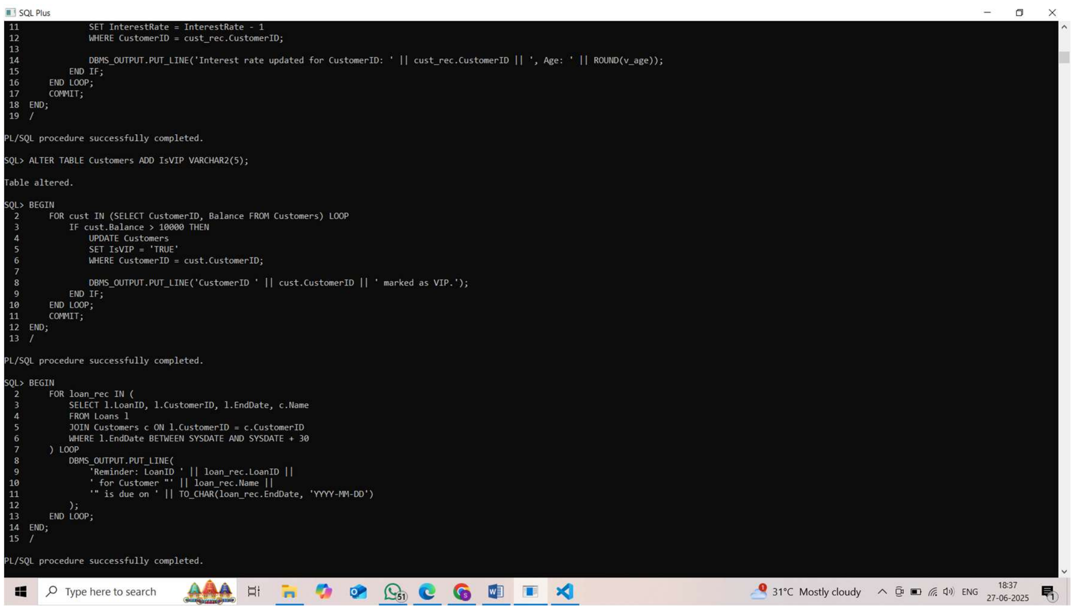
Scenario 2: The bank promotes customers to VIP status if they maintain high balances.
Question: Write a PL/SQL block that iterates through all customers and, if the balance is over $10,000, sets a column IsVIP to 'Y' or TRUE.

Scenario 3: The bank needs to send reminders for loans that are due in the next 30 days.
Question: Write a PL/SQL block that fetches all loans where the end date is within the next

30 days and prints a message: "Reminder: Loan for Customer <ID> is due soon".



```
SQL> SET SERVEROUTPUT ON;
SQL>
SQL> DECLARE
    CURSOR c_customers IS
        SELECT CustomerID, DOB FROM Customers;
    v_age NUMBER;
    v_today DATE := SYSDATE;
BEGIN
    FOR cust_rec IN c_customers LOOP
        v_age := MONTHS_BETWEEN(v_today, cust_rec.DOB) / 12;
        IF v_age > 60 THEN
            UPDATE Loans
            SET InterestRate = InterestRate - 1
            WHERE CustomerID = cust_rec.CustomerID;

            DBMS_OUTPUT.PUT_LINE('Interest rate updated for CustomerID: ' || cust_rec.CustomerID || ', Age: ' || ROUND(v_age));
        END IF;
    END LOOP;
    COMMIT;
END;
/

PL/SQL procedure successfully completed.

SQL> ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

Table altered.

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

            DBMS_OUTPUT.PUT_LINE('CustomerID ' || cust.CustomerID || ' marked as VIP.');
        END IF;
    END LOOP;
    COMMIT;
END;
/

PL/SQL procedure successfully completed.

SQL> BEGIN
    FOR loan_rec IN (
        SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name
        FROM Loans l
        JOIN Customers c ON l.CustomerID = c.CustomerID
```



```
            SET InterestRate = InterestRate - 1
            WHERE CustomerID = cust_rec.CustomerID;

            DBMS_OUTPUT.PUT_LINE('Interest rate updated for CustomerID: ' || cust_rec.CustomerID || ', Age: ' || ROUND(v_age));
        END IF;
    END LOOP;
    COMMIT;
END;
/

PL/SQL procedure successfully completed.

SQL> ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

Table altered.

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

            DBMS_OUTPUT.PUT_LINE('CustomerID ' || cust.CustomerID || ' marked as VIP.');
        END IF;
    END LOOP;
    COMMIT;
END;
/

PL/SQL procedure successfully completed.
```

