# **SQL CODING CHALLENGE**

**Name:Tejaswini Gokanakonda**

## **Roll no**:**DE142**

**Date:08-11-2024**

**Coding Challenge-1**

**CODE:**

**DTA BASE CREATION:**

CREATE DATABASE Companydatabase;

USE Companydatabase;

**--Employees Table with IDENTITY(1,1)**

CREATE TABLE Employees (

EmployeeID INT IDENTITY(1,1) PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Age INT,

Department VARCHAR(50)

);

**--Insert Data into Employees Table (without specifying EmployeeID)**

INSERT INTO Employees (FirstName, LastName, Age, Department)

VALUES

('Amit', 'Sharma', 30, 'HR'),

('Priya', 'Verma', 25, 'Engineering'),

('Ravi', 'Patel', 40, 'Marketing'),

('Neha', 'Reddy', 35, 'Engineering'),

('Suresh', 'Kumar', 50, 'HR');

**-- Departments Table**

CREATE TABLE Departments (

DepartmentName VARCHAR(50) PRIMARY KEY,

Manager VARCHAR(50)

);

**-- Insert Data into Departments Table**

INSERT INTO Departments (DepartmentName, Manager)

VALUES

('HR', 'Anil Kumar'),

('Engineering', 'Rajesh Gupta'),

('Marketing', 'Nisha Mehta'),

('Sales', 'Sunil Desai');

#### **Queries**

#### **Question 1: Querying Data Using Joins and Subqueries, with a focus on Subtotals**

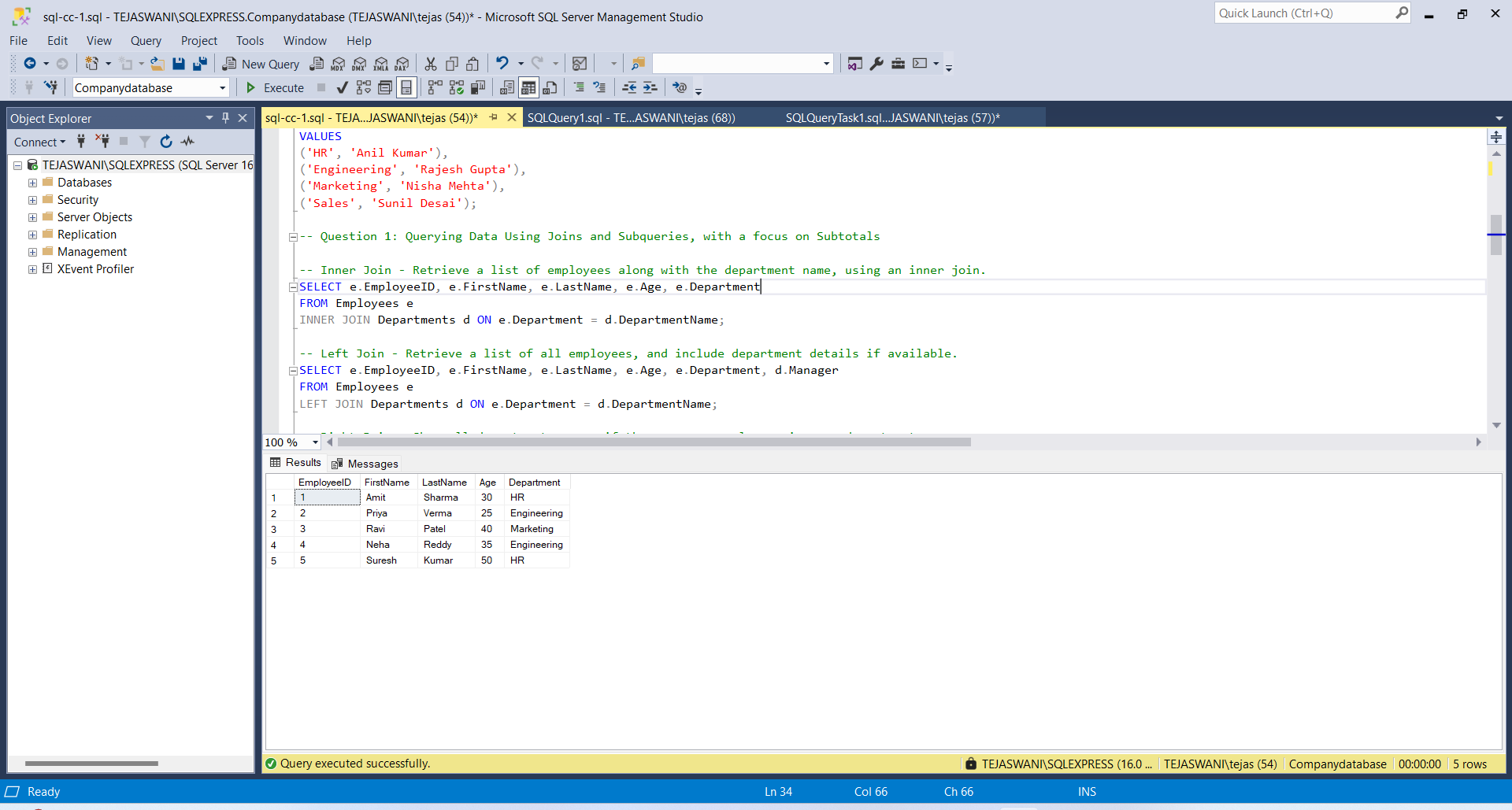
**1.Inner Join** - Retrieve a list of employees along with the department name, using an inner join.

