

SQL Exercises Solution Document (Superset ID: 6364376)

Advance Concepts

Exercise 1: Ranking and Window Functions

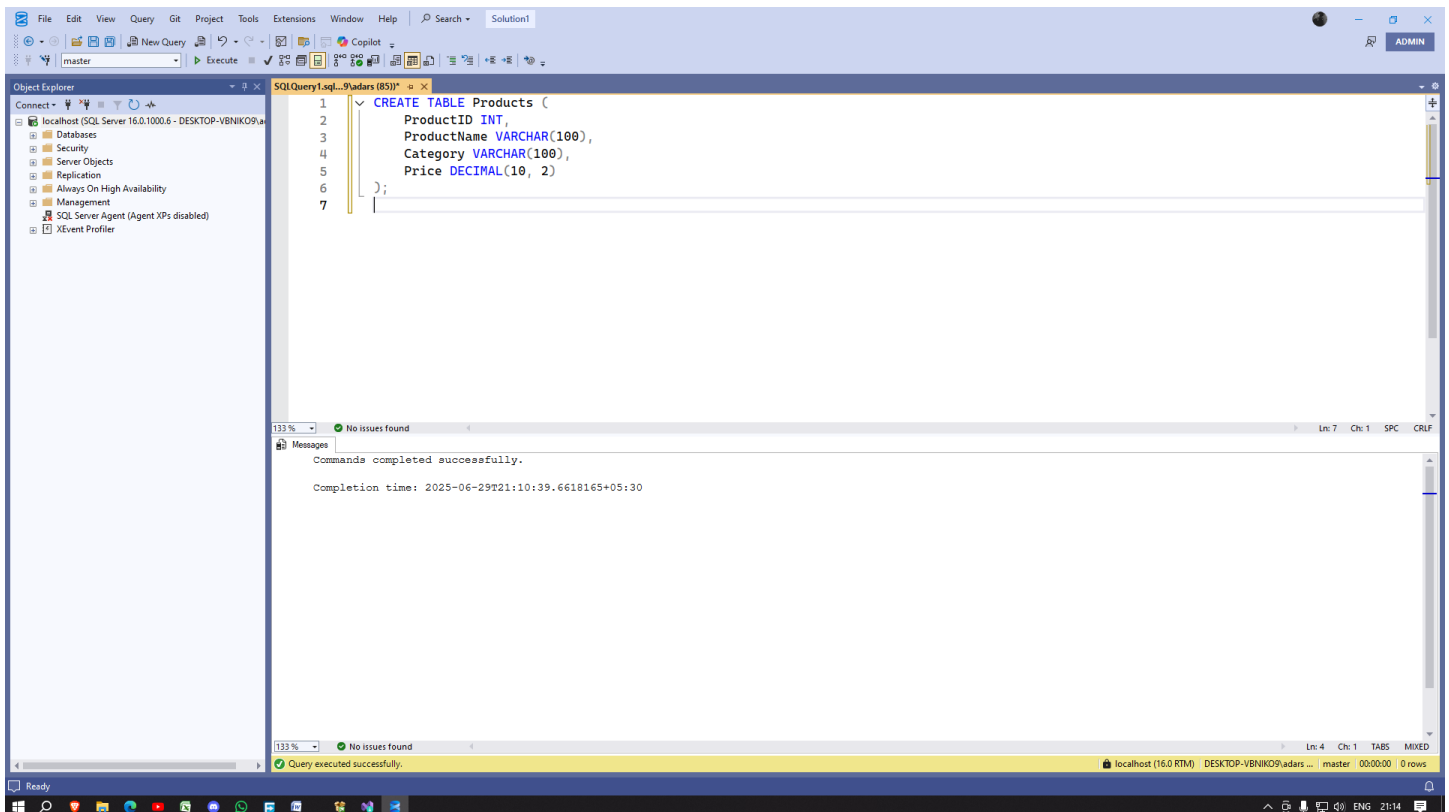
Objective

Demonstrate the use of different ranking window functions (ROW_NUMBER, RANK, DENSE_RANK) to rank products within categories by price.

Database Schema Setup

```
CREATE TABLE Products (  
    ProductID INT PRIMARY KEY,  
    ProductName VARCHAR(100),  
    Category VARCHAR(100),  
    Price DECIMAL(10, 2)  
);
```

