**IFT 530: Advanced Database Management System**

**Assignment: Final Project**

Group 16

[Jahnavi Lasyapriya Vavilala](mailto:lasyapriya.vavilala@gmail.com)[1231997652]

Sruthi Keerthana Nuttakki [1232316477]

**Submitted to:**

**Prof. Ashish Gulati**

**Dec 6, 2024**

**TABLE OF CONTENTS**

***Project: Database Management for Sentiment Analysis on Social Media.***

1. *Summary*
2. *Tables*
3. *ERD Diagram*
4. *Views*

* *View 1: Positive Sentiment Posts by Platform*
* *View 2: Active users with high engagement*
* *View 3: Active\_users\_Insights*

1. *Triggers*
2. *Stored Procedures*
3. *User Defined Functions*
4. *Cursors*

### **TOPIC: DATABASE MANAGEMENT FOR SENTIMENT ANALYSIS ON SOCIAL MEDIA**

Summary:

**1. ABSTRACT AND USE OF THE TABLES**

Social media is an engaging two-way platform fostering communication amongst the customers and brands/ events. With social media gaining popularity in recent years, it can be leveraged to understand brands’ perception in the public. Sentiment analysis determines the opinion of customers allowing businesses to optimize their products/ services. Apart from businesses sentiment analysis is helpful in law enforcement, research, education, journalism, entertainment etc. Therefore, with sentiment analysis one can understand the trends, public opinion, engagement to reflect on decision making. Using Natural Language Processing (NLP), Machine Learning (ML) and text classification sentiments are grouped into positive, negative and neutral.

In the scope of this project, a database system will be created to handle volumes of data for efficient storage, organization and retrieval. A database system ensures high querying, allows cross platform analysis and facilitates easy updation.

In this project, there would be 8 tables and 2 fact tables as follows:

**Main Tables**:

1. **Users**: Information about social media users (anonymous identifiers).
2. **Posts**: Raw social media posts (content, metadata).
3. **Platforms**: Details about social media platforms.
4. **Hashtags**: Track hashtags used in posts.
5. **Keywords**: Identify key terms for sentiment analysis.
6. **Sentiment\_Scores**: Store sentiment scores for analyzed posts.
7. **Languages**: Store supported languages.
8. **Locations**: Record the geographical location of posts.

**Fact Tables**:

1. **Post\_Analytics**: Aggregate data for posts (e.g., likes, shares, comments).
2. **Sentiment\_Trends**: Store trends over time (e.g., average sentiment by day).

**Task 1: Create the tables using SQL script you have defined in the Final project.**

Execution of creating Database named Group16 and using it for the project

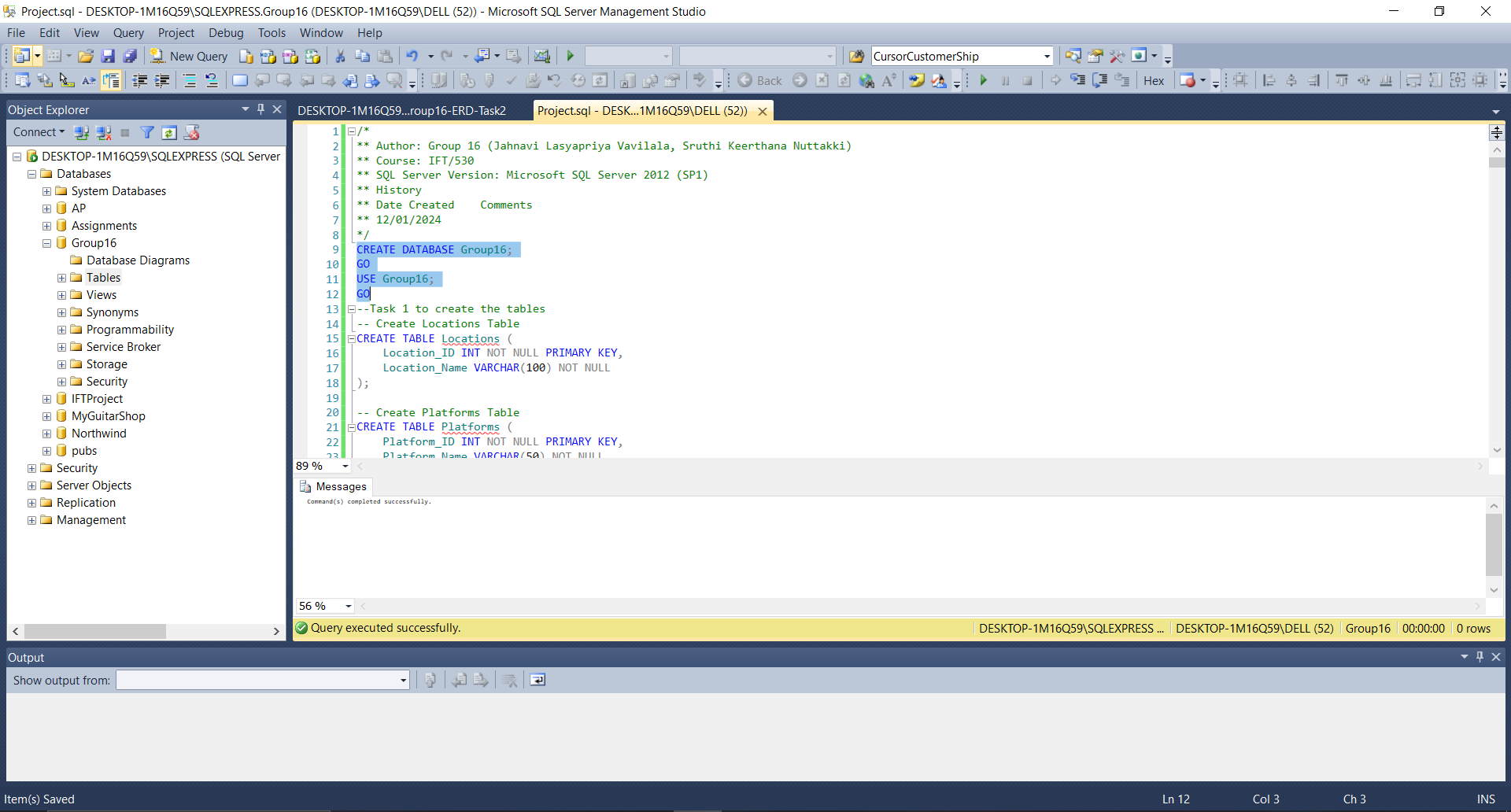
Code:

CREATE DATABASE Group16;

GO

USE Group16;

GO



Execution of creation of Tables as defined

Code:

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

CREATE DATABASE Group16;

GO

USE Group16;

GO

--Task 1 to create the tables

-- Create Locations Table

CREATE TABLE Locations (

Location\_ID INT NOT NULL PRIMARY KEY,

Location\_Name VARCHAR(100) NOT NULL

);

-- Create Platforms Table

CREATE TABLE Platforms (

Platform\_ID INT NOT NULL PRIMARY KEY,

Platform\_Name VARCHAR(50) NOT NULL

);

-- Create Languages Table

CREATE TABLE Languages (

Language\_ID INT NOT NULL PRIMARY KEY,

Language\_Name VARCHAR(50) NOT NULL UNIQUE

);

-- Create Users Table

CREATE TABLE Users (

User\_ID INT NOT NULL PRIMARY KEY,

Username VARCHAR(50) NOT NULL,

Age INT CHECK (Age >= 13),

Gender CHAR(1),

Location\_ID INT,

FOREIGN KEY (Location\_ID) REFERENCES Locations(Location\_ID)

);

-- Create Posts Table

CREATE TABLE Posts (

Post\_ID INT NOT NULL PRIMARY KEY,

User\_ID INT NOT NULL,

Platform\_ID INT NOT NULL,

Post\_Content TEXT NOT NULL,

Language\_ID INT,

Post\_Date DATETIME NOT NULL,

Hashtags TEXT,

Keywords TEXT,

FOREIGN KEY (User\_ID) REFERENCES Users(User\_ID),

FOREIGN KEY (Platform\_ID) REFERENCES Platforms(Platform\_ID),

FOREIGN KEY (Language\_ID) REFERENCES Languages(Language\_ID)

);

-- Create Hashtags Table

CREATE TABLE Hashtags (

Hashtag\_ID INT NOT NULL PRIMARY KEY,

Hashtag\_Text VARCHAR(100) NOT NULL UNIQUE

);

-- Create Keywords Table

CREATE TABLE Keywords (

Keyword\_ID INT NOT NULL PRIMARY KEY,

Keyword\_Text VARCHAR(100) NOT NULL UNIQUE

);

-- Create Sentiment\_Scores Table

CREATE TABLE Sentiment\_Scores (

Score\_ID INT NOT NULL PRIMARY KEY,

Post\_ID INT NOT NULL,

Sentiment\_Score DECIMAL(3, 2) NOT NULL CHECK (Sentiment\_Score BETWEEN -1.00 AND 1.00),

FOREIGN KEY (Post\_ID) REFERENCES Posts(Post\_ID)

);

-- Create Post\_Analytics Fact Table

CREATE TABLE Post\_Analytics (

Analytics\_ID INT NOT NULL PRIMARY KEY,

Post\_ID INT NOT NULL,

Likes\_Count INT NOT NULL DEFAULT 0,

Shares\_Count INT NOT NULL DEFAULT 0,

Comments\_Count INT NOT NULL DEFAULT 0,

Views\_Count INT NOT NULL DEFAULT 0,

Engagement\_Rate DECIMAL(5, 2) NOT NULL DEFAULT 0,

Analysis\_Date DATETIME NOT NULL,

FOREIGN KEY (Post\_ID) REFERENCES Posts(Post\_ID)

);

-- Create Sentiment\_Trends Fact Table

CREATE TABLE Sentiment\_Trends (

Trend\_ID INT NOT NULL PRIMARY KEY,

Platform\_ID INT NOT NULL,

Date DATE NOT NULL,

Avg\_Sentiment DECIMAL(3, 2) NOT NULL CHECK (Avg\_Sentiment BETWEEN -1.00 AND 1.00),

Post\_Count INT NOT NULL DEFAULT 0,

Positive\_Ratio DECIMAL(5, 2) NOT NULL CHECK (Positive\_Ratio BETWEEN 0 AND 100),

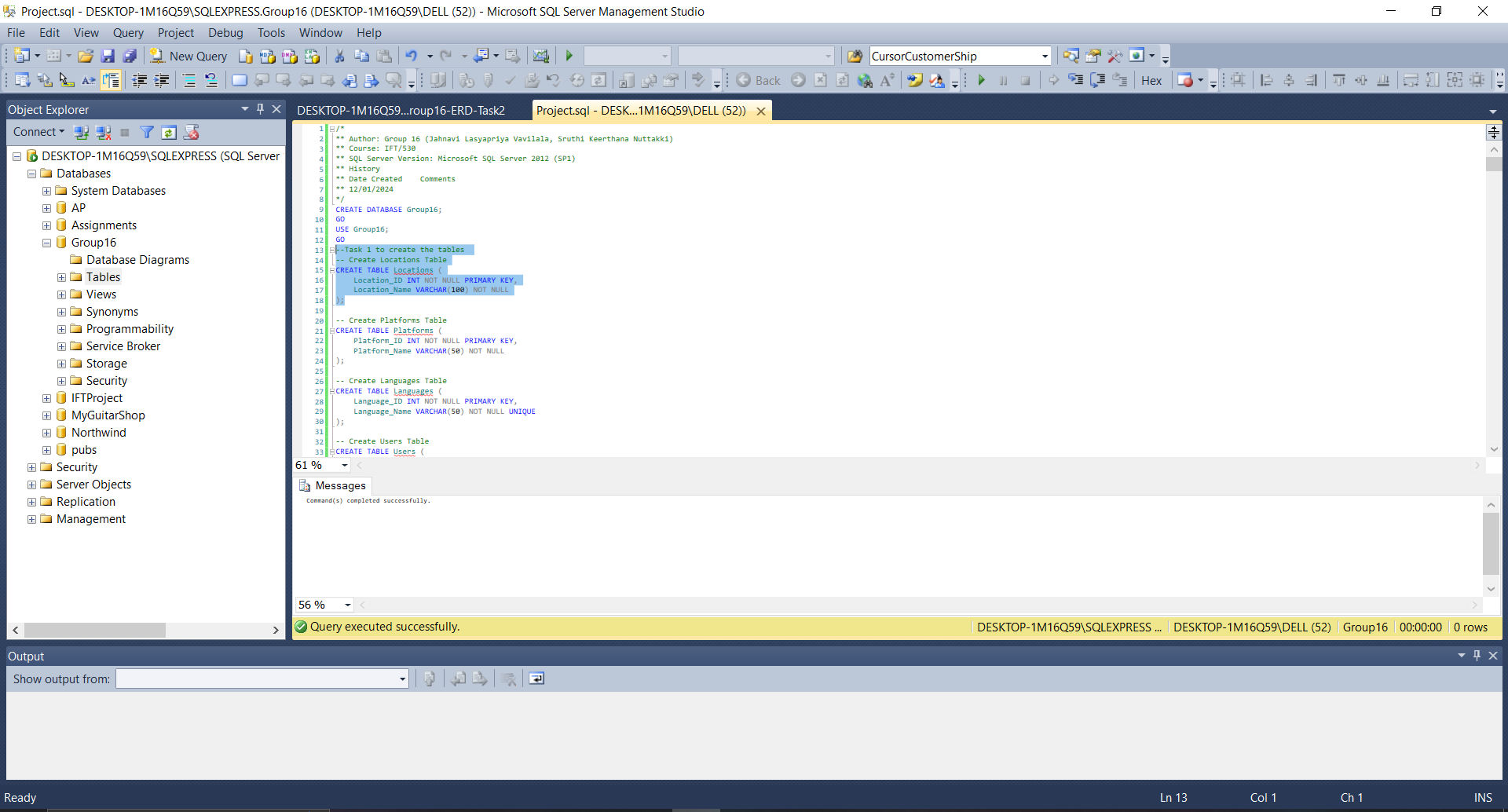
Negative\_Ratio DECIMAL(5, 2) NOT NULL CHECK (Negative\_Ratio BETWEEN 0 AND 100),

Neutral\_Ratio DECIMAL(5, 2) NOT NULL CHECK (Neutral\_Ratio BETWEEN 0 AND 100),

FOREIGN KEY (Platform\_ID) REFERENCES Platforms(Platform\_ID)

);

Note: Please execute the tables with no reference to another table first so that there will be no FK error.