SELECT e.EmployeeID, e.FirstName, e.LastName, e.Age, e.Department FROM Employees e INNER JOIN Departments d ON e.Department = d.DepartmentName;

This query is an Inner Join operation between the Employees and Departments tables. Here's a brief explanation:

* It retrieves employee details (EmployeeID, FirstName, LastName, Age, Department) from the Employees table.
* The INNER JOIN ensures that only employees with matching departments in the Departments table are included in the result.
* The join condition is based on matching the Department column from Employees with the DepartmentName column from Departments.

This query effectively lists all employees along with their department information, but only for employees whose departments exist in the Departments table.



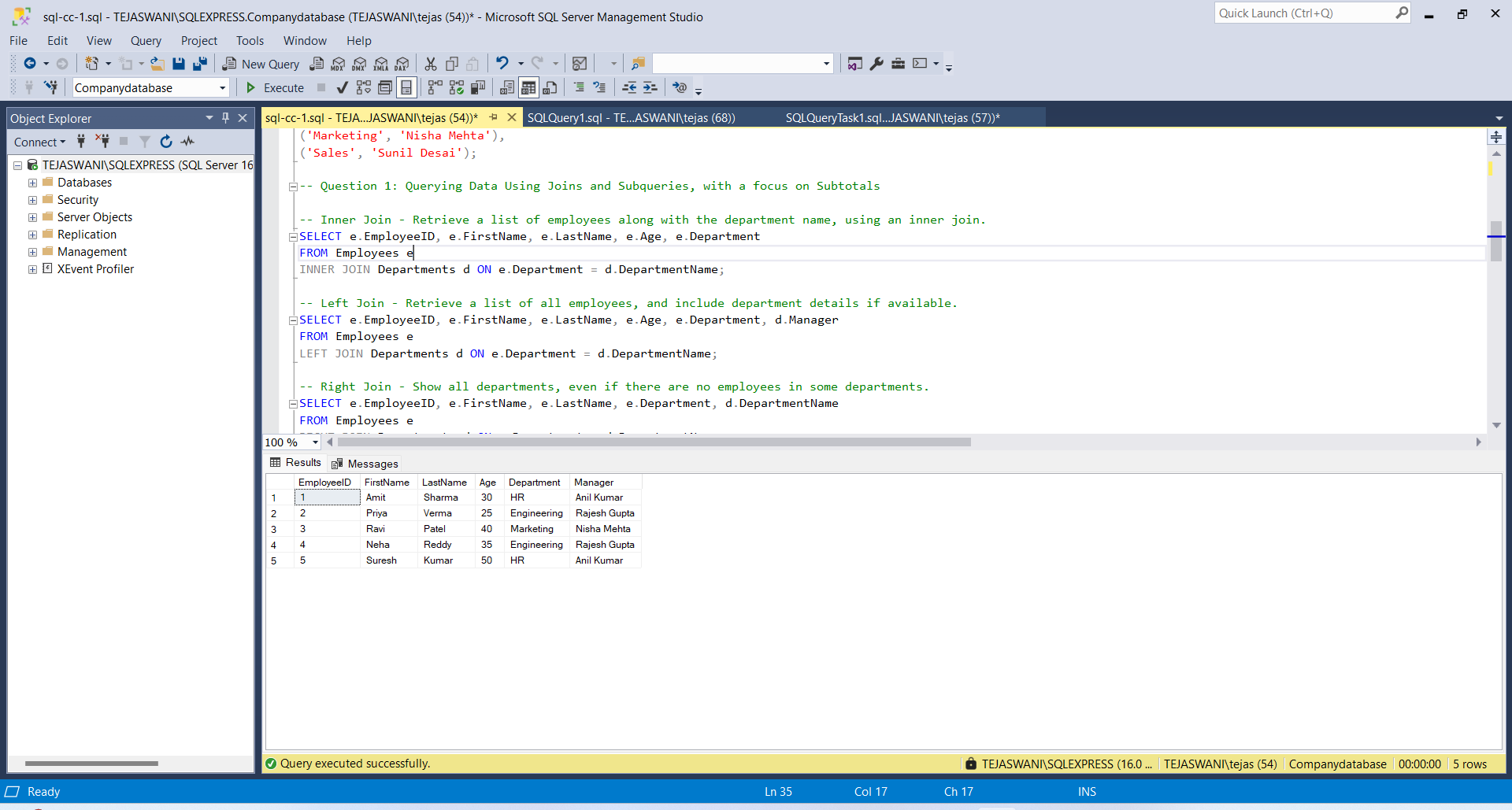
**2.Left Join** - Retrieve a list of all employees, and include department details if available.

SELECT e.EmployeeID, e.FirstName, e.LastName, e.Age, e.Department, d.Manager FROM Employees e LEFT JOIN Departments d ON e.Department = d.DepartmentName;

​

* This query is a Left Join operation between the Employees and Departments tables. Here's a brief explanation:
* It retrieves all employees (EmployeeID, FirstName, LastName, Age, Department) from the Employees table.
* It includes the Manager information from the Departments table if available.
* The LEFT JOIN ensures that all employees are included in the result, even if they don't have a matching department.
* If an employee's department doesn't exist in the Departments table, the Manager field will be NULL for that employee.

This query is useful for getting a complete list of employees along with their department details, including cases where an employee might not be assigned to a known department.