Execution Status: ☐ Commands completed successfully

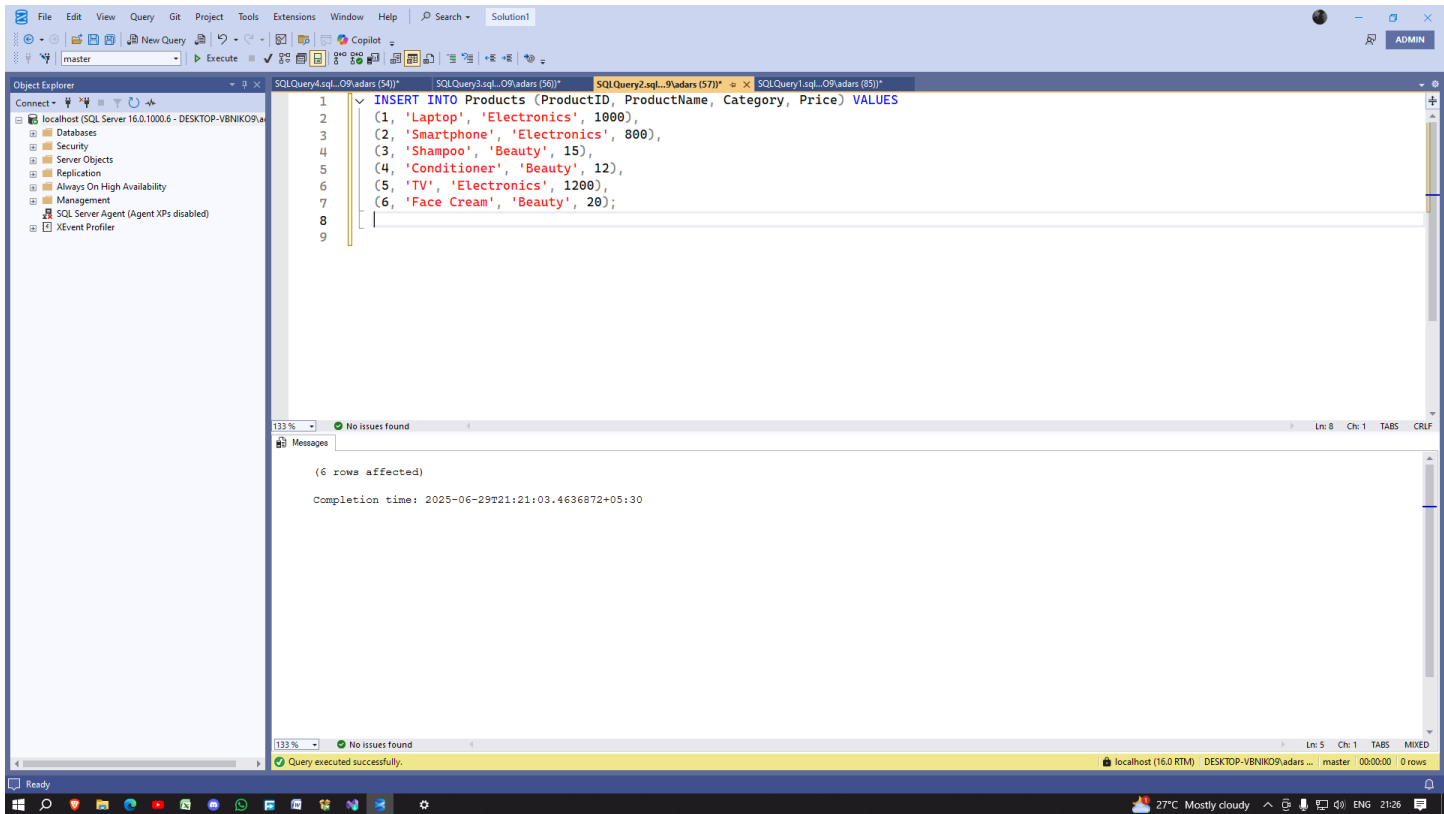


Sample Data Insertion

INSERT INTO Products (ProductID, ProductName, Category, Price) VALUES

- (1, 'Laptop', 'Electronics', 1000),
- (2, 'Smartphone', 'Electronics', 800),
- (3, 'Shampoo', 'Beauty', 15),
- (4, 'Conditioner', 'Beauty', 12),
- (5, 'TV', 'Electronics', 1200),
- (6, 'Face Cream', 'Beauty', 20);

Execution Status: ☐ (6 rows affected)

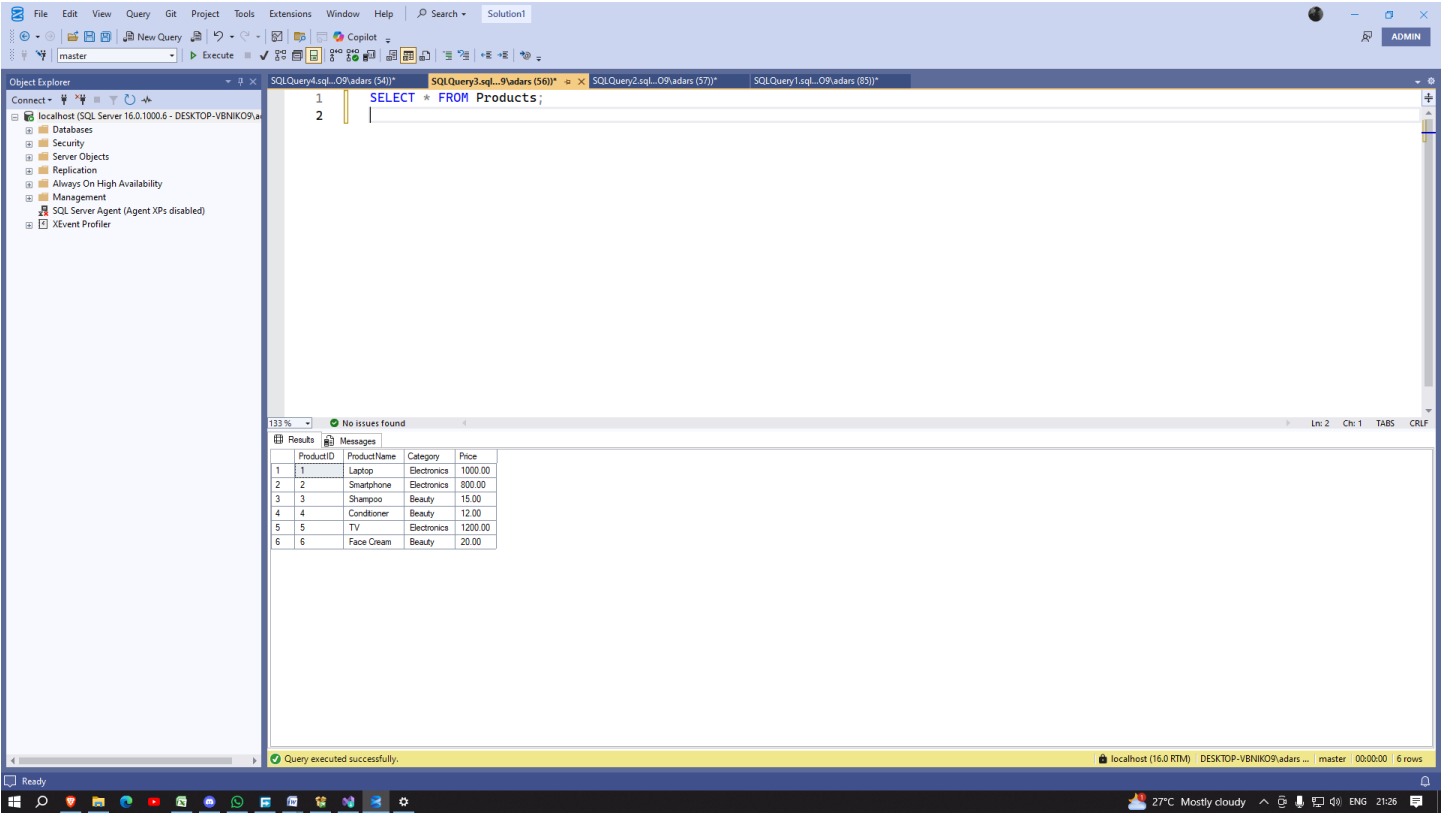


Data Verification Query

SELECT * FROM Products;