Executed script to see the created tables:

Code:

-- Script to see the tables

SELECT \* FROM Locations;

SELECT \* FROM Platforms;

SELECT \* FROM Languages;

SELECT \* FROM Users;

SELECT \* FROM Keywords;

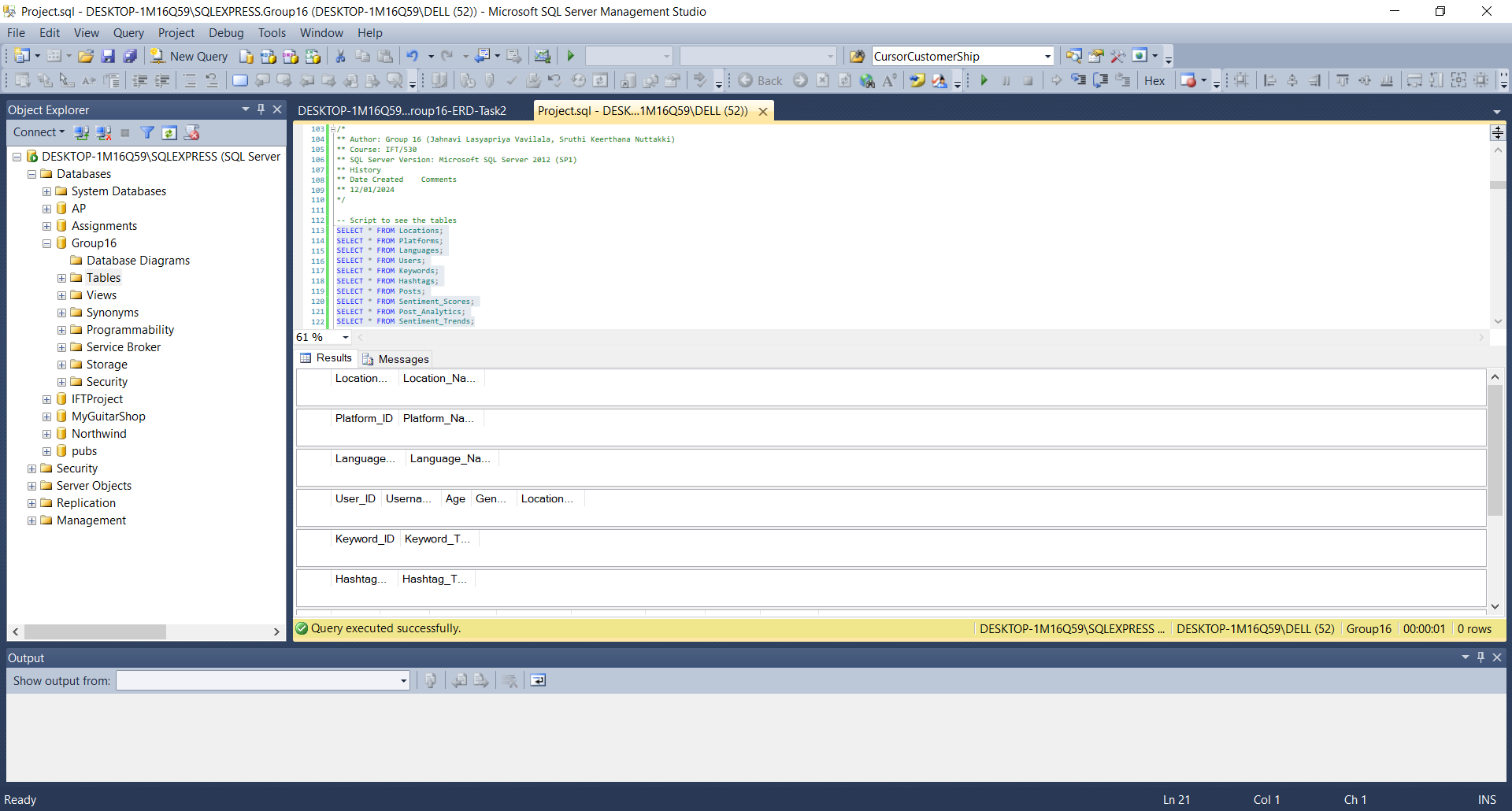
SELECT \* FROM Hashtags;

SELECT \* FROM Posts;

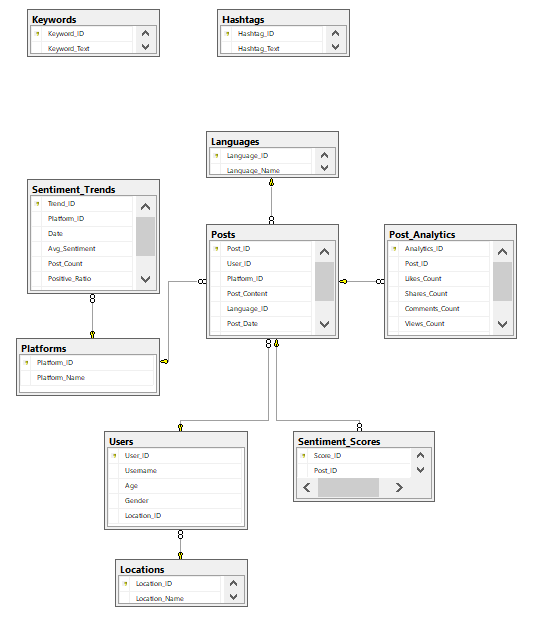
SELECT \* FROM Sentiment\_Scores;

SELECT \* FROM Post\_Analytics;

SELECT \* FROM Sentiment\_Trends;

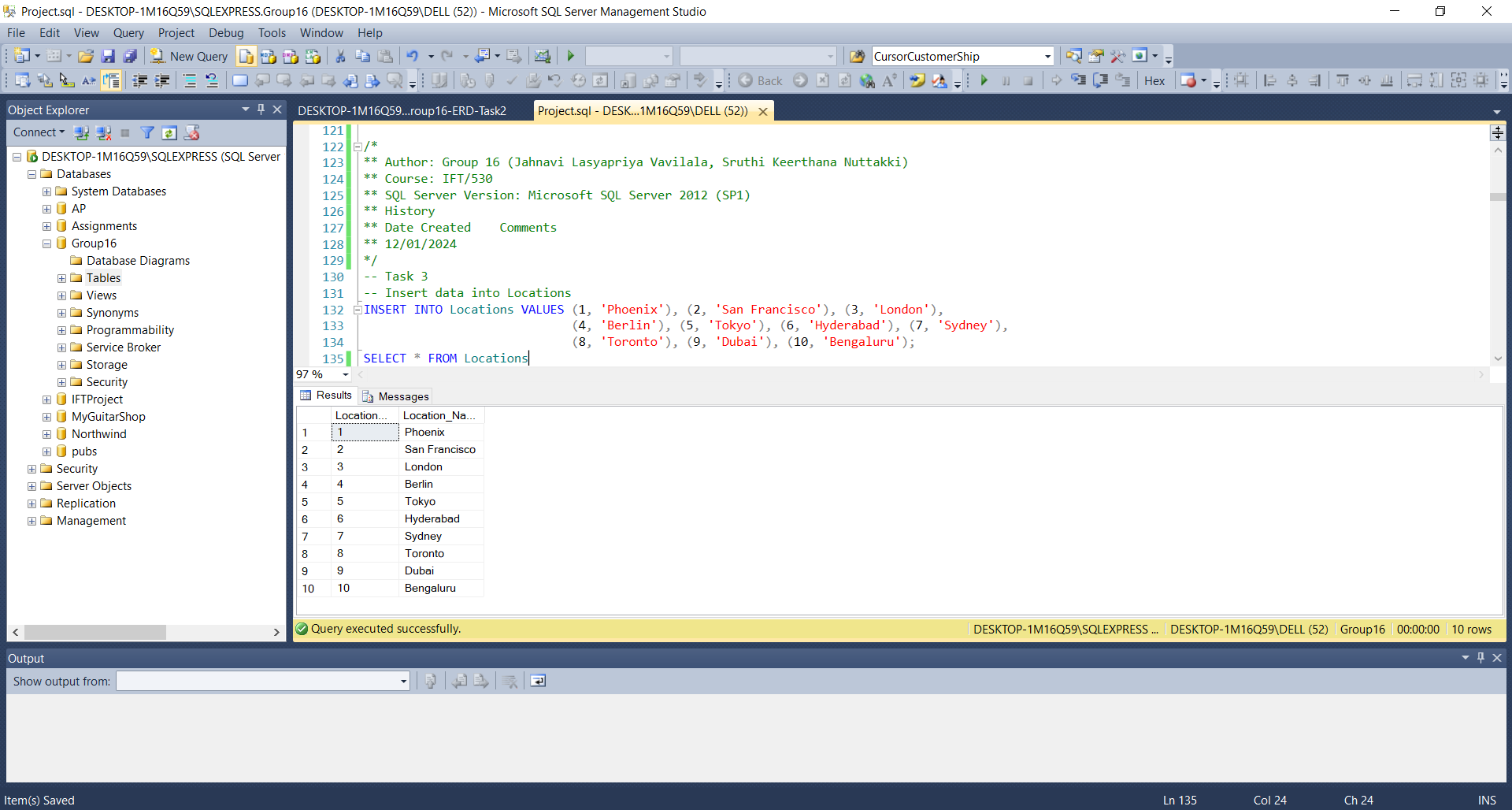


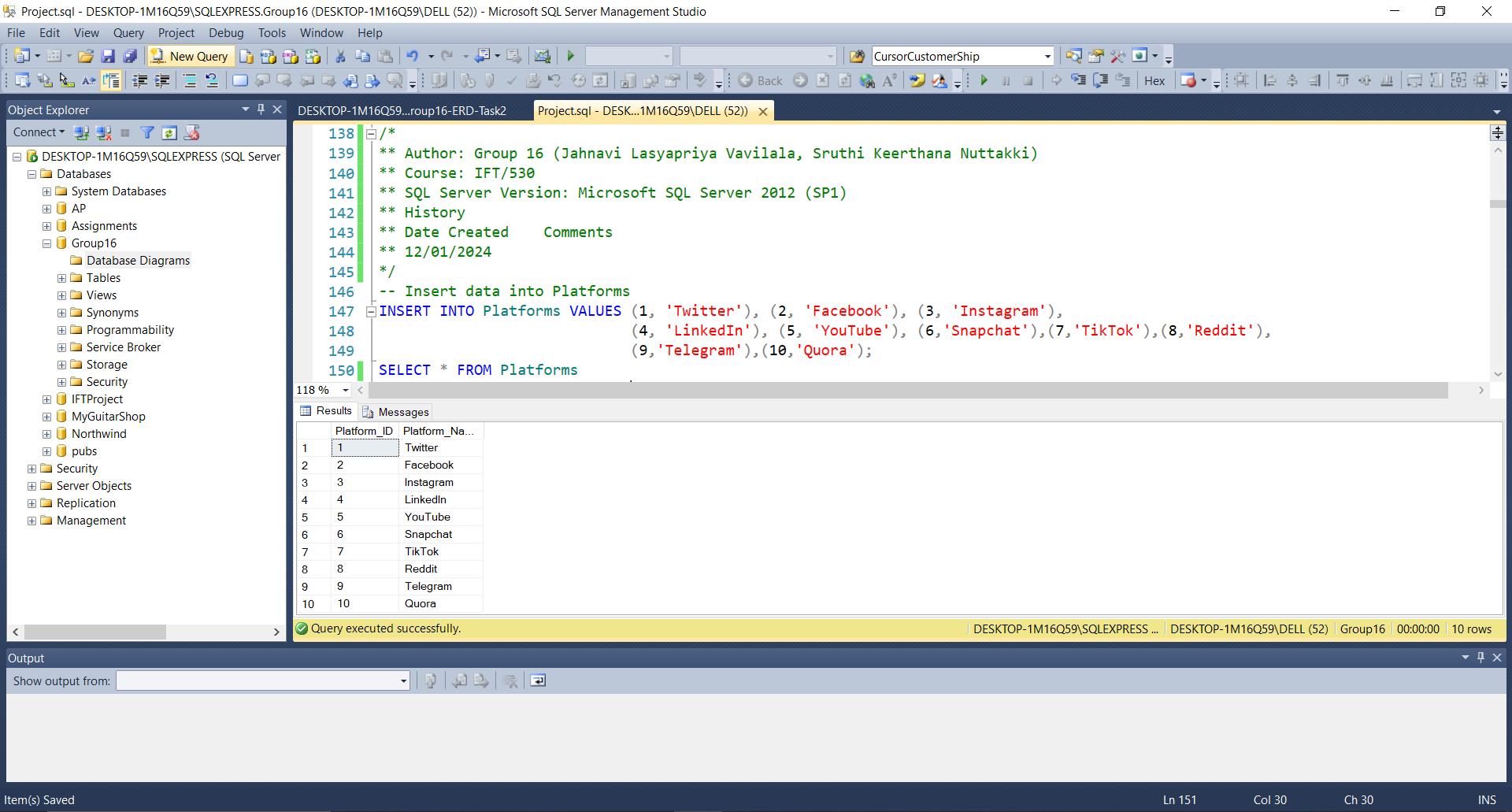
**Task 2: Generate the ERD diagram and paste it in a word document.**