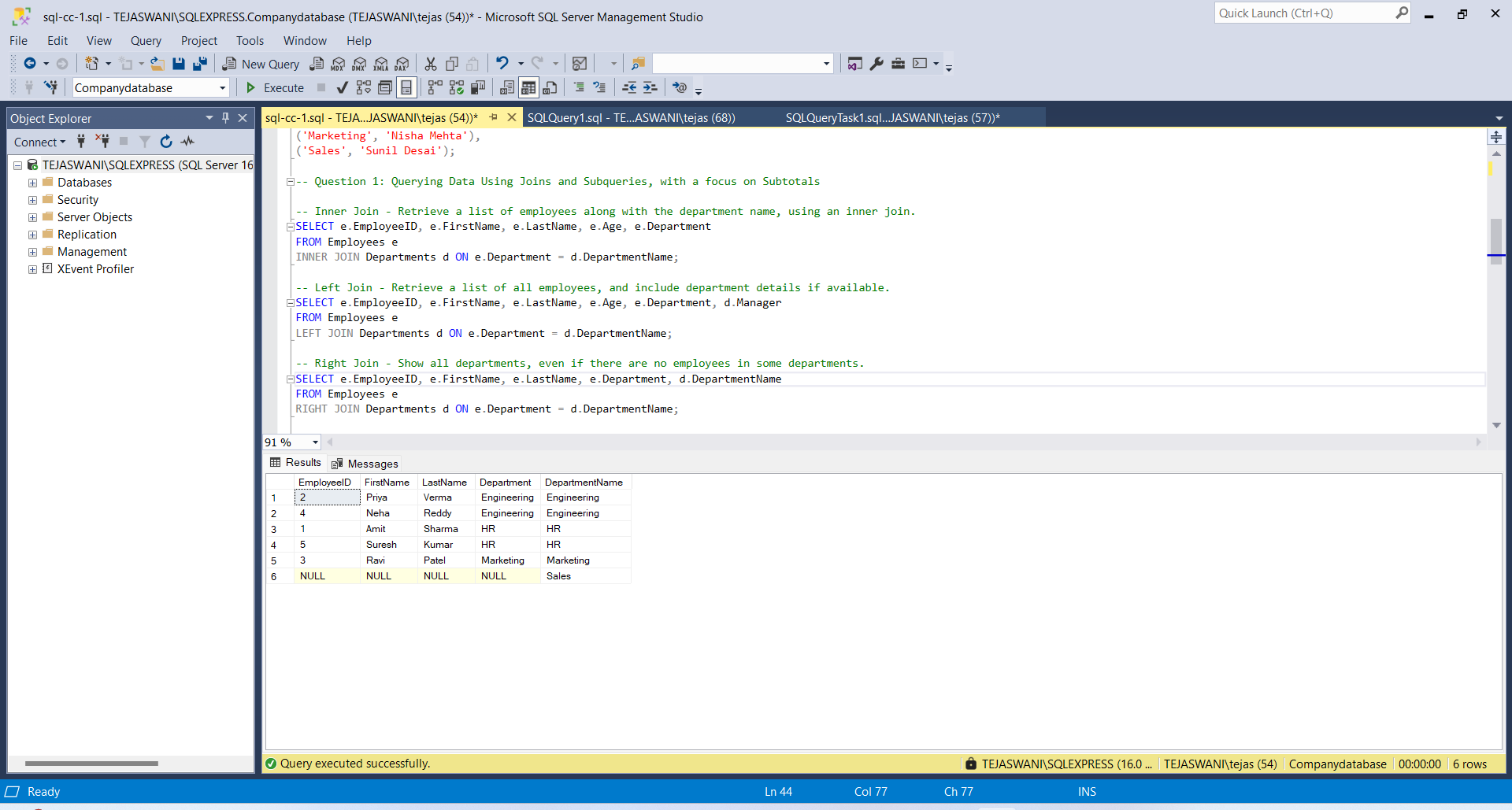
**3.Right Join** - Show all departments, even if there are no employees in some departments.

SELECT e.EmployeeID, e.FirstName, e.LastName, e.Department, d.DepartmentName FROM Employees e RIGHT JOIN Departments d ON e.Department = d.DepartmentName;

This SQL query uses a RIGHT JOIN to show all departments, even if there are no employees in some departments. Here's a brief explanation:

* It selects employee details (EmployeeID, FirstName, LastName, Department) and the department name from the Departments table.
* The RIGHT JOIN ensures that all departments from the Departments table are included in the result, even if there are no matching employees.
* If a department has no employees, the employee fields will be NULL in the result.

This query is useful for identifying departments that may not have any assigned employees, providing a complete view of all departments regardless of their staffing status.