Execution Results:

ProductID	ProductName	Category	Price
1	Laptop	Electronics	1000.00
2	Smartphone	Electronics	800.00
3	Shampoo	Beauty	15.00
4	Conditioner	Beauty	12.00
5	TV	Electronics	1200.00
6	Face Cream	Beauty	20.00

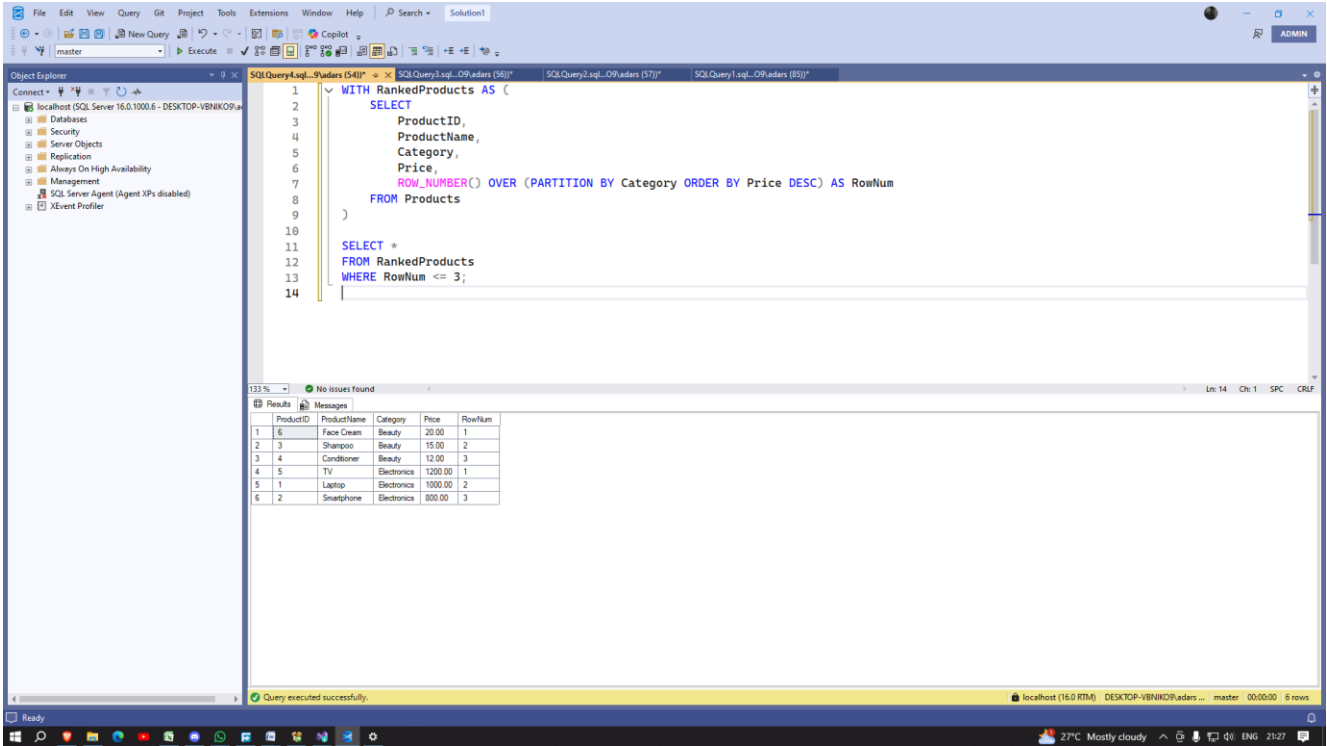


Solution 1: ROW_NUMBER() Function

```
WITH RankedProducts AS (  
    SELECT  
        ProductID,  
        ProductName,  
        Category,  
        Price,  
        ROW_NUMBER() OVER (PARTITION BY Category ORDER BY Price DESC) AS RowNum  
    FROM Products  
)  
SELECT *  
FROM RankedProducts  
WHERE RowNum <= 3;
```

Execution Results:

ProductID	ProductName	Category	Price	RowNum
6	Face Cream	Beauty	20.00	1
3	Shampoo	Beauty	15.00	2
4	Conditioner	Beauty	12.00	3
5	TV	Electronics	1200.00	1
1	Laptop	Electronics	1000.00	2
2	Smartphone	Electronics	800.00	3



Solution 2: RANK() Function

```
WITH RankedProducts AS (  
    SELECT  
        ProductID,  
        ProductName,  
        Category,  
        Price,  
        RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS PriceRank  
    FROM Products  
)  
SELECT *  
FROM RankedProducts  
WHERE PriceRank <= 3;
```

Execution Results:

ProductID	ProductName	Category	Price	PriceRank
6	Face Cream	Beauty	20.00	1
3	Shampoo	Beauty	15.00	2
4	Conditioner	Beauty	12.00	3
5	TV	Electronics	1200.00	1
1	Laptop	Electronics	1000.00	2
2	Smartphone	Electronics	800.00	3

The screenshot shows the SQL Server Enterprise Manager interface. The SQL Query Editor contains the following query:

```
1 WITH RankedProducts AS (  
2     SELECT  
3         ProductID,  
4         ProductName,  
5         Category,  
6         Price,  
7         RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS PriceRank  
8     FROM Products  
9 )  
10 SELECT *  
11 FROM RankedProducts  
12 WHERE PriceRank <= 3;  
13
```

The Results pane displays the following data:

	ProductID	ProductName	Category	Price	PriceRank
1	6	Face Cream	Beauty	20.00	1
2	3	Shampoo	Beauty	15.00	2
3	4	Conditioner	Beauty	12.00	3
4	5	TV	Electronics	1200.00	1
5	1	Laptop	Electronics	1000.00	2
6	2	Smartphone	Electronics	800.00	3

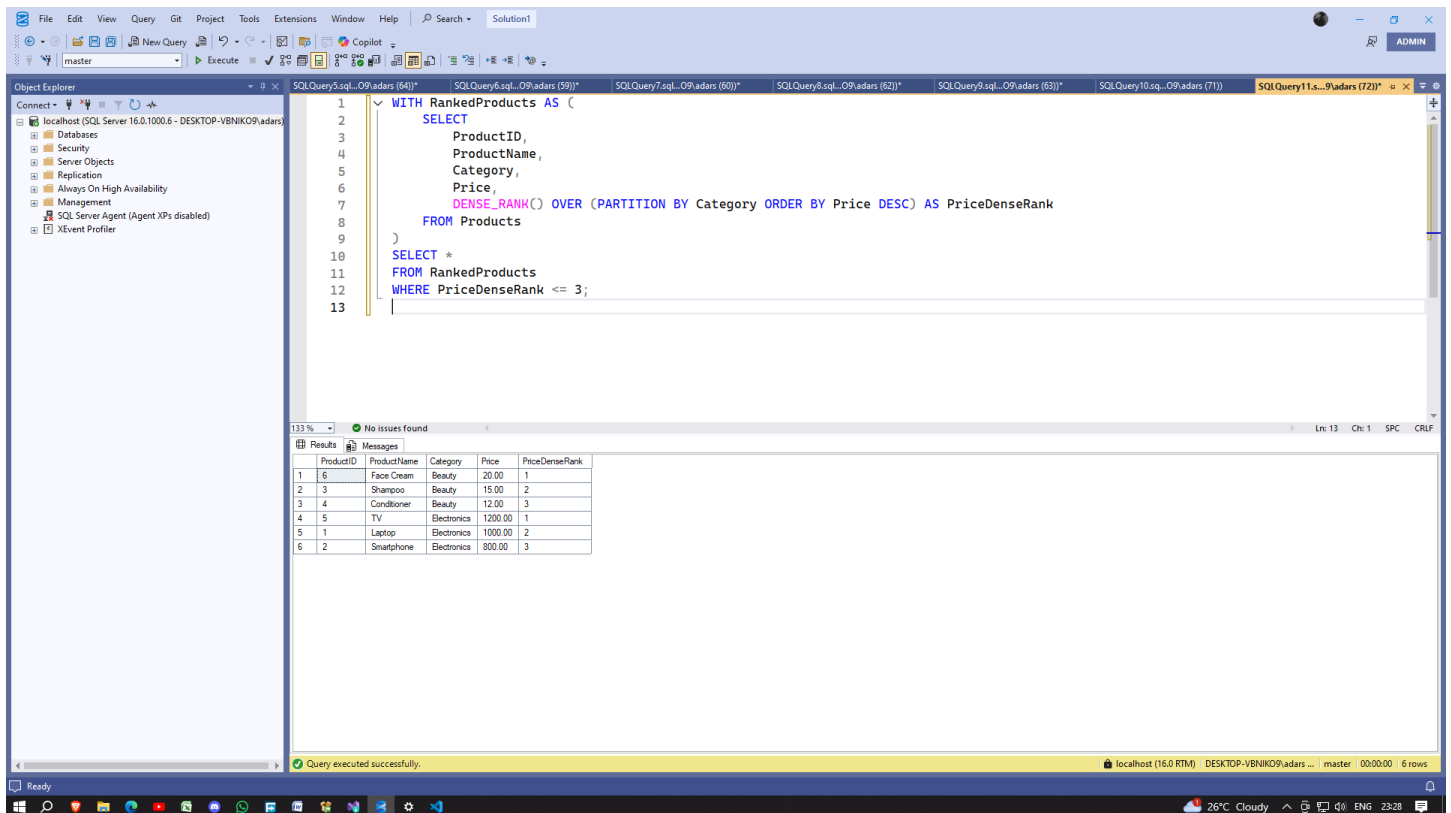
The status bar at the bottom indicates: "Query executed successfully. localhost (16.0 RTM) / DESKTOP-VBNK09adars -- master 00:00:00 6 rows".

Solution 3: DENSE_RANK() Function

```
WITH RankedProducts AS (  
    SELECT  
        ProductID,  
        ProductName,  
        Category,  
        Price,  
        DENSE_RANK() OVER (PARTITION BY Category ORDER BY Price DESC) AS PriceDenseRank  
    FROM Products  
)  
SELECT *  
FROM RankedProducts  
WHERE PriceDenseRank <= 3;
```

Execution Results:

Similar to RANK() results for this dataset since there are no ties.



The screenshot displays the SQL Server Enterprise Manager interface. The central pane shows a SQL query being executed. The query uses a Common Table Expression (CTE) named 'RankedProducts' to calculate the DENSE_RANK of products within each category, ordered by price in descending order. The main query then selects all columns from 'RankedProducts' where the 'PriceDenseRank' is less than or equal to 3.

The results pane at the bottom shows the output of the query, which is a table with 6 rows and 5 columns: ProductID, ProductName, Category, Price, and PriceDenseRank. The results are as follows:

	ProductID	ProductName	Category	Price	PriceDenseRank
1	6	Face Cream	Beauty	20.00	1
2	3	Shampoo	Beauty	15.00	2
3	4	Conditioner	Beauty	12.00	3
4	5	TV	Electronics	1200.00	1
5	1	Laptop	Electronics	1000.00	2
6	2	Smartphone	Electronics	800.00	3

The status bar at the bottom indicates that the query was executed successfully, returning 6 rows.

Stored Procedure

Exercise 1: Employee Management System - Stored Procedures

Objective

Create and implement stored procedures for employee management operations including data retrieval and insertion.