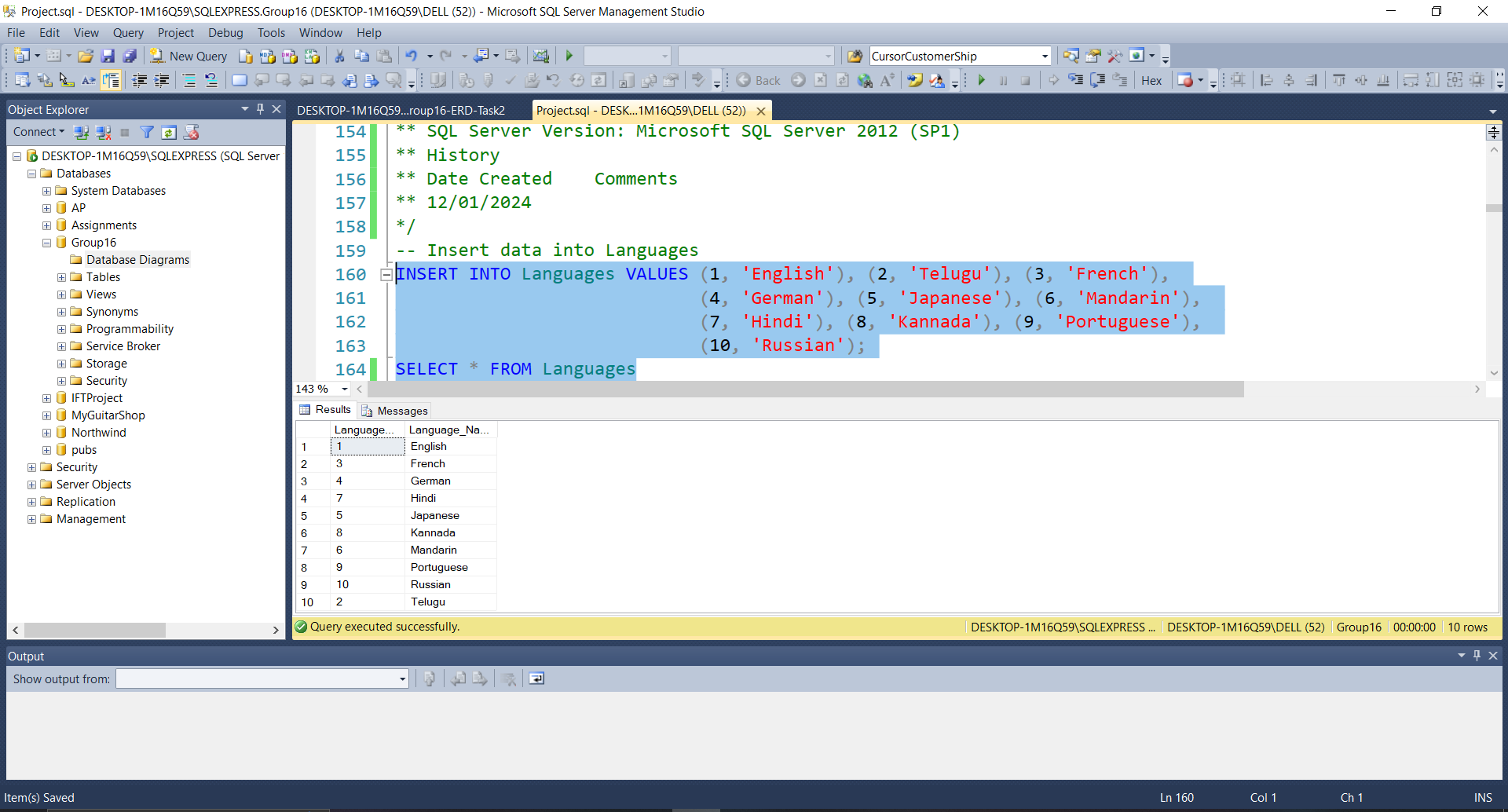


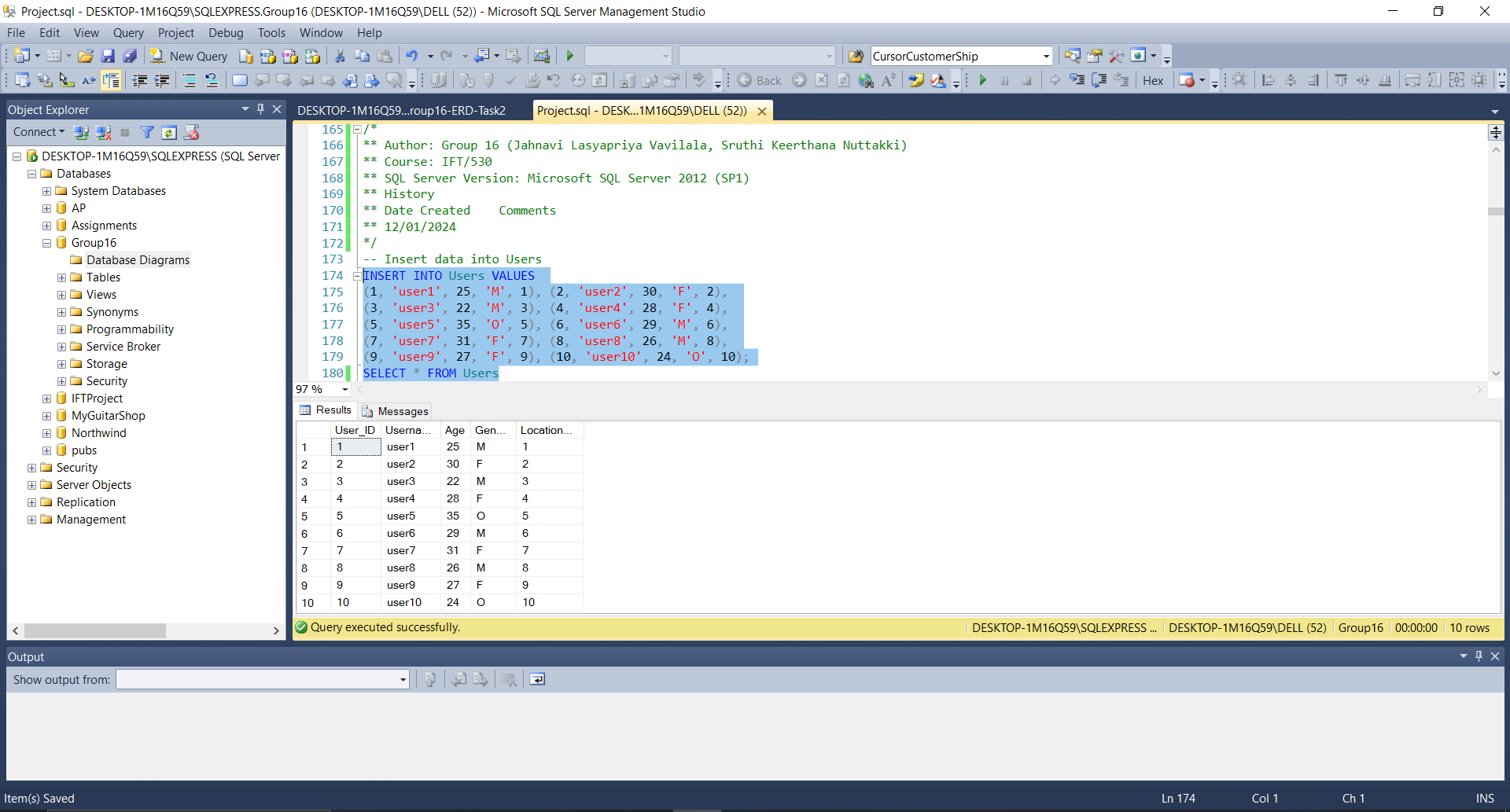
**Task 3:**

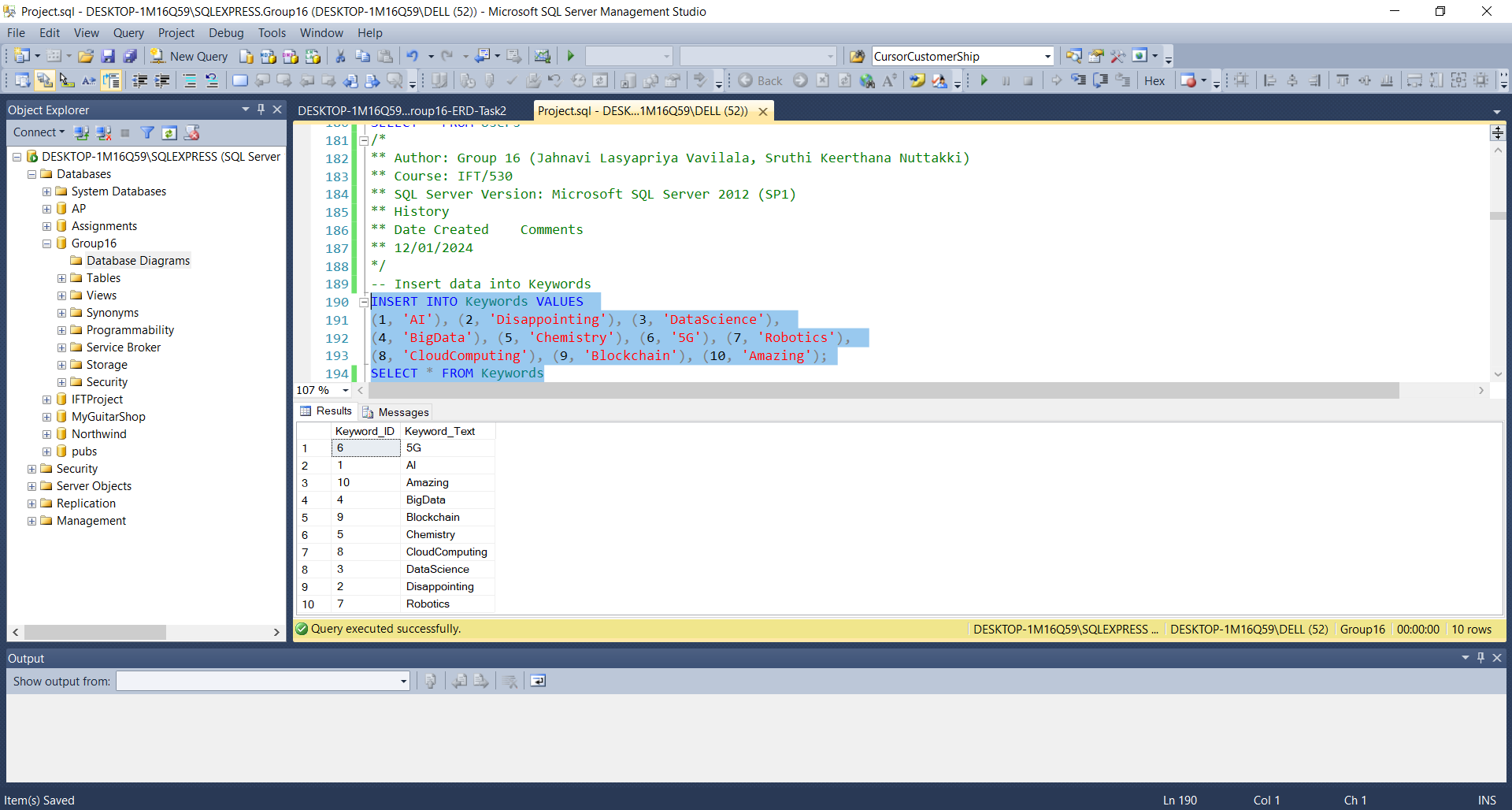
**Populate the table with 10 rows for dimension tables, 20 – 50 rows for transactional tables.**