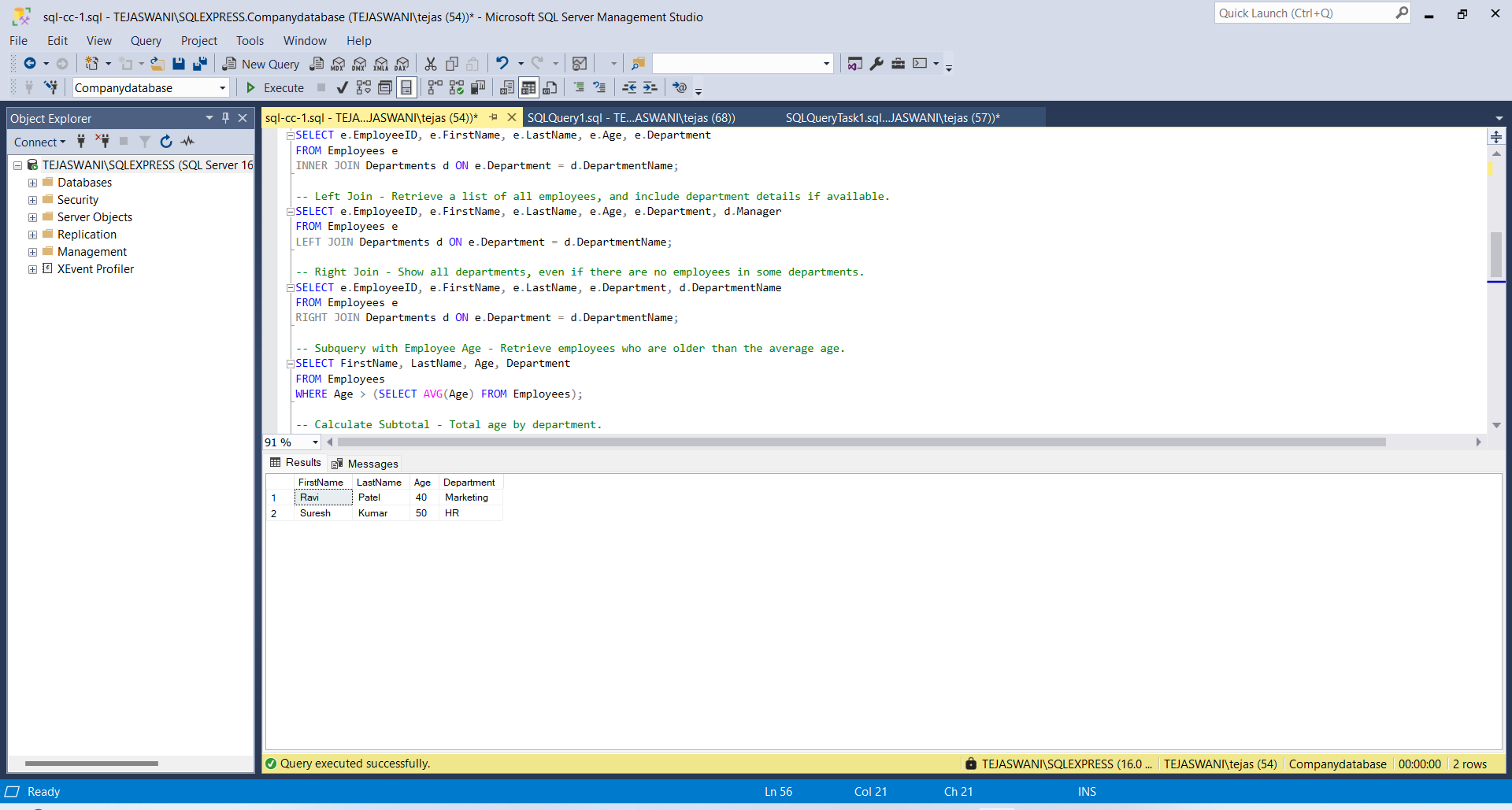
**4.Subquery with Employee Age** - Retrieve employees who are older than the average age.

SELECT FirstName, LastName, Age, Department FROM Employees WHERE Age > (SELECT AVG(Age) FROM Employees);

This query is designed to retrieve employees who are older than the average age of all employees in the company. Here's a brief explanation:

* It selects the FirstName, LastName, Age, and Department of employees.
* The WHERE clause filters employees whose Age is greater than the average age.
* The subquery (SELECT AVG(Age) FROM Employees) calculates the average age of all employees.

This query helps identify employees who are above the company's average age, which could be useful for various HR analyses and decision-making processes.



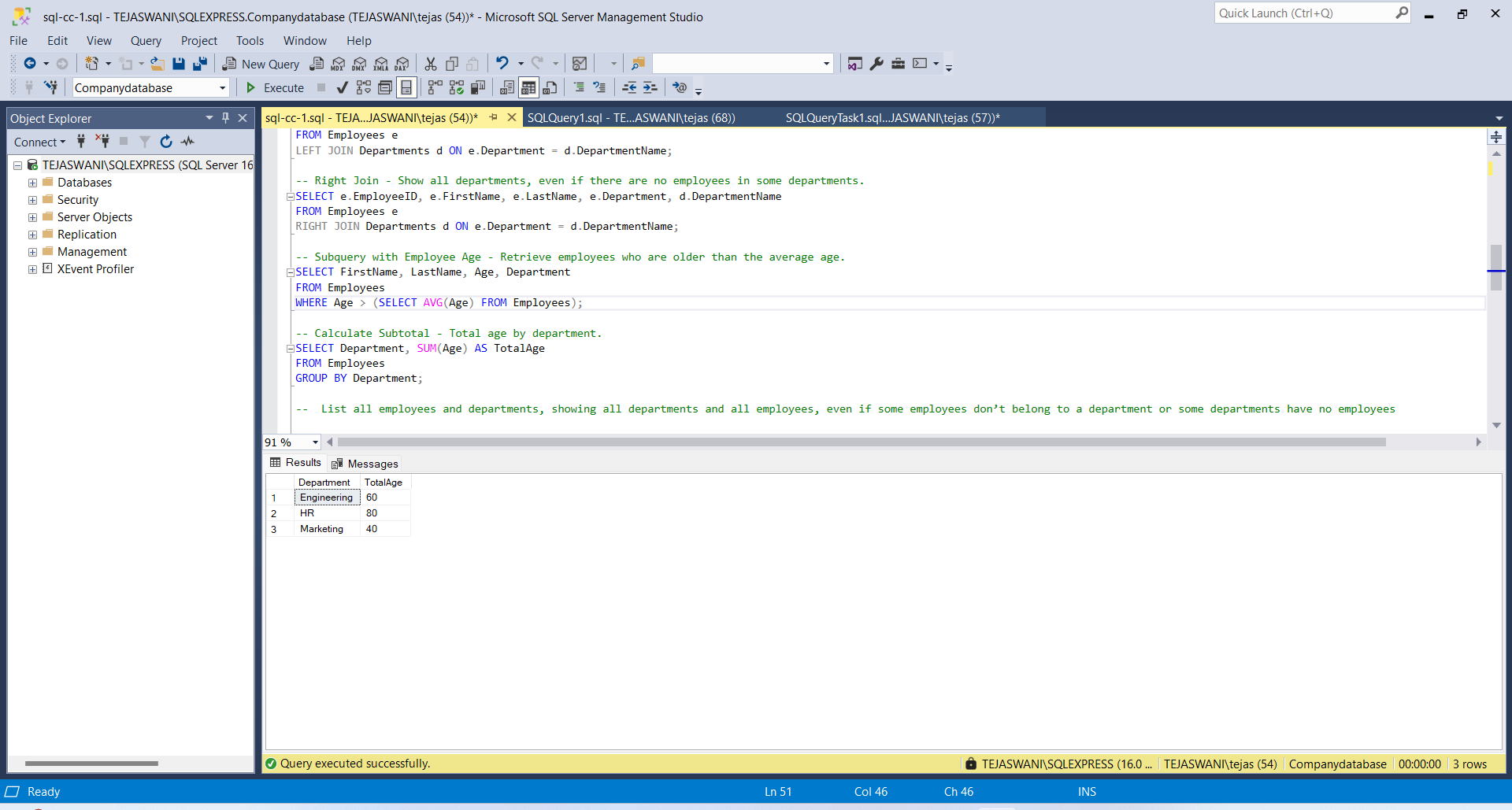
**5.Calculate Subtotal** - Total age by department.