Database Schema Setup

```
CREATE TABLE Departments (  
    DepartmentID INT PRIMARY KEY,  
    DepartmentName VARCHAR(100)  
);  
  
CREATE TABLE Employees (  
    EmployeeID INT IDENTITY(1,1) PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),  
    Salary DECIMAL(10,2),  
    JoinDate DATE  
);
```

Execution Status: ☒ Commands completed successfully

Master Data Setup

```
INSERT INTO Departments (DepartmentID, DepartmentName) VALUES  
(1, 'HR'),  
(2, 'Finance'),  
(3, 'IT'),  
(4, 'Marketing');  
  
INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate) VALUES  
(John, Doe, 1, 5000.00, '2020-01-15'),  
(Jane, Smith, 2, 6000.00, '2019-03-22'),  
(Michael, Johnson, 3, 7000.00, '2018-07-30'),  
(Emily, Davis, 4, 5500.00, '2021-11-05');
```

File Edit View Query Git Project Tools Extensions Window Help Search Solution1

master

Object Explorer

localhost (SQL Server 16.0.1000.6 - DESKTOP-VBNIKO9\adars)

Databases

Security

Server Objects

Replication

Always On High Availability

Management

SQL Server Agent (Agent XPs disabled)

XEvent Profiler

```
1 CREATE TABLE Departments (  
2     DepartmentID INT PRIMARY KEY,  
3     DepartmentName VARCHAR(100)  
4 );  
5 CREATE TABLE Employees (  
6     EmployeeID INT IDENTITY(1,1) PRIMARY KEY, -- Auto-increment enabled  
7     FirstName VARCHAR(50),  
8     LastName VARCHAR(50),  
9     DepartmentID INT FOREIGN KEY REFERENCES Departments(DepartmentID),  
10    Salary DECIMAL(10,2),  
11    JoinDate DATE  
12 );  
13 INSERT INTO Departments (DepartmentID, DepartmentName) VALUES  
14 (1, 'HR'),  
15 (2, 'Finance'),  
16 (3, 'IT'),  
17 (4, 'Marketing');  
18 INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate) VALUES  
19 ('John', 'Doe', 1, 5000.00, '2020-01-15'),  
20 ('Jane', 'Smith', 2, 6000.00, '2019-03-22'),  
21 ('Michael', 'Johnson', 3, 7000.00, '2018-07-30'),  
22 ('Emily', 'Davis', 4, 5500.00, '2021-11-05');  
23
```

133% No issues found

Messages

(4 rows affected)

(4 rows affected)

Completion time: 2025-06-29T21:44:25.8852708+05:30

133% No issues found

Query executed successfully.

localhost (16.0 RTM) DESKTOP-VBNIKO9\adars ... master 00:00:00 0 rows

Ready

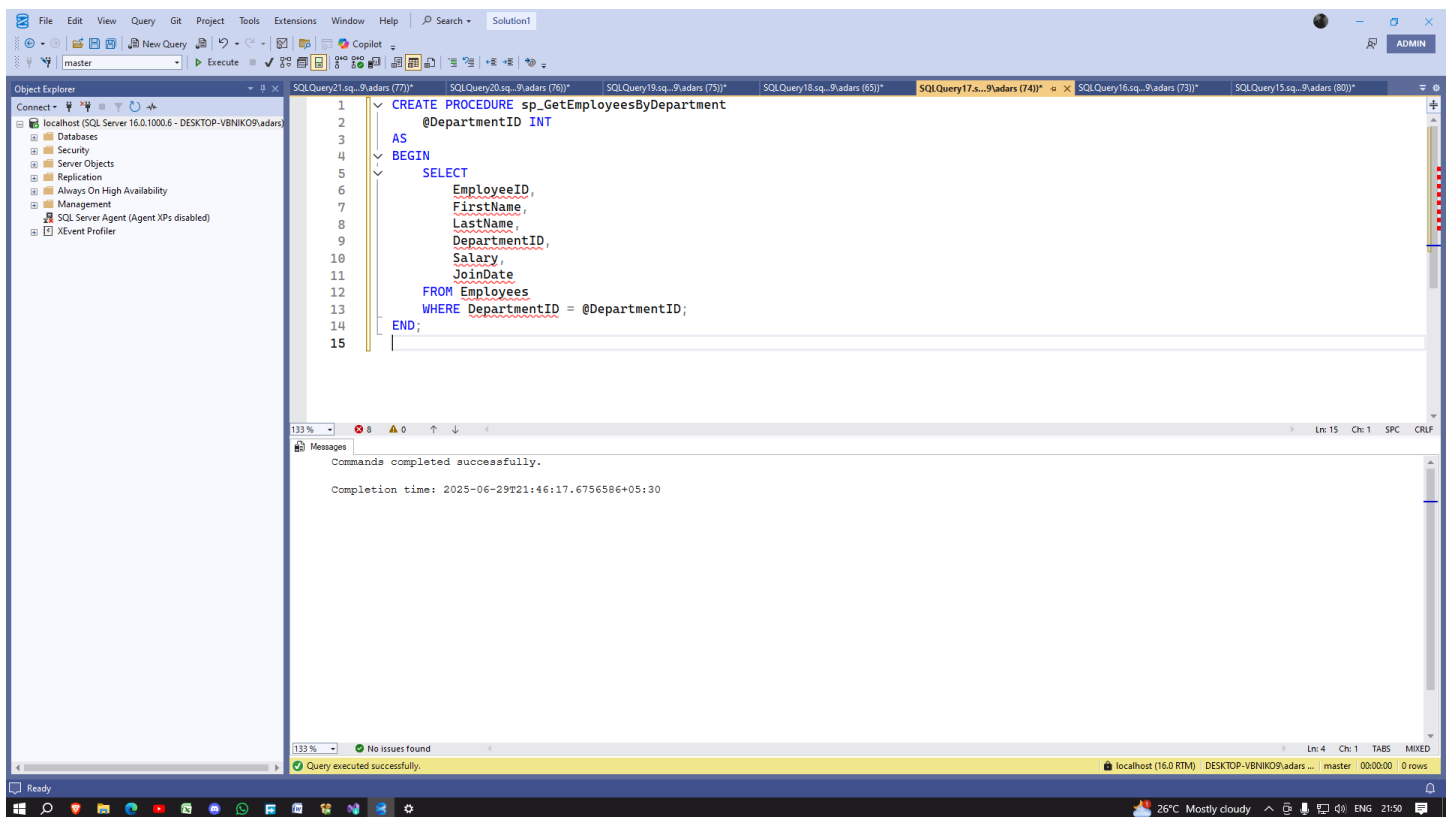
26°C Mostly cloudy

ENG 21:50

Solution 1: Stored Procedure for Employee Retrieval

```
CREATE PROCEDURE sp_GetEmployeesByDepartment
    @DepartmentID INT
AS
BEGIN
    SELECT
        EmployeeID,
        FirstName,
        LastName,
        DepartmentID,
        Salary,
        JoinDate
    FROM Employees
    WHERE DepartmentID = @DepartmentID;
END;
```