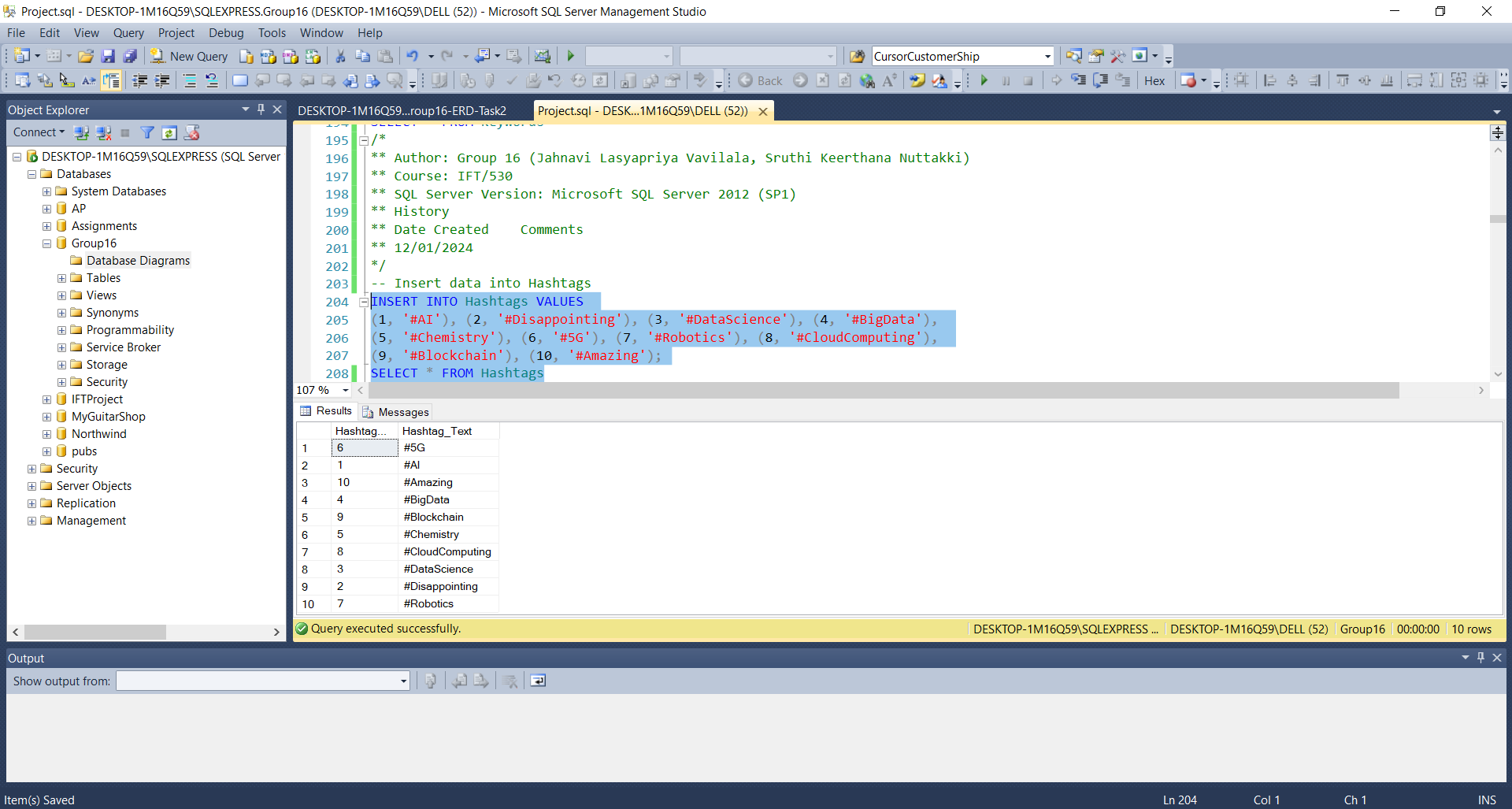


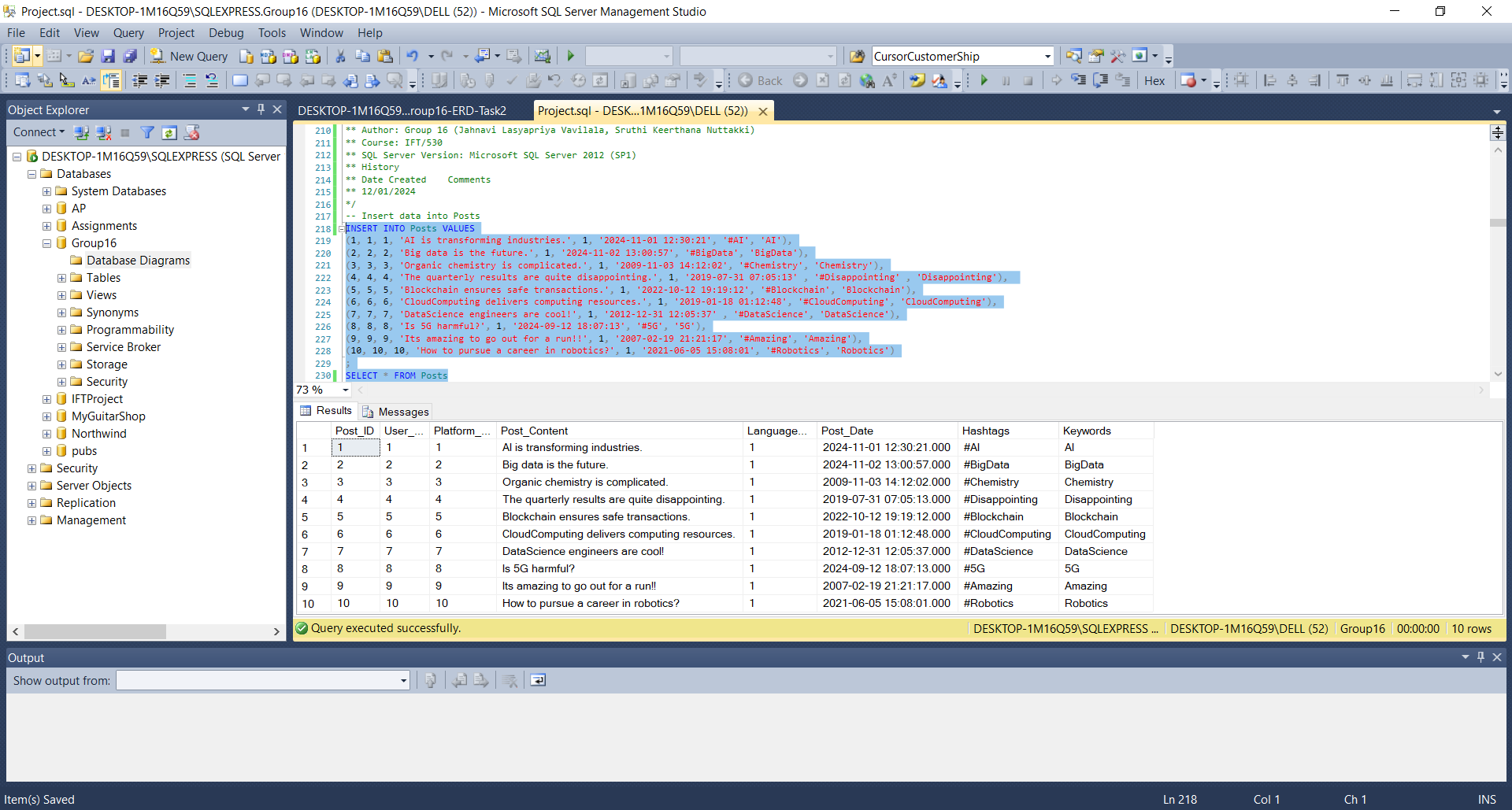


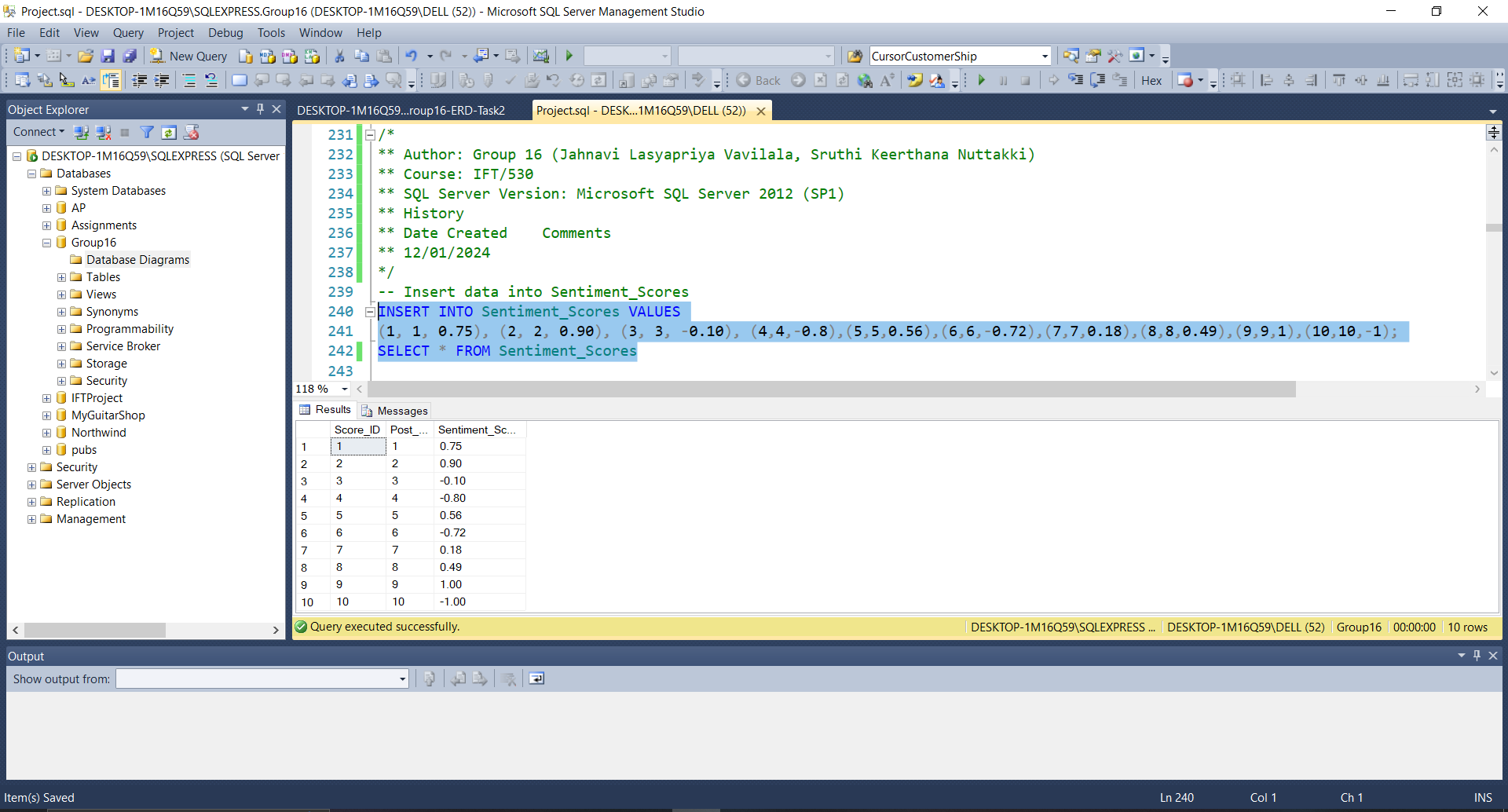


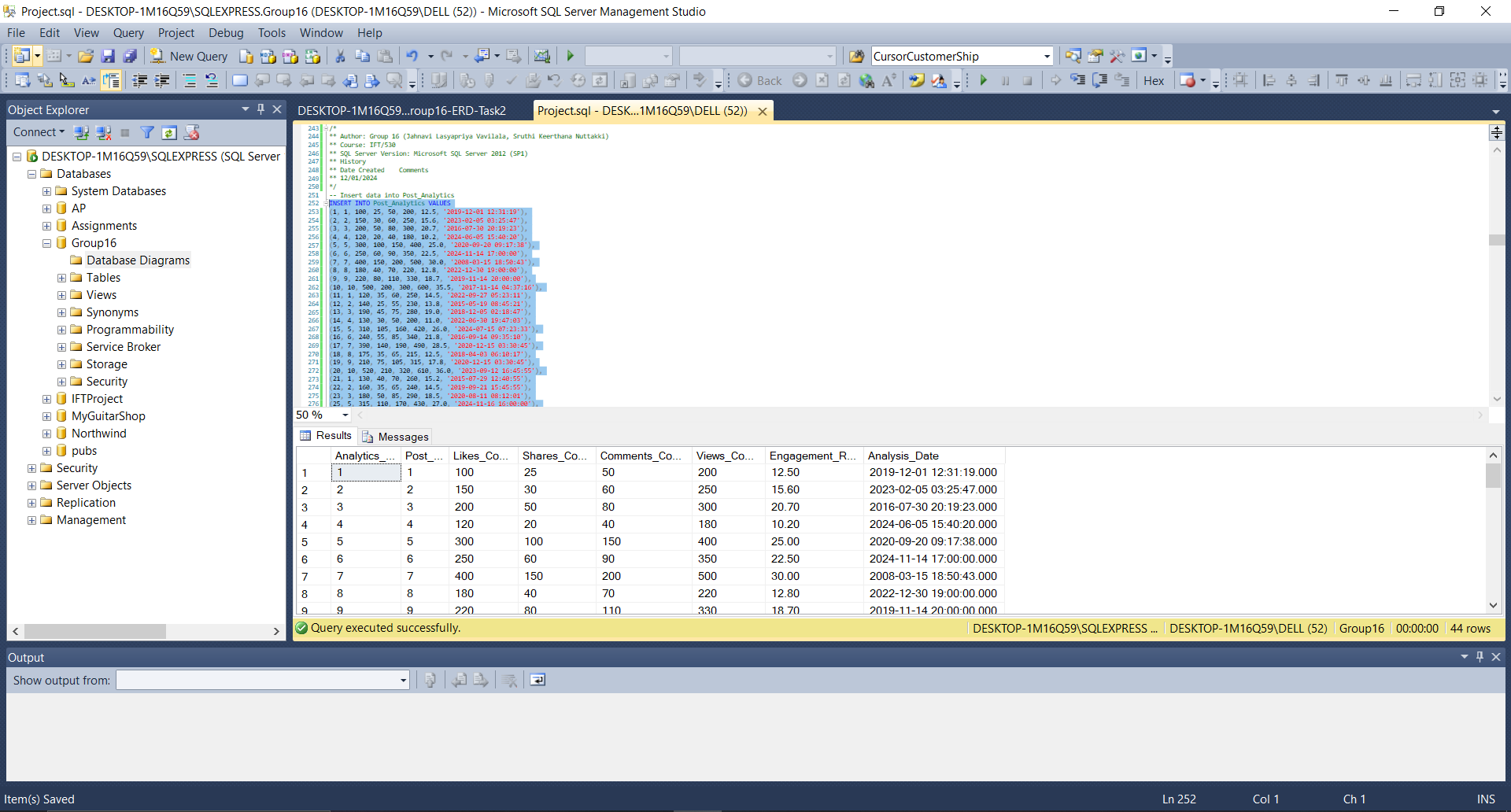


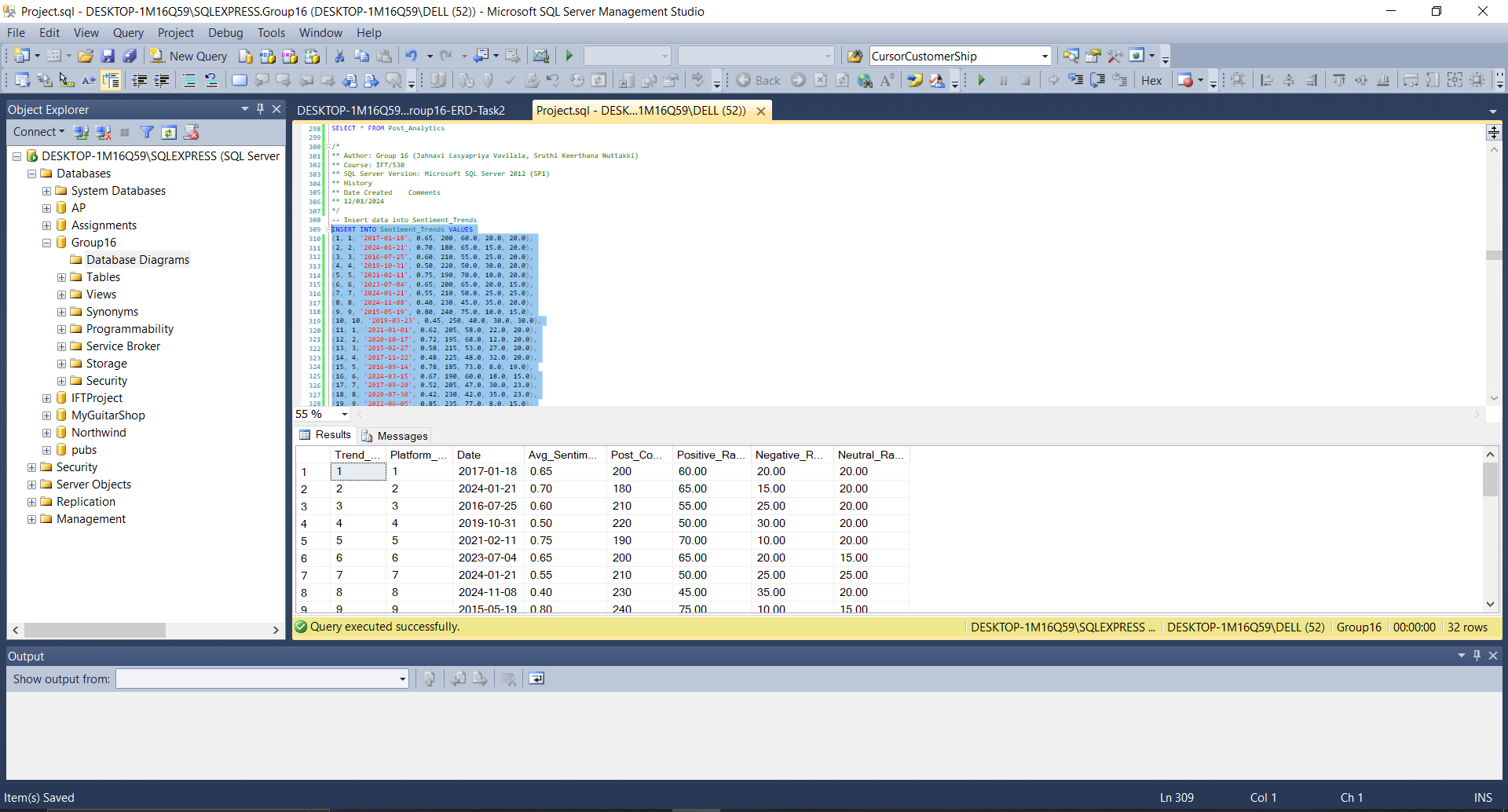












The entire code for populating the tables is as follows:

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Task 3

-- Insert data into Locations

INSERT INTO Locations VALUES (1, 'Phoenix'), (2, 'San Francisco'), (3, 'London'),