SELECT Department, SUM(Age) AS TotalAge FROM Employees GROUP BY Department;

This SQL query calculates the total age of employees for each department. Here's a brief explanation:

* It selects the Department column and calculates the sum of Age, aliasing it as TotalAge.
* The SUM function is used to add up all the ages of employees in each department.
* The GROUP BY clause groups the results by Department, allowing for a total age calculation per department.

This query provides insights into the age distribution across different departments, which can be useful for various HR and management analyses.



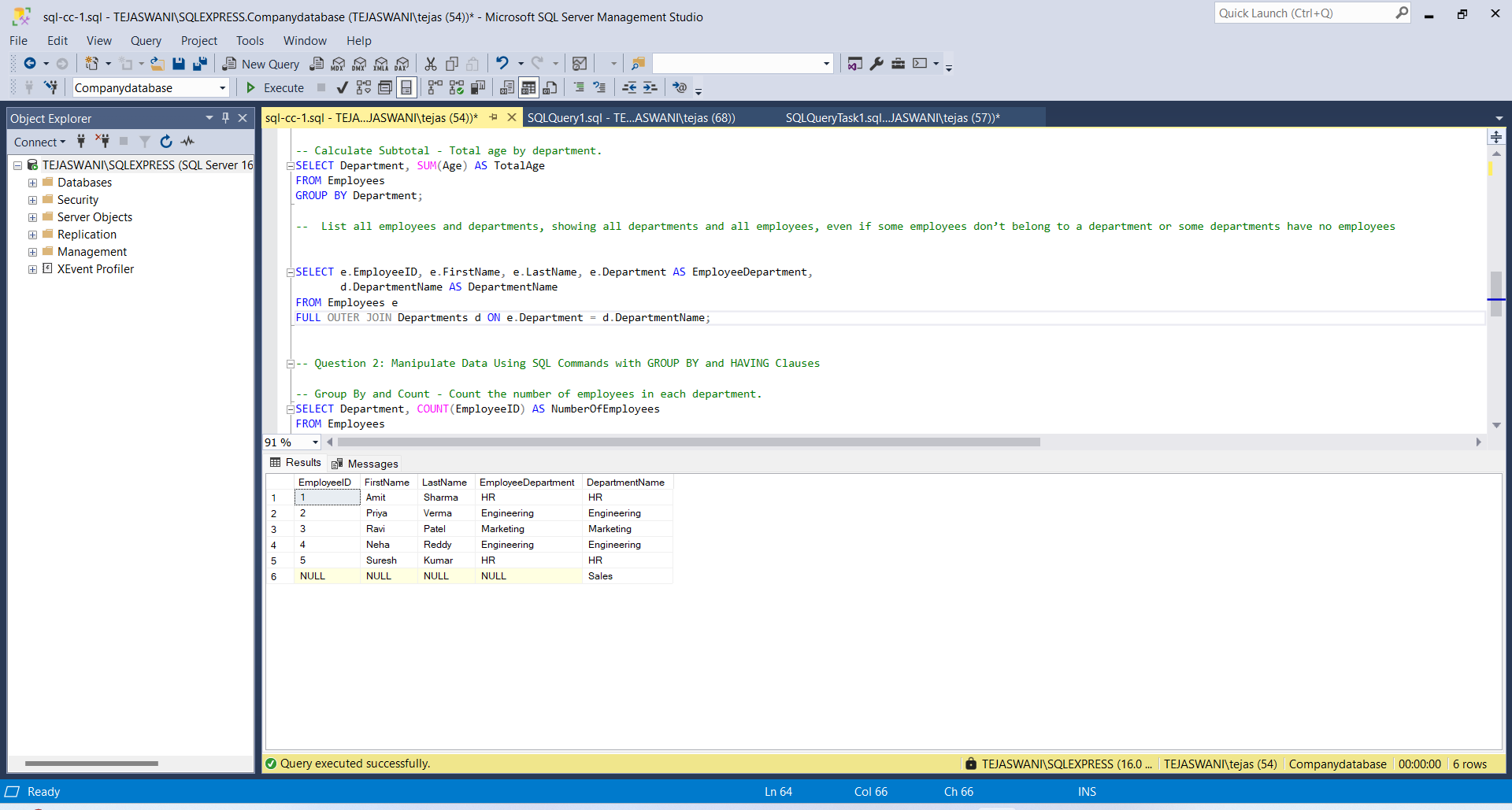
**6.Full Outer Join** - List all employees and departments, showing all departments and all employees, even if some employees don’t belong to a department or some departments have no employees