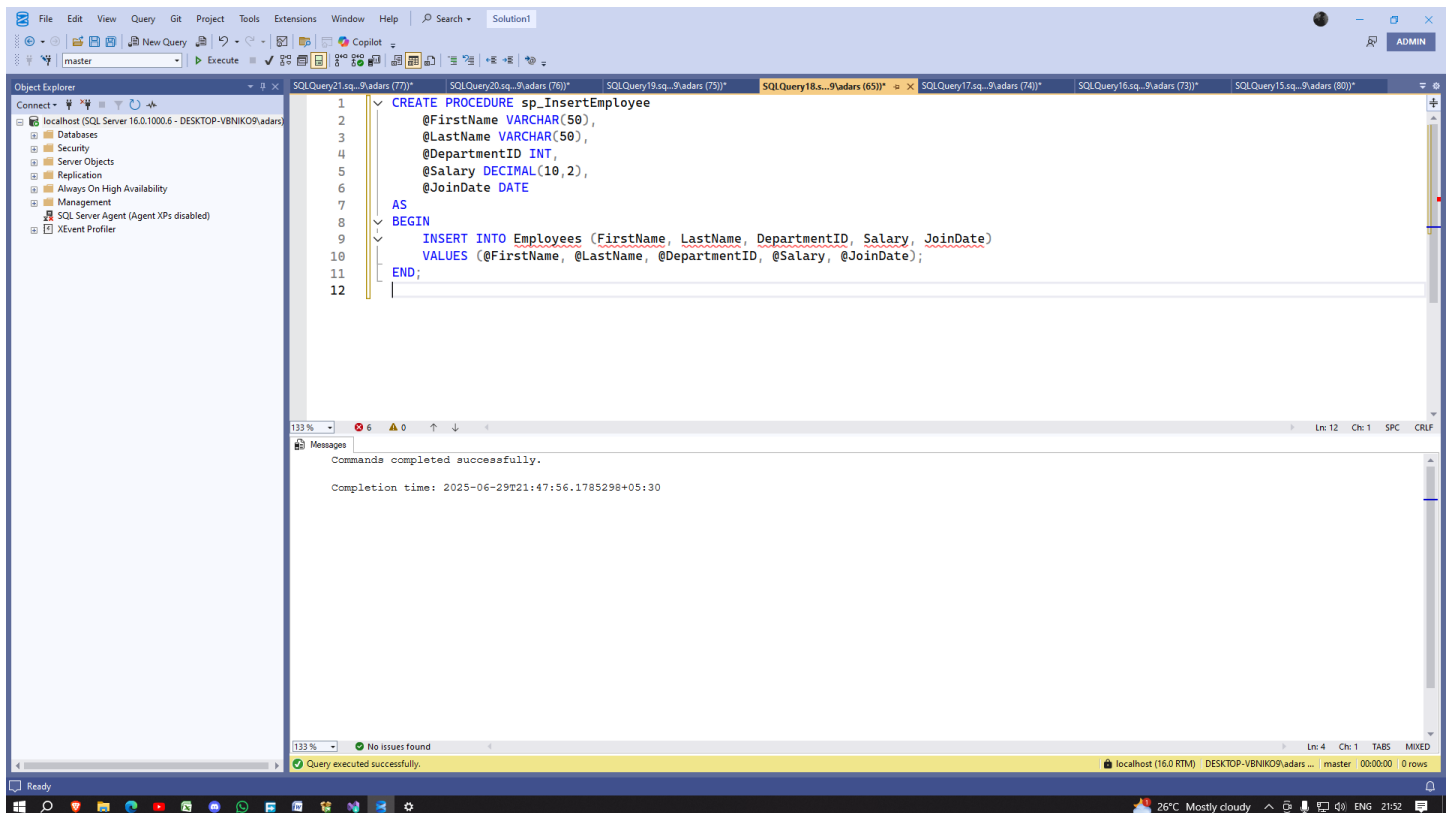
Execution Status: ☐ Commands completed successfully



Solution 2: Stored Procedure for Employee Insertion

```
CREATE PROCEDURE sp_InsertEmployee
    @FirstName VARCHAR(50),
    @LastName VARCHAR(50),
    @DepartmentID INT,
    @Salary DECIMAL(10,2),
    @JoinDate DATE
AS
BEGIN
    INSERT INTO Employees (FirstName, LastName, DepartmentID, Salary, JoinDate)
    VALUES (@FirstName, @LastName, @DepartmentID, @Salary, @JoinDate);
END;
```