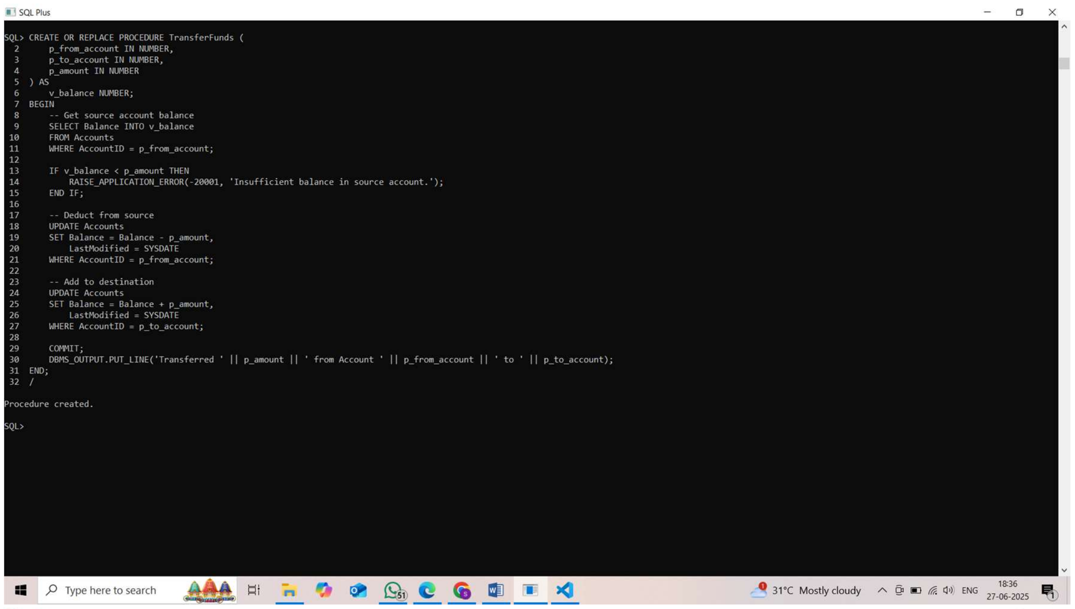
```
SQL> BEGIN
    FOR loan_rec IN (
        SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name
        FROM Loans l
        JOIN Customers c ON l.CustomerID = c.CustomerID
        WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30
    ) LOOP
        DBMS_OUTPUT.PUT_LINE(
            'Reminder: LoanID ' || loan_rec.LoanID ||
            ' for Customer "' || loan_rec.Name ||
            '" is due on ' || TO_CHAR(loan_rec.EndDate, 'YYYY-MM-DD')
        );
    END LOOP;
END;
/

PL/SQL procedure successfully completed.
```