SELECT e.EmployeeID, e.FirstName, e.LastName, e.Department AS EmployeeDepartment, d.DepartmentName AS DepartmentName FROM Employees e FULL OUTER JOIN Departments d ON e.Department = d.DepartmentName;

The SQL query you've selected is a Full Outer Join between the Employees and Departments tables. Here's a brief explanation:

* It retrieves all employees and all departments, regardless of whether there's a match between them.
* The query will show employees even if they don't belong to a department, and departments even if they have no employees.
* It selects employee details (ID, FirstName, LastName) and their department, as well as the department name from the Departments table.
* The FULL OUTER JOIN ensures that all records from both tables are included in the result, providing a comprehensive view of the relationship between employees and departments.

This type of join is useful for identifying mismatches or gaps in the data, such as departments without employees or employees not assigned to any department.



#### **Question 2: Manipulate Data Using SQL Commands with GROUP BY and HAVING Clauses**

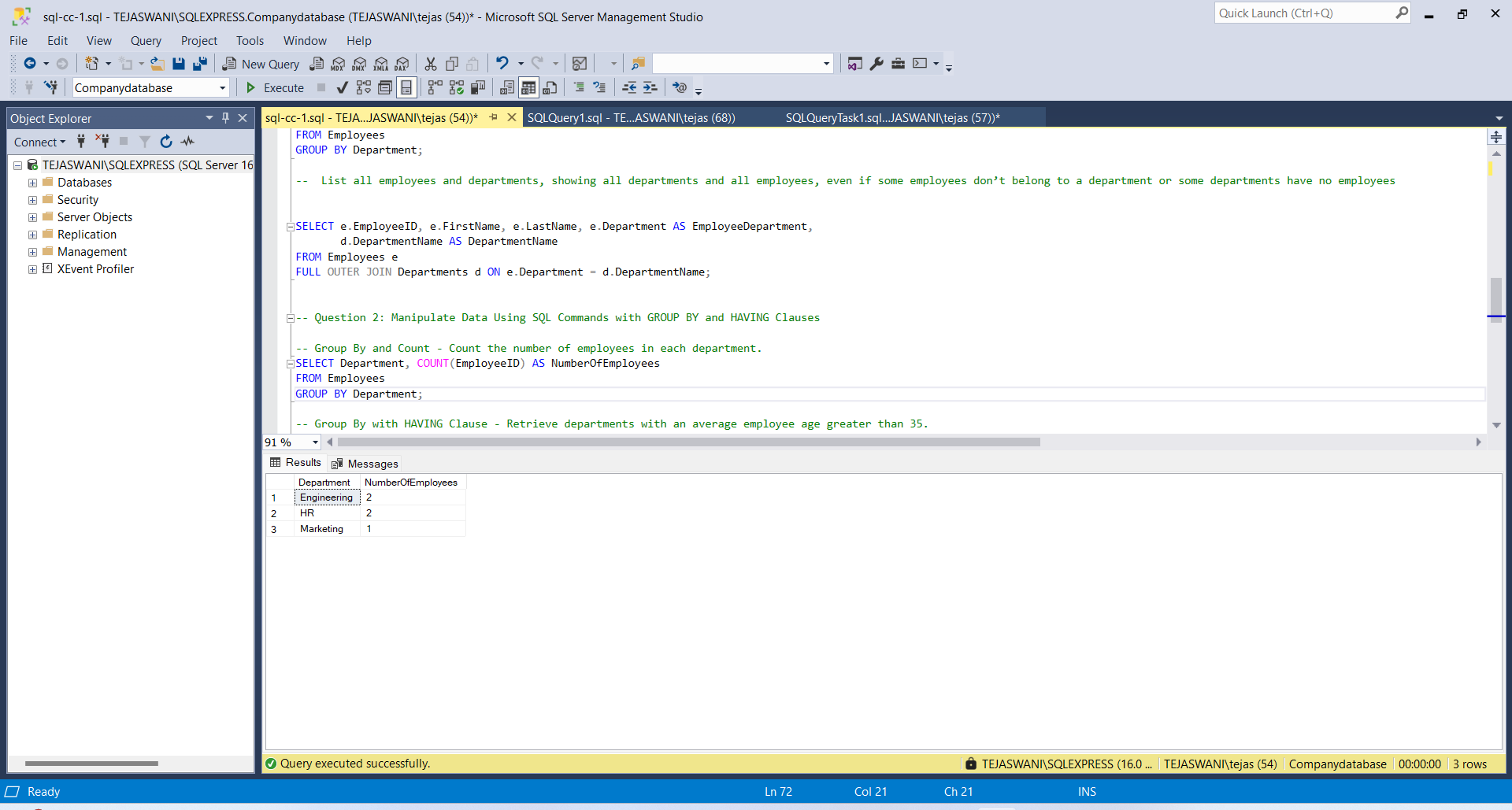
**1.Group By and Count** - Count the number of employees in each department.

SELECT Department, COUNT(EmployeeID) AS NumberOfEmployees FROM Employees GROUP BY Department;

This SQL query is designed to count the number of employees in each department. Here's a brief explanation:

* It selects the Department column and counts the number of EmployeeIDs, aliasing the count as NumberOfEmployees.
* The COUNT function is used to tally the number of employees.
* The GROUP BY clause groups the results by Department, allowing for a count per department.