Execution Status: ☐ Commands completed successfully

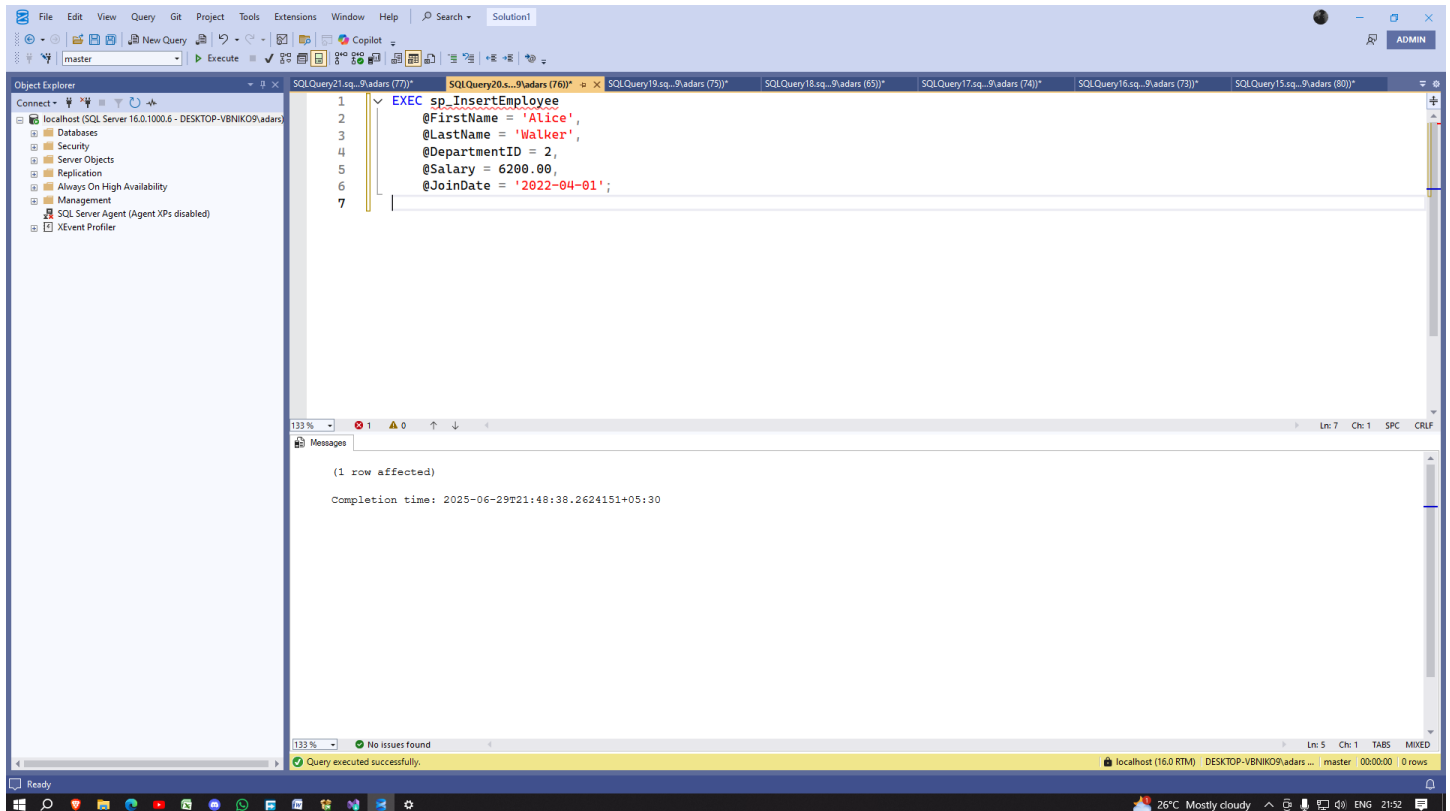


Testing Stored Procedure Execution

EXEC sp_InsertEmployee

```
@FirstName = 'Alice',  
@LastName = 'Walker',  
@DepartmentID = 2,  
@Salary = 6200.00,  
@JoinDate = '2022-04-01';
```