(4, 'Berlin'), (5, 'Tokyo'), (6, 'Hyderabad'), (7, 'Sydney'),

(8, 'Toronto'), (9, 'Dubai'), (10, 'Bengaluru');

SELECT \* FROM Locations

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Insert data into Platforms

INSERT INTO Platforms VALUES (1, 'Twitter'), (2, 'Facebook'), (3, 'Instagram'),

(4, 'LinkedIn'), (5, 'YouTube'), (6,'Snapchat'),(7,'TikTok'),(8,'Reddit'),

(9,'Telegram'),(10,'Quora');

SELECT \* FROM Platforms

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Insert data into Languages

INSERT INTO Languages VALUES (1, 'English'), (2, 'Telugu'), (3, 'French'),

(4, 'German'), (5, 'Japanese'), (6, 'Mandarin'),

(7, 'Hindi'), (8, 'Kannada'), (9, 'Portuguese'),

(10, 'Russian');

SELECT \* FROM Languages

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Insert data into Users

INSERT INTO Users VALUES

(1, 'user1', 25, 'M', 1), (2, 'user2', 30, 'F', 2),

(3, 'user3', 22, 'M', 3), (4, 'user4', 28, 'F', 4),

(5, 'user5', 35, 'O', 5), (6, 'user6', 29, 'M', 6),

(7, 'user7', 31, 'F', 7), (8, 'user8', 26, 'M', 8),

(9, 'user9', 27, 'F', 9), (10, 'user10', 24, 'O', 10);

SELECT \* FROM Users

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Insert data into Keywords

INSERT INTO Keywords VALUES

(1, 'AI'), (2, 'Disappointing'), (3, 'DataScience'),

(4, 'BigData'), (5, 'Chemistry'), (6, '5G'), (7, 'Robotics'),

(8, 'CloudComputing'), (9, 'Blockchain'), (10, 'Amazing');

SELECT \* FROM Keywords

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Insert data into Hashtags

INSERT INTO Hashtags VALUES

(1, '#AI'), (2, '#Disappointing'), (3, '#DataScience'), (4, '#BigData'),

(5, '#Chemistry'), (6, '#5G'), (7, '#Robotics'), (8, '#CloudComputing'),

(9, '#Blockchain'), (10, '#Amazing');

SELECT \* FROM Hashtags

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Insert data into Posts

INSERT INTO Posts VALUES

(1, 1, 1, 'AI is transforming industries.', 1, '2024-11-01 12:30:21', '#AI', 'AI'),

(2, 2, 2, 'Big data is the future.', 1, '2024-11-02 13:00:57', '#BigData', 'BigData'),

(3, 3, 3, 'Organic chemistry is complicated.', 1, '2009-11-03 14:12:02', '#Chemistry', 'Chemistry'),

(4, 4, 4, 'The quarterly results are quite disappointing.', 1, '2019-07-31 07:05:13' , '#Disappointing' , 'Disappointing'),

(5, 5, 5, 'Blockchain ensures safe transactions.', 1, '2022-10-12 19:19:12', '#Blockchain', 'Blockchain'),

(6, 6, 6, 'CloudComputing delivers computing resources.', 1, '2019-01-18 01:12:48', '#CloudComputing', 'CloudComputing'),

(7, 7, 7, 'DataScience engineers are cool!', 1, '2012-12-31 12:05:37' , '#DataScience', 'DataScience'),

(8, 8, 8, 'Is 5G harmful?', 1, '2024-09-12 18:07:13', '#5G', '5G'),

(9, 9, 9, 'Its amazing to go out for a run!!', 1, '2007-02-19 21:21:17', '#Amazing', 'Amazing'),

(10, 10, 10, 'How to pursue a career in robotics?', 1, '2021-06-05 15:08:01', '#Robotics', 'Robotics')

;

SELECT \* FROM Posts

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Insert data into Sentiment\_Scores

INSERT INTO Sentiment\_Scores VALUES

(1, 1, 0.75), (2, 2, 0.90), (3, 3, -0.10), (4,4,-0.8),(5,5,0.56),(6,6,-0.72),(7,7,0.18),(8,8,0.49),(9,9,1),(10,10,-1);

SELECT \* FROM Sentiment\_Scores

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Insert data into Post\_Analytics

INSERT INTO Post\_Analytics VALUES

(1, 1, 100, 25, 50, 200, 12.5, '2019-12-01 12:31:19'),

(2, 2, 150, 30, 60, 250, 15.6, '2023-02-05 03:25:47'),

(3, 3, 200, 50, 80, 300, 20.7, '2016-07-30 20:19:23'),

(4, 4, 120, 20, 40, 180, 10.2, '2024-06-05 15:40:20'),

(5, 5, 300, 100, 150, 400, 25.0, '2020-09-20 09:17:38'),

(6, 6, 250, 60, 90, 350, 22.5, '2024-11-14 17:00:00'),

(7, 7, 400, 150, 200, 500, 30.0, '2008-03-15 18:50:43'),

(8, 8, 180, 40, 70, 220, 12.8, '2022-12-30 19:00:00'),

(9, 9, 220, 80, 110, 330, 18.7, '2019-11-14 20:00:00'),

(10, 10, 500, 200, 300, 600, 35.5, '2017-11-14 04:37:16'),

(11, 1, 120, 35, 60, 250, 14.5, '2022-09-27 05:23:11'),

(12, 2, 140, 25, 55, 230, 13.8, '2015-05-19 08:45:21'),

(13, 3, 190, 45, 75, 280, 19.0, '2018-12-05 02:18:47'),

(14, 4, 130, 30, 50, 200, 11.0, '2022-06-30 19:47:03'),

(15, 5, 310, 105, 160, 420, 26.0, '2024-07-15 07:23:33'),

(16, 6, 240, 55, 85, 340, 21.8, '2016-09-14 09:35:10'),

(17, 7, 390, 140, 190, 490, 28.5, '2020-12-15 03:30:45'),

(18, 8, 175, 35, 65, 215, 12.5, '2018-04-03 06:10:17'),

(19, 9, 210, 75, 105, 315, 17.8, '2020-12-15 03:30:45'),

(20, 10, 520, 210, 320, 610, 36.0, '2023-09-12 16:45:55'),

(21, 1, 130, 40, 70, 260, 15.2, '2015-07-29 12:40:55'),

(22, 2, 160, 35, 65, 240, 14.5, '2019-09-21 15:45:55'),

(23, 3, 180, 50, 85, 290, 18.5, '2020-08-11 08:12:01'),

(25, 5, 315, 110, 170, 430, 27.0, '2024-11-16 16:00:00'),

(26, 6, 230, 50, 80, 330, 21.0, '2022-02-28 20:12:39'),

(27, 7, 410, 160, 210, 510, 29.0, '2020-04-15 22:14:37'),

(28, 8, 165, 30, 60, 205, 11.8, '2020-11-19 19:10:04'),

(29, 9, 225, 85, 115, 325, 19.0, '2023-07-04 17:35:47'),

(30, 10, 510, 195, 305, 620, 37.2, '2017-08-15 13:40:55'),

(31, 1, 140, 45, 75, 270, 16.0, '2020-10-17 16:00:31'),

(32, 2, 150, 40, 70, 250, 15.0, '2017-11-22 22:59:45'),

(33, 3, 175, 55, 90, 300, 17.5, '2022-02-28 20:12:39'),

(34, 4, 135, 35, 55, 210, 12.0, '2024-03-15 17:30:027'),

(35, 5, 320, 115, 175, 440, 28.0, '2018-12-04 05:06:43'),

(36, 6, 220, 45, 75, 320, 20.0, '2024-11-17 17:00:00'),

(37, 7, 405, 155, 205, 520, 29.5, '2016-03-30 17:13:08'),

(38, 8, 170, 35, 65, 215, 12.2, '2019-01-15 02:34:57'),

(39, 9, 220, 80, 110, 335, 18.2, '2023-05-20 09:55:27'),

(40, 10, 515, 200, 310, 630, 36.5, '2018-12-05 02:18:47'),

(41, 1, 120, 35, 65, 240, 14.5, '2015-12-26 12:15:35'),

(42, 2, 145, 30, 60, 220, 13.5, '2020-10-17 16:00:31'),

(43, 3, 170, 50, 85, 280, 18.0, '2023-04-10 21:45:55'),

(44, 4, 125, 25, 45, 190, 10.5, '2019-07-11 09:31:07'),

(45, 5, 310, 105, 165, 420, 26.0, '2021-01-01 11:11:11');

;

SELECT \* FROM Post\_Analytics

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Insert data into Sentiment\_Trends

INSERT INTO Sentiment\_Trends VALUES

(1, 1, '2017-01-18', 0.65, 200, 60.0, 20.0, 20.0),

(2, 2, '2024-01-21', 0.70, 180, 65.0, 15.0, 20.0),

(3, 3, '2016-07-25', 0.60, 210, 55.0, 25.0, 20.0),

(4, 4, '2019-10-31', 0.50, 220, 50.0, 30.0, 20.0),

(5, 5, '2021-02-11', 0.75, 190, 70.0, 10.0, 20.0),

(6, 6, '2023-07-04', 0.65, 200, 65.0, 20.0, 15.0),

(7, 7, '2024-01-21', 0.55, 210, 50.0, 25.0, 25.0),

(8, 8, '2024-11-08', 0.40, 230, 45.0, 35.0, 20.0),

(9, 9, '2015-05-19', 0.80, 240, 75.0, 10.0, 15.0),

(10, 10, '2019-03-23', 0.45, 250, 40.0, 30.0, 30.0),

(11, 1, '2021-01-01', 0.62, 205, 58.0, 22.0, 20.0),

(12, 2, '2020-10-17', 0.72, 195, 68.0, 12.0, 20.0),

(13, 3, '2015-02-27', 0.58, 215, 53.0, 27.0, 20.0),

(14, 4, '2017-11-22', 0.48, 225, 48.0, 32.0, 20.0),

(15, 5, '2016-09-14', 0.78, 185, 73.0, 8.0, 19.0),

(16, 6, '2024-03-15', 0.67, 190, 60.0, 18.0, 15.0),

(17, 7, '2017-09-20', 0.52, 205, 47.0, 30.0, 23.0),

(18, 8, '2020-07-30', 0.42, 230, 42.0, 35.0, 23.0),

(19, 9, '2022-06-05', 0.85, 235, 77.0, 8.0, 15.0),

(20, 10, '2015-03-31', 0.49, 245, 45.0, 30.0, 25.0),

(21, 1, '2021-11-21', 0.65, 210, 58.0, 20.0, 22.0),

(22, 2, '2020-12-17', 0.75, 200, 68.0, 10.0, 22.0),

(23, 3, '2017-05-27', 0.63, 215, 55.0, 25.0, 20.0),

(24, 4, '2015-07-29', 0.52, 220, 50.0, 30.0, 22.0),

(25, 5, '2024-09-14', 0.77, 180, 70.0, 8.0, 18.0),

(26, 6, '2017-11-19', 0.66, 190, 62.0, 18.0, 14.0),

(27, 7, '2022-09-20', 0.53, 210, 49.0, 29.0, 24.0),

(28, 8, '2016-04-03', 0.44, 225, 44.0, 35.0, 24.0),

(29, 9, '2018-01-29', 0.88, 235, 78.0, 7.0, 14.0),

(30, 10, '2024-11-30', 0.46, 240, 43.0, 29.0, 26.0),

(31, 1, '2015-08-09', 0.68, 215, 60.0, 22.0, 23.0),

(32, 2, '2023-12-27', 0.73, 205, 69.0, 11.0, 23.0)

SELECT \* FROM Sentiment\_Trends;

**Task 4: Create three queries and convert them into views. Explain why you think would be useful to the user. All queries must be joined to one or more tables. Make sure you restrict the data by using a where clause for each of the queries.**

**View 1: Positive Sentiment Posts by Platform**

Purpose: This view lists posts with positive sentiment scores (Sentiment\_Score > 0.5) for each platform. It helps businesses identify which platforms host the most positive content, enabling targeted engagement strategies and campaign optimization.

**Code:**

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

--Task 4

--Query for View 1, Positive Sentiment Posts by Platform, Explanation: This view lists posts with positive sentiment scores (Sentiment\_Score > 0.5) for each platform.

CREATE VIEW Positive\_Posts\_By\_Platform AS

SELECT

ps.Post\_ID,

p.Platform\_Name,

ps.Post\_Content,

ss.Sentiment\_Score

FROM

Posts ps

JOIN

Platforms p ON ps.Platform\_ID = p.Platform\_ID

JOIN

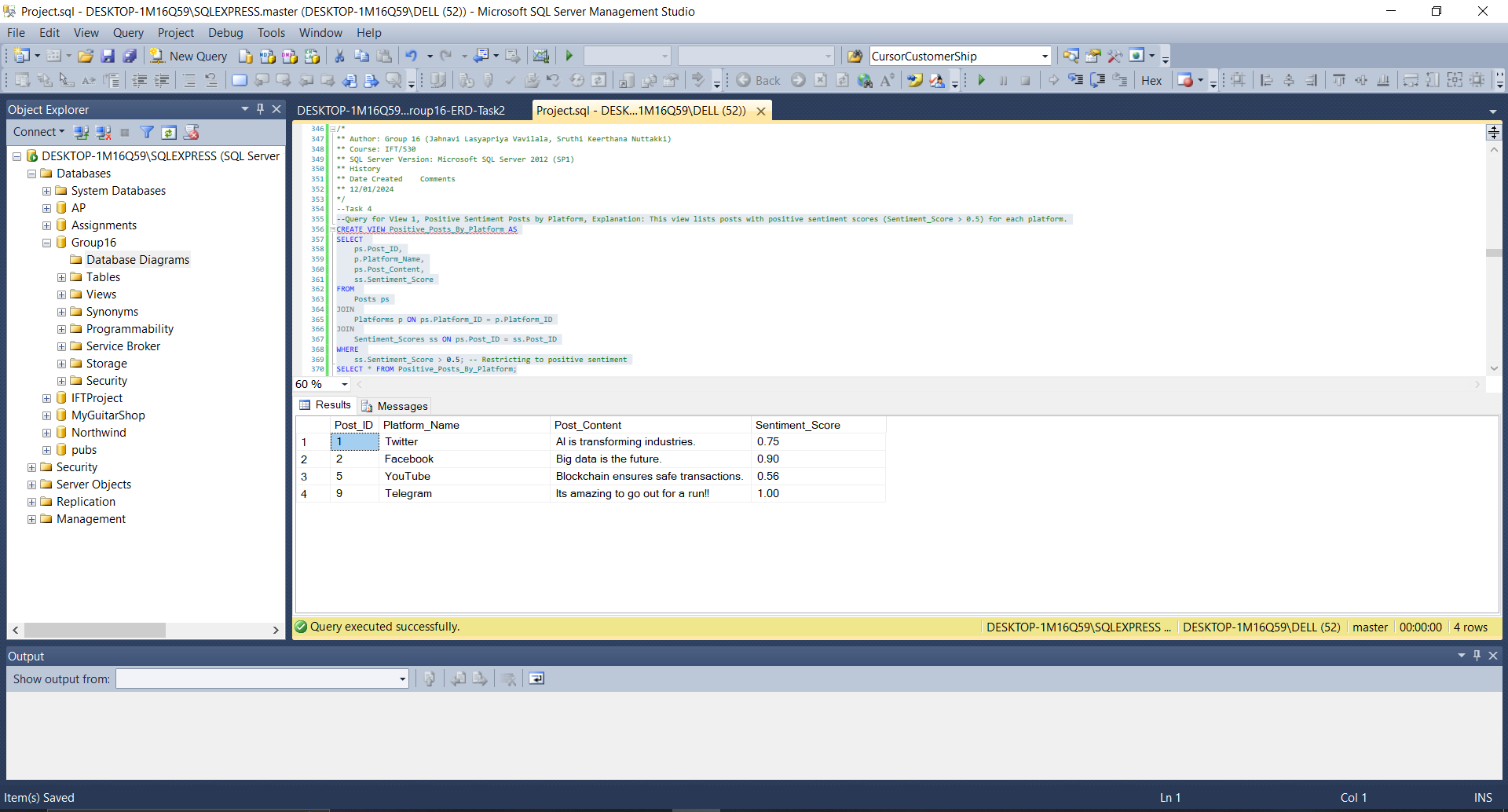
Sentiment\_Scores ss ON ps.Post\_ID = ss.Post\_ID

WHERE

ss.Sentiment\_Score > 0.5; -- Restricting to positive sentiment

SELECT \* FROM Positive\_Posts\_By\_Platform;

**Screenshot with results:**



**View 2: Active Users with High Engagement**

Purpose: This view identifies users whose posts have high engagement rates (Engagement\_Rate > 10.0). It can be used to recognize and target active or influential users for partnerships or deeper engagement strategies.