This query provides a quick overview of the size of each department within the company, which can be useful for various HR and management purposes.



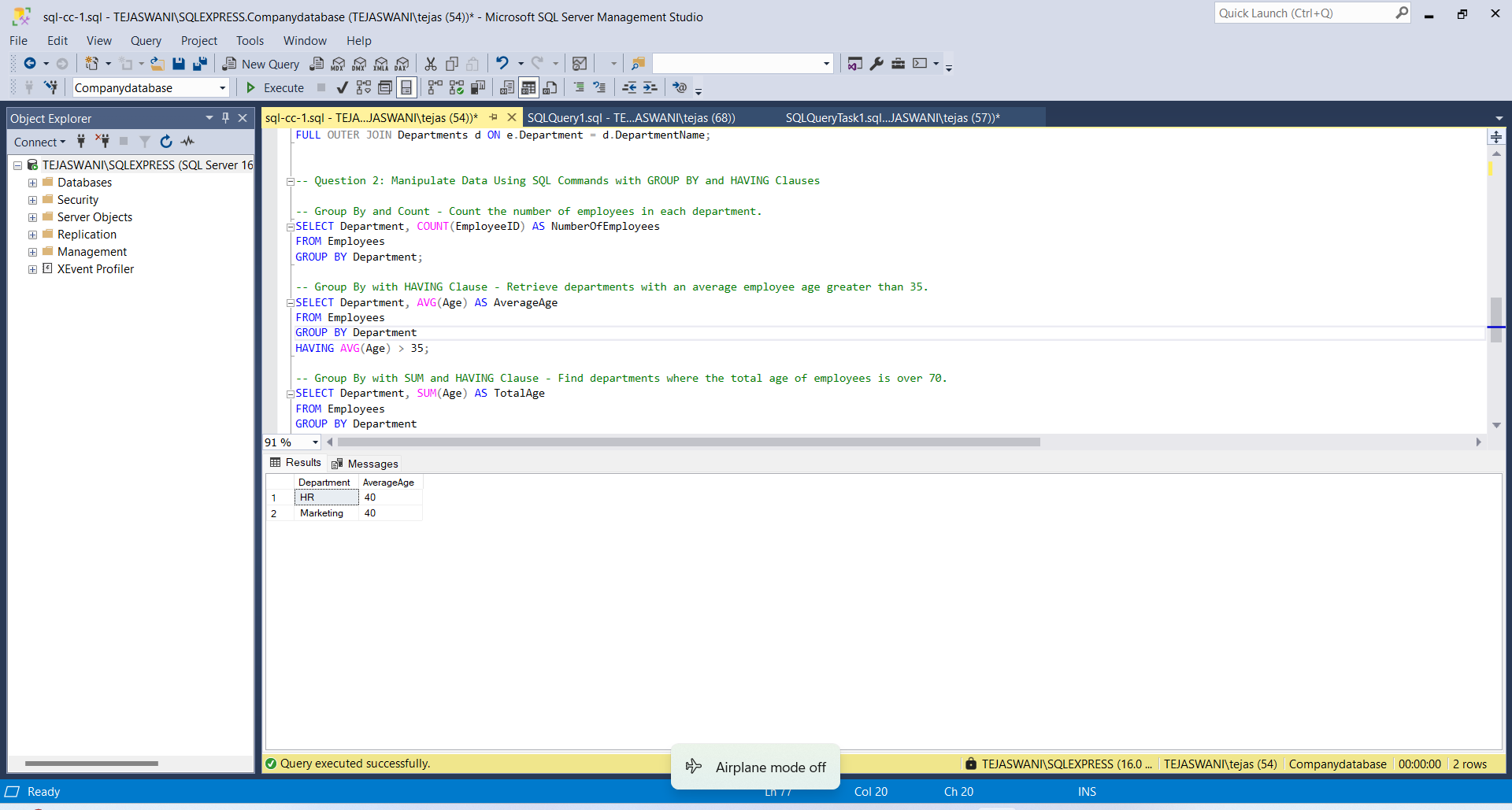
**2.Group By with HAVING Clause** - Retrieve departments with an average employee age greater than 35.

SELECT Department, AVG(Age) AS AverageAge FROM Employees GROUP BY Department HAVING AVG(Age) > 35;

This query is designed to retrieve departments with an average employee age greater than 35. Here's a brief explanation:

* It selects the Department and calculates the average age (AVG(Age)) for each department.
* The results are grouped by Department using the GROUP BY clause.
* The HAVING clause filters the results to show only departments where the average age is greater than 35.

This query helps identify departments with older employees on average, which could be useful for various HR and management decisions.



**3.Group By with SUM and HAVING Clause** - Find departments where the total age of employees is over 70.

SELECT Department, SUM(Age) AS TotalAge FROM Employees GROUP BY Department HAVING SUM(Age) > 70;

This SQL query is designed to find departments where the total age of employees is over 70. Let's break it down:

**1. SELECT clause:**

* SELECT Department: This selects the Department column.
* SUM(Age) AS TotalAge: This calculates the sum of all employee ages in each department and aliases it as TotalAge.

**2. FROM clause:**

* FROM Employees: This specifies that we're querying the Employees table.