Exercise 3: Stored Procedures

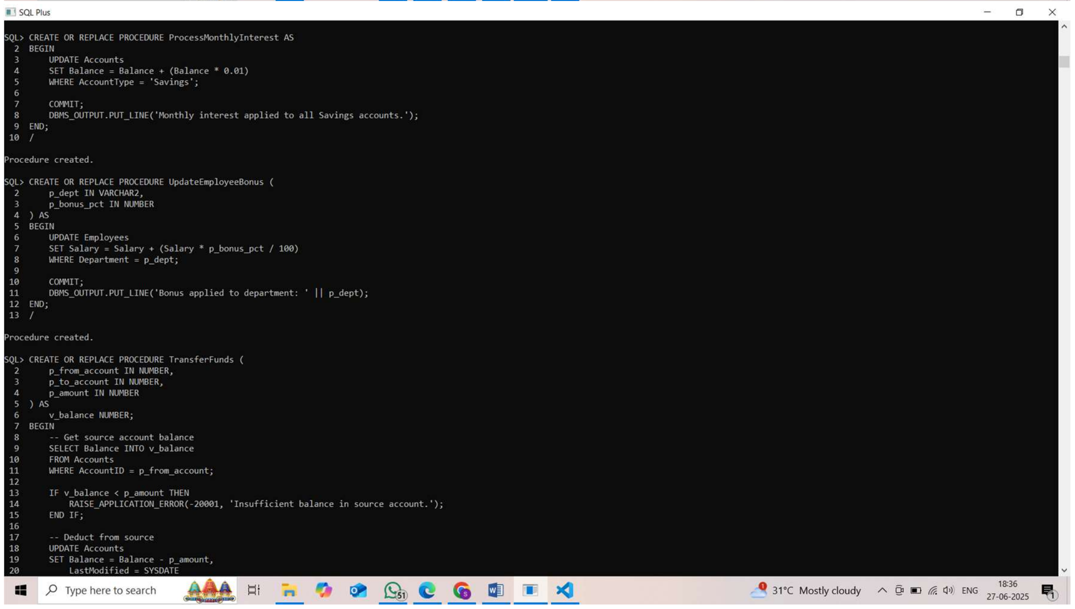
Scenario 1: Each month, the bank applies 1% interest to all savings accounts.
Question: Write a stored procedure named ProcessMonthlyInterest that finds all savings accounts and adds 1% of their current balance to the balance.

Scenario 2: Based on performance, employees in a department get a salary bonus.
Question: Write a stored procedure named UpdateEmployeeBonus that accepts a department name and a bonus percentage, and updates the salary of all employees in that department.

Scenario 3: Customers want to transfer funds between their own accounts.
Question: Write a stored procedure named TransferFunds that accepts from_account, to_account, and amount. It checks if from_account has sufficient balance, and if yes, deducts the amount from from_account and adds it to to_account. If not, it should print "Insufficient Balance".



```
SQL> CREATE OR REPLACE PROCEDURE TransferFunds (
    p_from_account IN NUMBER,
    p_to_account IN NUMBER,
    p_amount IN NUMBER
) AS
    v_balance NUMBER;
BEGIN
    -- Get source account balance
    SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = p_from_account;

    IF v_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in source account.');
    END IF;

    -- Deduct from source
    UPDATE Accounts
    SET Balance = Balance - p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_from_account;

    -- Add to destination
    UPDATE Accounts
    SET Balance = Balance + p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_to_account;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Transferred ' || p_amount || ' from Account ' || p_from_account || ' to ' || p_to_account);
END;
/

Procedure created.

SQL>
```



```
SQL> CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS
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;
/

Procedure created.

SQL> CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
    p_dept IN VARCHAR2,
    p_bonus_pct IN NUMBER
) AS
BEGIN
    UPDATE Employees
    SET Salary = Salary + (Salary * p_bonus_pct / 100)
    WHERE Department = p_dept;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Bonus applied to department: ' || p_dept);
END;
/

Procedure created.

SQL> CREATE OR REPLACE PROCEDURE TransferFunds (
    p_from_account IN NUMBER,
    p_to_account IN NUMBER,
    p_amount IN NUMBER
) AS
    v_balance NUMBER;
BEGIN
    -- Get source account balance
    SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = p_from_account;

    IF v_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in source account.');
    END IF;

    -- Deduct from source
    UPDATE Accounts
    SET Balance = Balance - p_amount,
        LastModified = SYSDATE
```