**Code:**

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

--Query for View 2, Active Users with High Engagement, Explanation: This view identifies users whose posts have high engagement rates (Engagement\_Rate > 10.0)

CREATE VIEW Active\_Users\_With\_Engagement AS

SELECT

u.Username,

SUM(pa.Likes\_Count) AS Total\_Likes,

SUM(pa.Shares\_Count) AS Total\_Shares,

SUM(pa.Comments\_Count) AS Total\_Comments

FROM

Users u

JOIN

Posts ps ON u.User\_ID = ps.User\_ID

JOIN

Post\_Analytics pa ON ps.Post\_ID = pa.Post\_ID

WHERE

pa.Engagement\_Rate > 10.0 -- Restricting to high engagement posts

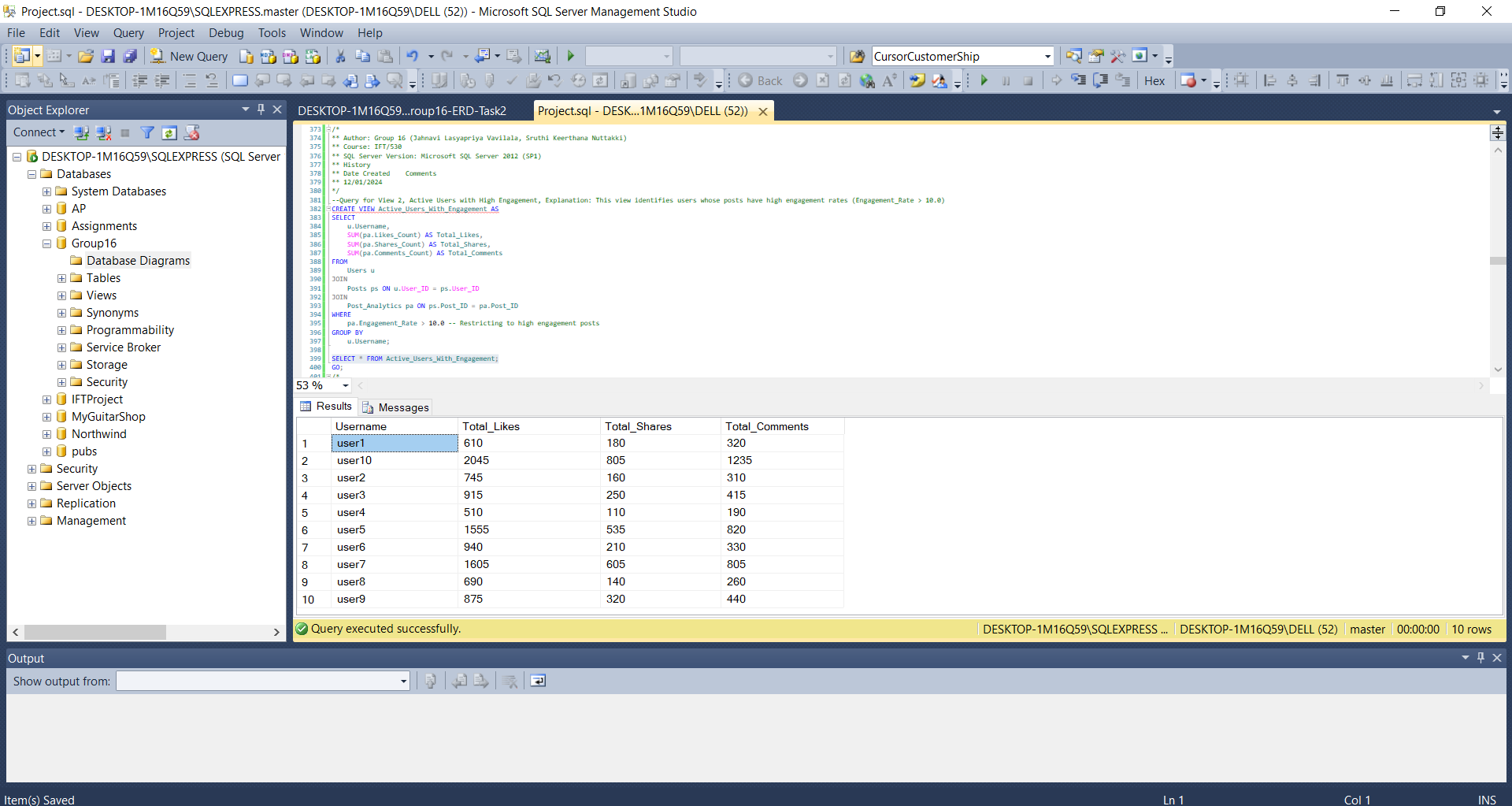
GROUP BY

u.Username;

SELECT \* FROM Active\_Users\_With\_Engagement;

GO;

**Screenshot with Results:**



**View 3: Active\_User\_Insights**

Purpose: The Active\_User\_Insights view highlights the activity of users who have posted within the last year. It combines information from multiple tables to display the username, location, total number of posts, average sentiment score, and the maximum views for any of their posts. This view is particularly useful for identifying active and influential users across different locations, providing actionable insights into user engagement and sentiment trends.

**Code:**

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

--Query for View 3, Active\_User\_Insights, Explanation:This view provides insights into user activity by showing their total posts, average sentiment score, and maximum views on any post.

CREATE VIEW Active\_User\_Insights AS

SELECT

u.Username,

l.Location\_Name,

COUNT(p.Post\_ID) AS Total\_Posts,

AVG(ss.Sentiment\_Score) AS Avg\_Sentiment,

MAX(pa.Views\_Count) AS Max\_Views

FROM

Users u

JOIN

Posts p ON u.User\_ID = p.User\_ID

JOIN

Locations l ON u.Location\_ID = l.Location\_ID

JOIN

Sentiment\_Scores ss ON p.Post\_ID = ss.Post\_ID

JOIN

Post\_Analytics pa ON p.Post\_ID = pa.Post\_ID

WHERE

p.Post\_Date >= DATEADD(YEAR, -1, GETDATE())

GROUP BY

u.Username, l.Location\_Name;

SELECT \* FROM Active\_User\_Insights

GO

**Screenshot with results:**



**Task 5: Create an audit table for one of the lookup tables and demonstrate data saved to that audit table when data in the original table is inserted, modified, or deleted. Include an additional column in the audit table that will have a datetime field when the data was changed in the original table. Include the script to test all the operations.**

**Objective:**

The purpose of this task is to create an audit table for the Hashtags table, which tracks the INSERT, UPDATE, and DELETE operations performed on the original table. Additionally, the audit table will include a Change\_Timestamp column that records the date and time when each change occurred.

**Code:**

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- Step 1: Dropping the existing tables

IF OBJECT\_ID('Hashtags\_Audit', 'U') IS NOT NULL

DROP TABLE Hashtags\_Audit;

GO

IF OBJECT\_ID('Hashtags', 'U') IS NOT NULL

DROP TABLE Hashtags;

GO

-- Step 2: Create the Hashtags Table

CREATE TABLE Hashtags (

Hashtag\_ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY, -- Auto-incrementing ID

Hashtag\_Text NVARCHAR(100) NOT NULL UNIQUE -- Unique hashtag text

);

GO

-- Step 3: Create the Hashtags\_Audit Table

CREATE TABLE Hashtags\_Audit (

Audit\_ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY, -- Unique identifier for each audit entry

Hashtag\_ID INT NOT NULL, -- Reference to the original Hashtag\_ID

Operation\_Type NVARCHAR(10) NOT NULL, -- Type of operation: 'INSERT', 'UPDATE', 'DELETE'

Hashtag\_Text NVARCHAR(100), -- The hashtag text being audited

Change\_Timestamp DATETIME DEFAULT GETDATE() -- Timestamp of the operation

);

GO

-- Step 4: Create Triggers for the Hashtags Table

-- Trigger for INSERT Operations

CREATE TRIGGER trg\_Hashtags\_Insert

ON Hashtags

AFTER INSERT

AS

BEGIN

INSERT INTO Hashtags\_Audit (Hashtag\_ID, Operation\_Type, Hashtag\_Text, Change\_Timestamp)

SELECT

inserted.Hashtag\_ID,

'INSERT',

inserted.Hashtag\_Text,

GETDATE()

FROM

inserted;

END;

GO

-- Trigger for UPDATE Operations

CREATE TRIGGER trg\_Hashtags\_Update

ON Hashtags

AFTER UPDATE

AS

BEGIN

INSERT INTO Hashtags\_Audit (Hashtag\_ID, Operation\_Type, Hashtag\_Text, Change\_Timestamp)

SELECT

inserted.Hashtag\_ID,

'UPDATE',

inserted.Hashtag\_Text,

GETDATE()

FROM

inserted;

END;

GO

-- Trigger for DELETE Operations

CREATE TRIGGER trg\_Hashtags\_Delete

ON Hashtags

AFTER DELETE

AS

BEGIN

INSERT INTO Hashtags\_Audit (Hashtag\_ID, Operation\_Type, Hashtag\_Text, Change\_Timestamp)

SELECT

deleted.Hashtag\_ID,

'DELETE',

deleted.Hashtag\_Text,

GETDATE()

FROM

deleted;

END;

GO

-- Step 5: Test the Triggers

-- Insert test data into Hashtags

INSERT INTO Hashtags (Hashtag\_Text) VALUES

('#AI'),

('#BigData'),

('#CloudComputing');

GO

-- Check the audit table after insert

SELECT \* FROM Hashtags\_Audit;

GO

-- Update test data in the Hashtags table

UPDATE Hashtags

SET Hashtag\_Text = '#ArtificialIntelligence'

WHERE Hashtag\_ID = 1;

GO

-- Check the audit table after update

SELECT \* FROM Hashtags\_Audit;

GO

-- Delete test data from Hashtags table

DELETE FROM Hashtags

WHERE Hashtag\_ID = 2;

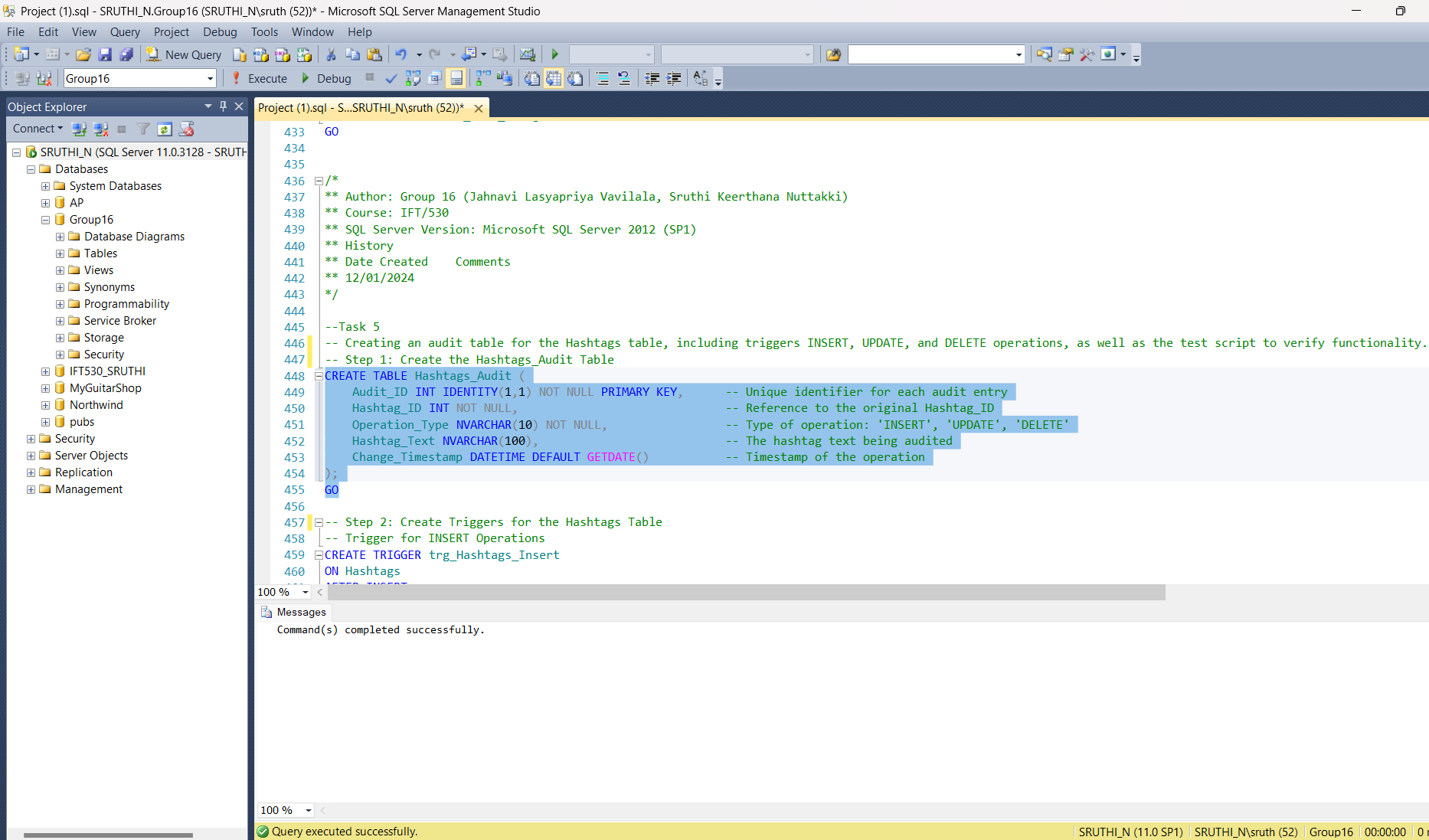
GO

-- Check the audit table after delete

SELECT \* FROM Hashtags\_Audit;