Execution Status: ☐ Commands completed successfully

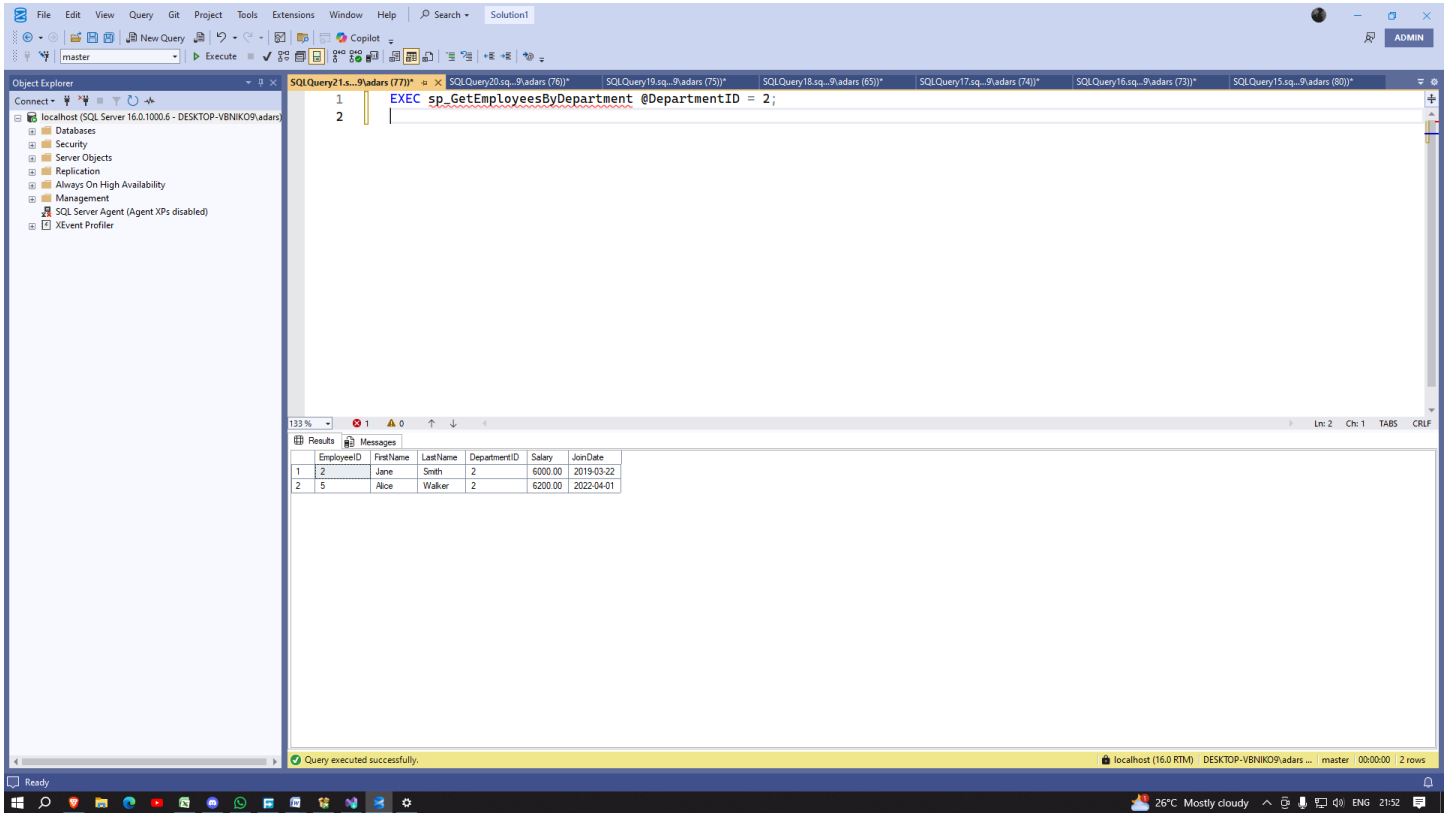


Verifying Data Retrieval

```
EXEC sp_GetEmployeesByDepartment @DepartmentID = 2;
```

Execution Results:

EmployeeID	FirstName	LastName	DepartmentID	Salary	JoinDate
2	Jane	Smith	2	6000.00	2019-03-22
5	Alice	Walker	2	6200.00	2022-04-01



Exercise 5: Return Data from Stored Procedure

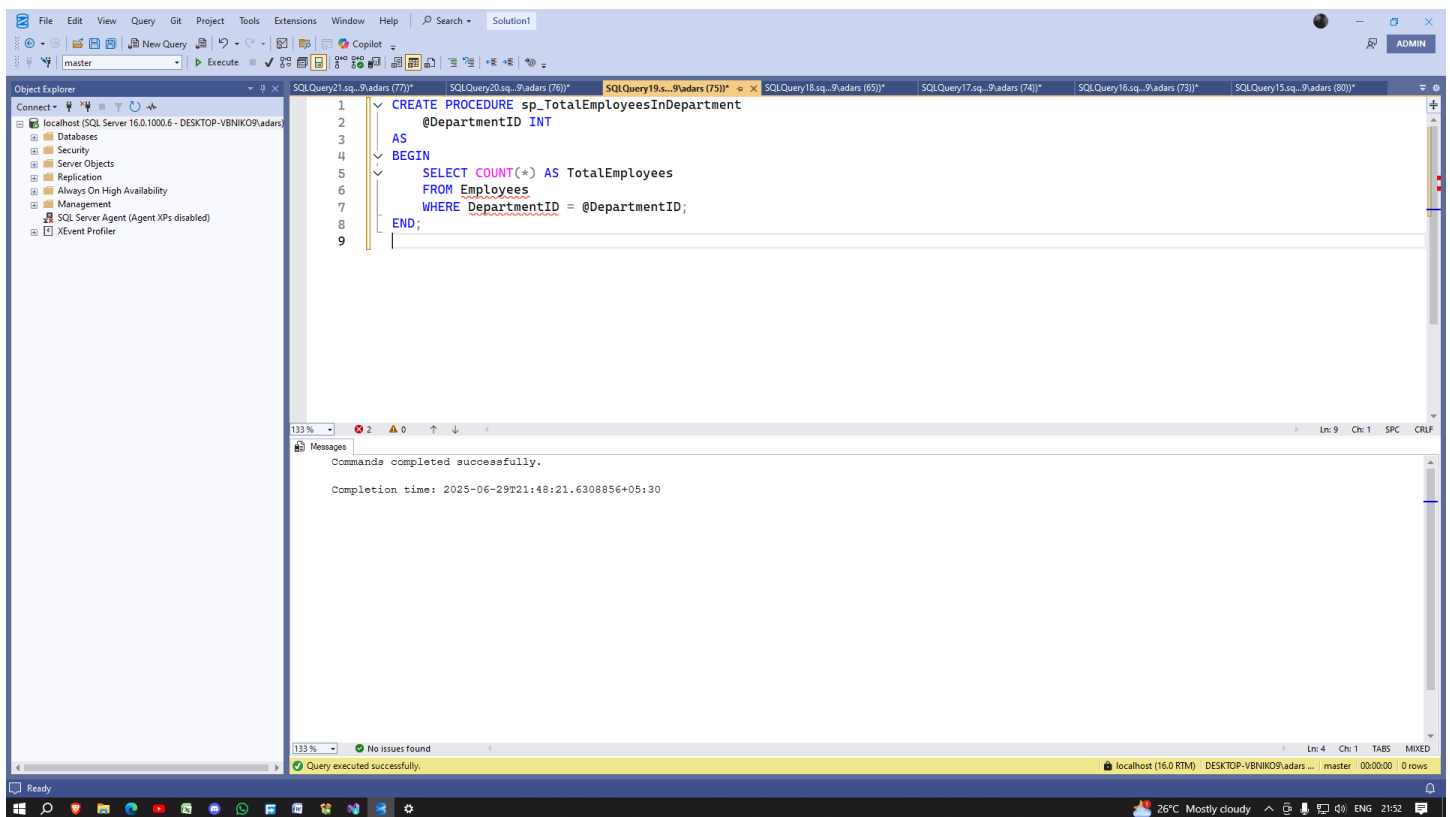
Objective

Create a stored procedure that returns aggregated data (count of employees in a department).

Solution: Employee Counting Procedure

```
CREATE PROCEDURE sp_TotalEmployeesInDepartment
    @DepartmentID INT
AS
BEGIN
    SELECT COUNT(*) AS TotalEmployees
    FROM Employees
    WHERE DepartmentID = @DepartmentID;
END;
```

Execution Status: ☐ Commands completed successfully



Testing the Counting Procedure

```
EXEC sp_TotalEmployeesInDepartment @DepartmentID = 2;
```

Execution Results:

TotalEmployees
2