**3. GROUP BY clause:**

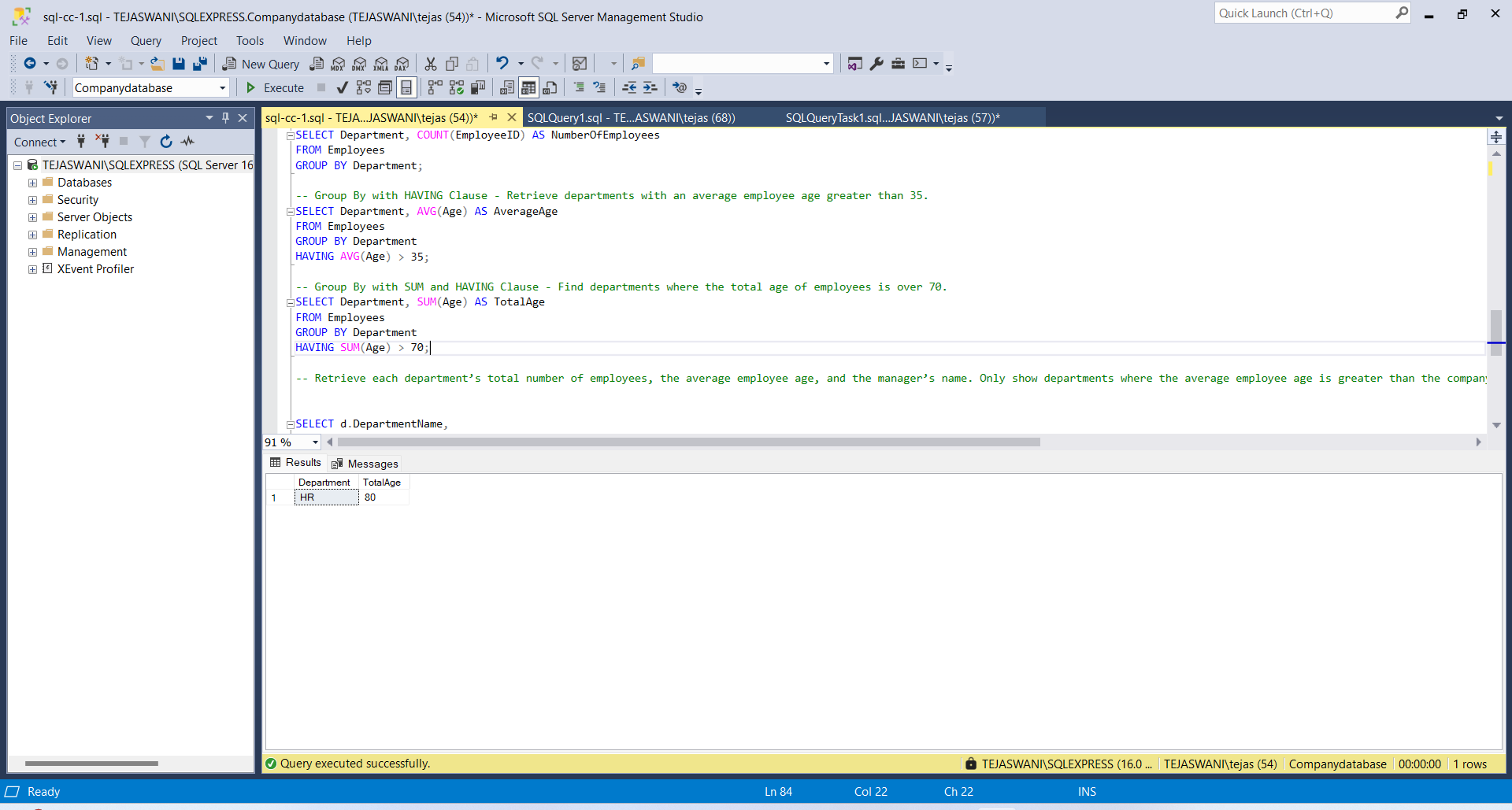
* GROUP BY Department: This groups the results by Department, allowing us to perform aggregate functions (like SUM) on each department separately.

**4. HAVING clause:**

* HAVING SUM(Age) > 70: This filters the grouped results, showing only departments where the sum of employee ages is greater than 70.

The HAVING clause is used instead of WHERE because it allows filtering on aggregate functions (like SUM) after the grouping has been performed.

This query effectively identifies departments with a higher total age, which could indicate departments with more senior employees or larger teams.



### **4.Retrieve each department’s total number of employees, the average employee age, and the manager’s name. Only show departments where the average employee age is greater than the company-wide average age. Additionally, include departments even if they have no employees.**

This combined question requires:

* Using **joins** to include information from both Employees and Departments.
* Using **subqueries** to compare the average age within each department to the company-wide average age.
* Utilizing **grouping** to calculate totals and averages by department.
* Applying a **HAVING clause** to filter based on the average age condition.

Here’s how you could structure the query to answer this question:

SELECT d.DepartmentName, d.Manager, COUNT(e.EmployeeID) AS NumberOfEmployees, AVG(e.Age) AS AverageAge FROM Departments d LEFT JOIN Employees e ON d.DepartmentName = e.Department GROUP BY d.DepartmentName, d.Manager HAVING AVG(ISNULL(e.Age, 0)) > (SELECT AVG(Age) FROM Employees);

### Explanation:

1. **LEFT JOIN**: Ensures all departments are included, even if they have no employees.
2. **COUNT(e.EmployeeID)**: Counts the number of employees per department.
3. **AVG(e.Age)**: Calculates the average age of employees in each department.
4. **HAVING Clause**: Filters the result to show only departments where the average age is greater than the company-wide average.
5. **Subquery** (SELECT AVG(Age) FROM Employees): Calculates the company-wide average age.

This query provides insights into departments with an older-than-average workforce, while including departments even if they have no employees.