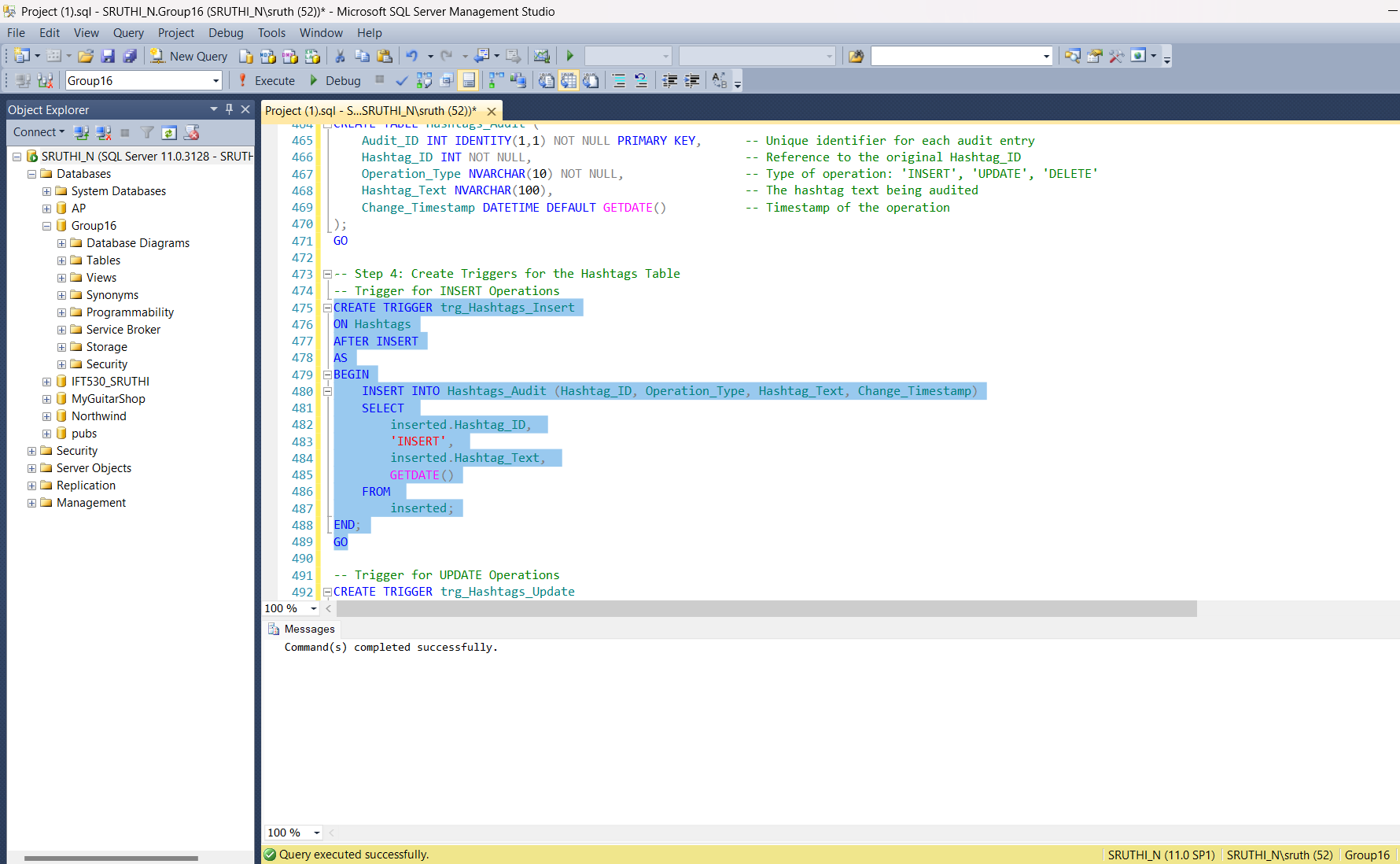
GO

**5.1 Created Hashtag\_Audits table:**

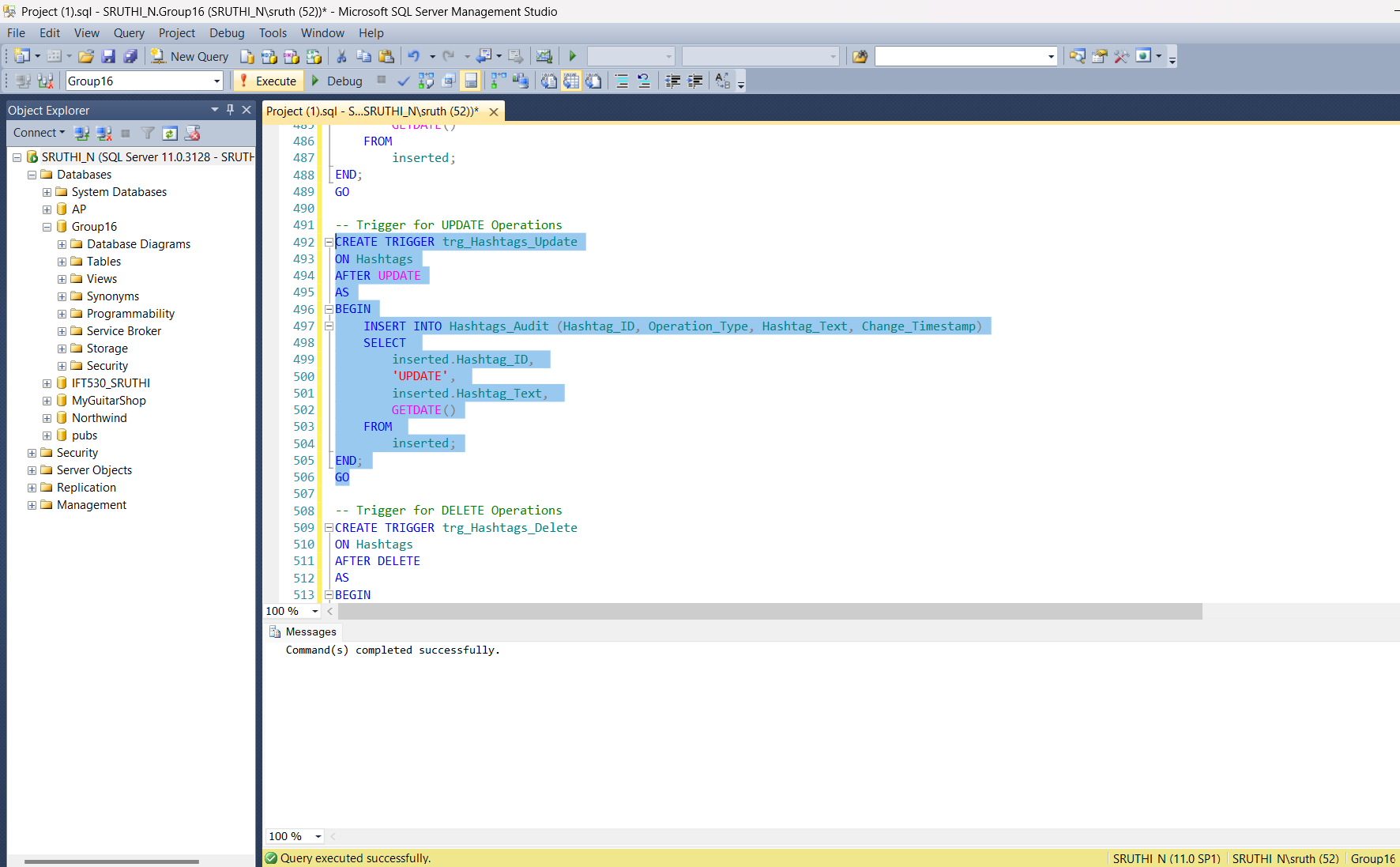
****

**5.2 Create Triggers for INSERT, UPDATE, DELETE Operations**

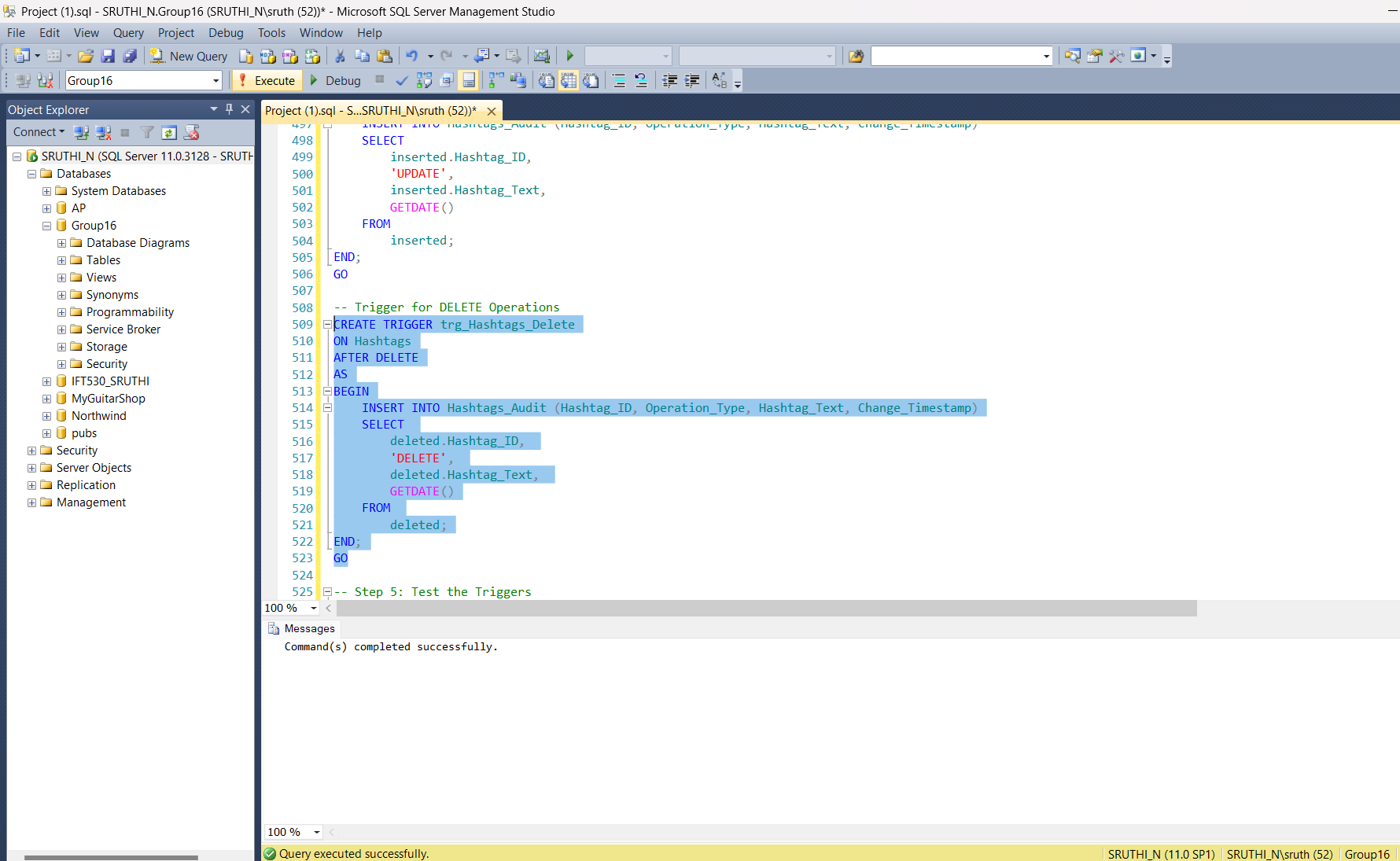
**Trigger for INSERT Operation**

****

**Trigger for UPDATE Operation**

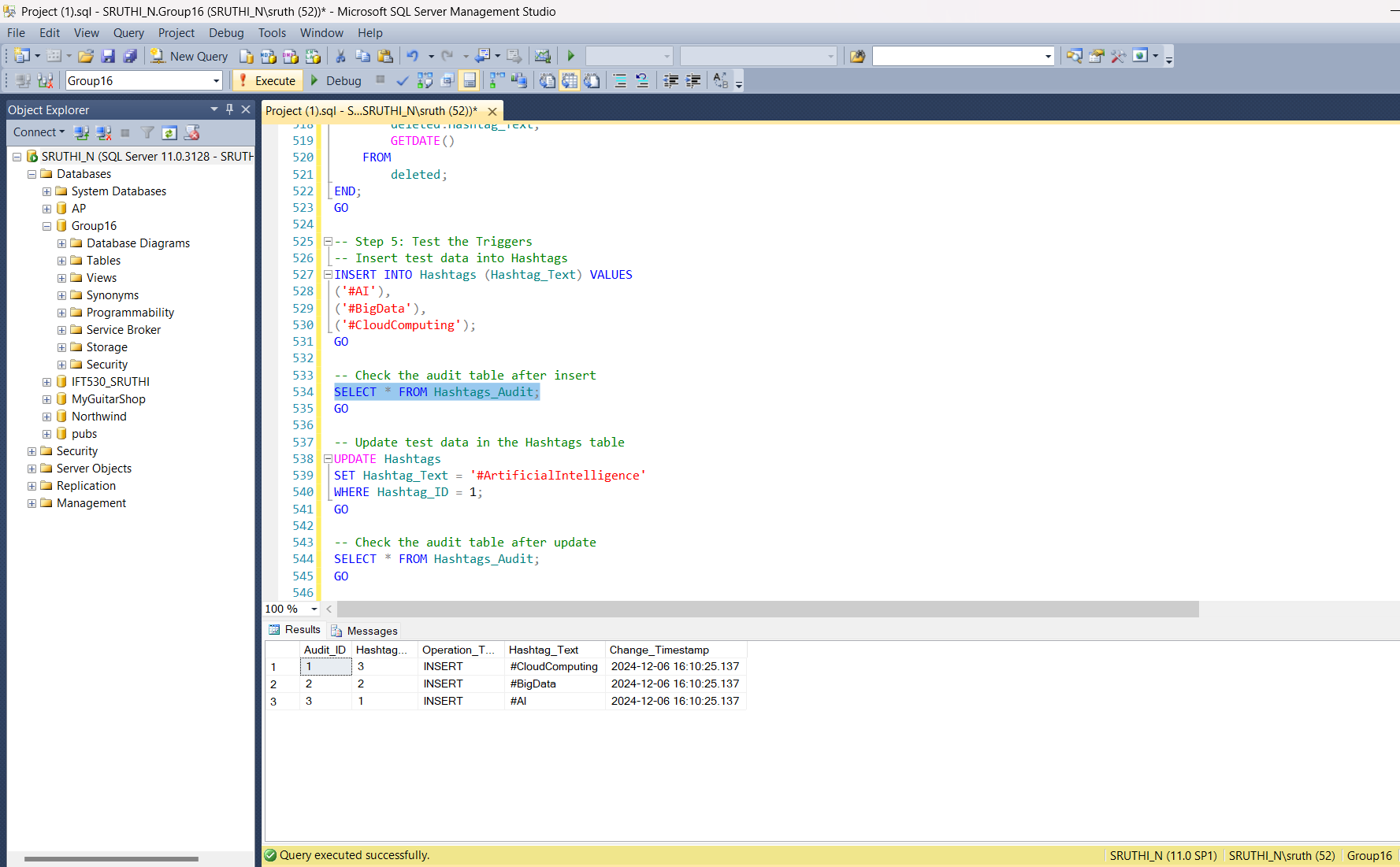
****

**Trigger for DELETE Operation**

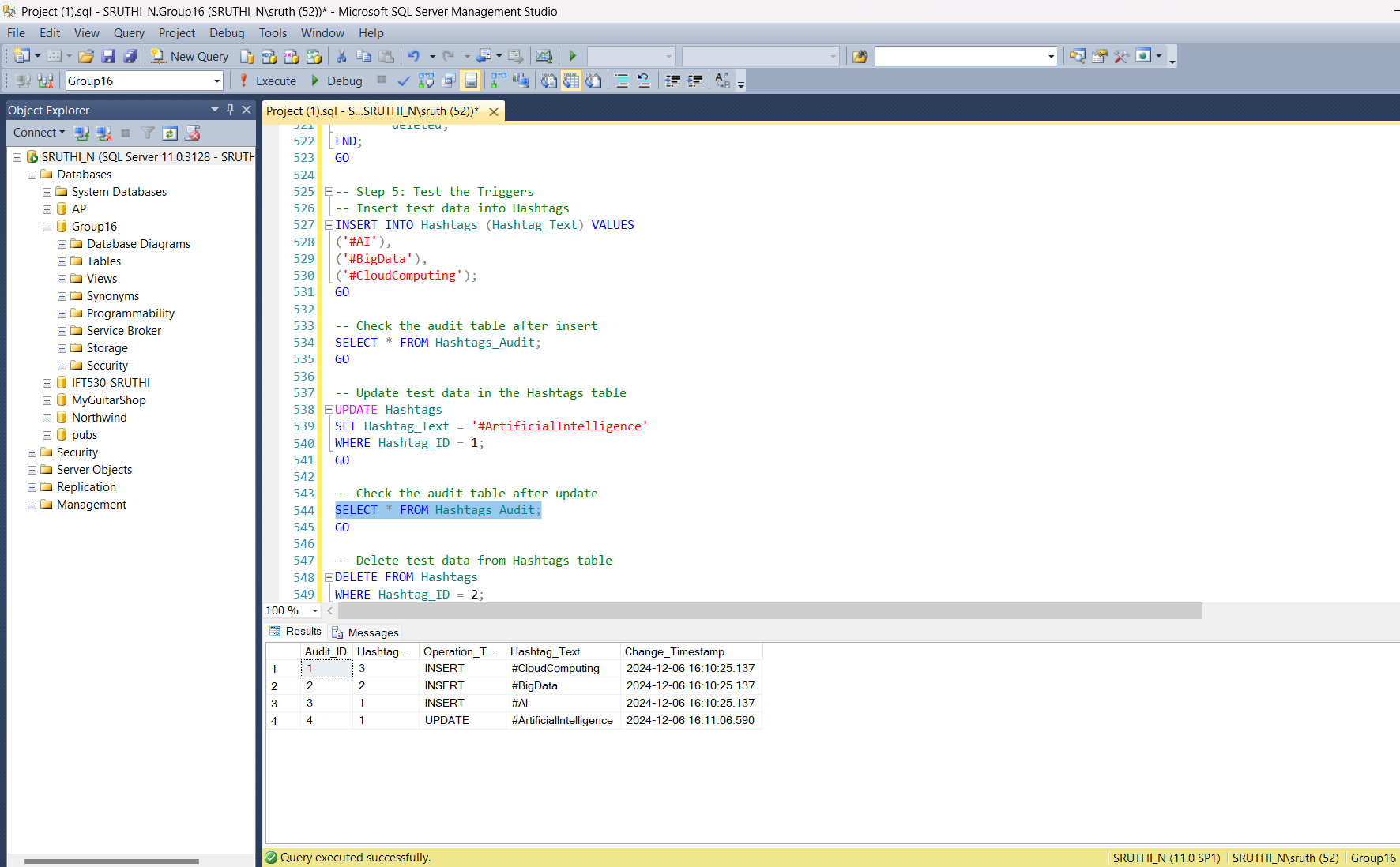
****

**5.3 Testing the triggers**

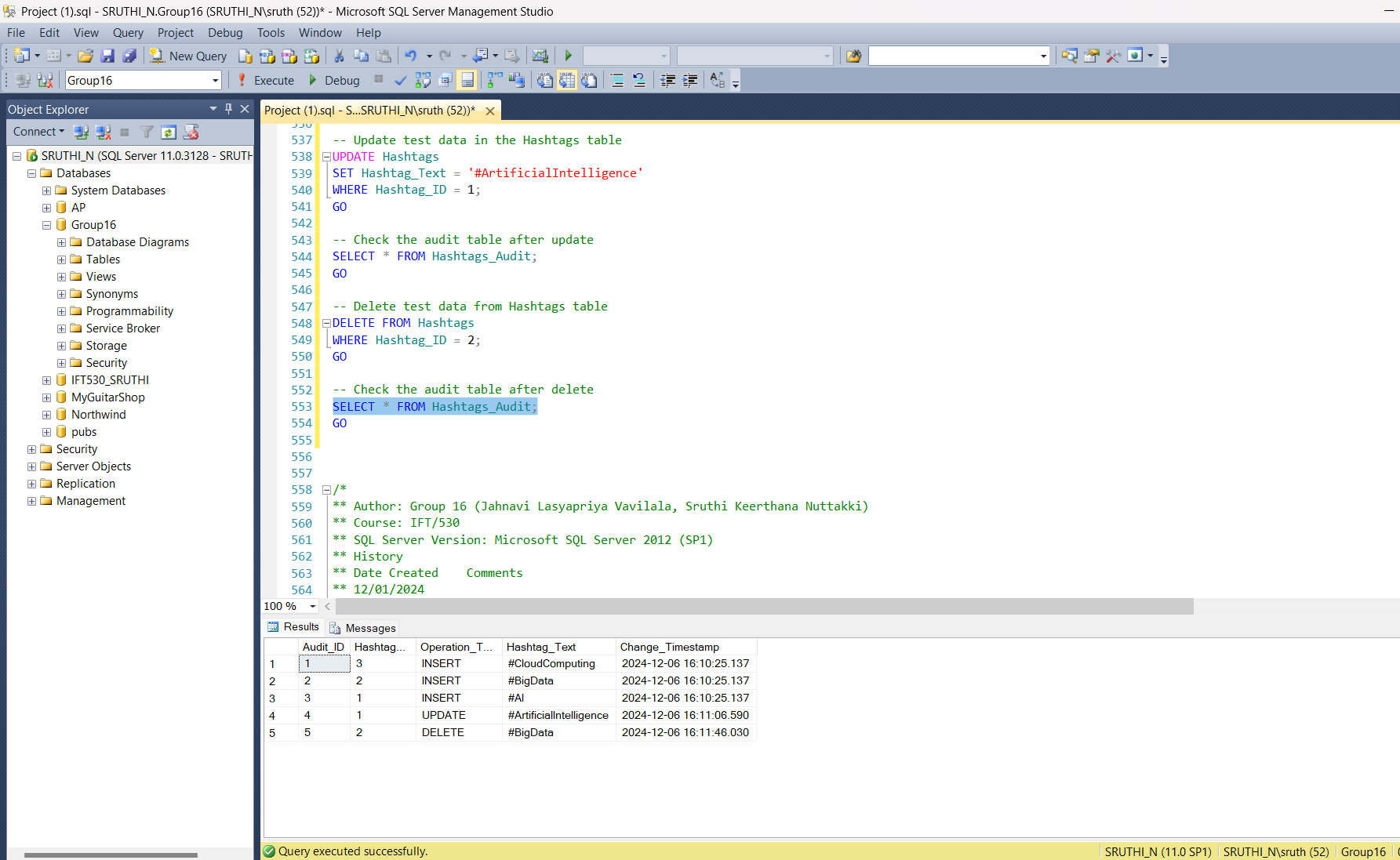
**Inserted the test data into Hashtags and checked the audit table after insertion.**

****

**Updated the test data in the Hashtags table and checked the audit table after the update.**

****

**Deleted the test data from the Hashtags table and checked the audit table after the deletion.**

****

**Task 6: Demonstrate a use of the one stored procedures and User Defined Function (UDF) for your database. Include create and drop scripts.**

**6.1 Stored Procedure**

**Purpose:** A stored procedure to fetch all hashtags with a specified keyword in the Hashtag\_Text.

**Code:**

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

--1. Stored Procedure

--Purpose: A stored procedure to fetch all hashtags with a specified keyword in the Hashtag\_Text.

--Create Stored Procedure:

-- Drop the stored procedure if it exists

IF OBJECT\_ID('GetHashtagsByKeyword', 'P') IS NOT NULL

DROP PROCEDURE GetHashtagsByKeyword;

GO

-- Create the stored procedure

CREATE PROCEDURE GetHashtagsByKeyword

@Keyword NVARCHAR(50)

AS

BEGIN

SELECT Hashtag\_ID, Hashtag\_Text

FROM Hashtags

WHERE Hashtag\_Text LIKE '%' + @Keyword + '%';

END;

GO

-- Example 1: Fetch hashtags containing 'AI'

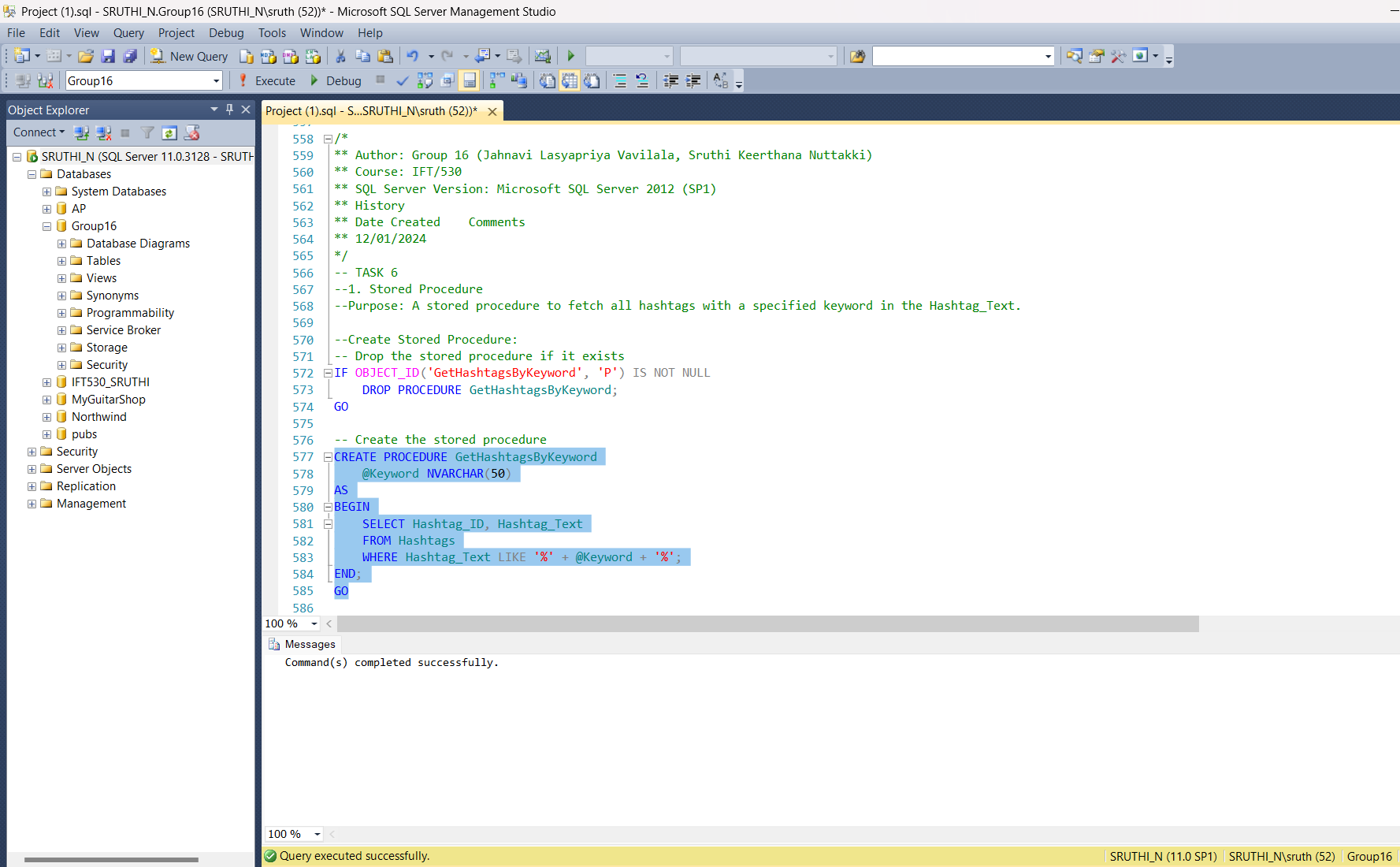
EXEC GetHashtagsByKeyword @Keyword = 'CloudComputing';

-- Drop the stored procedure

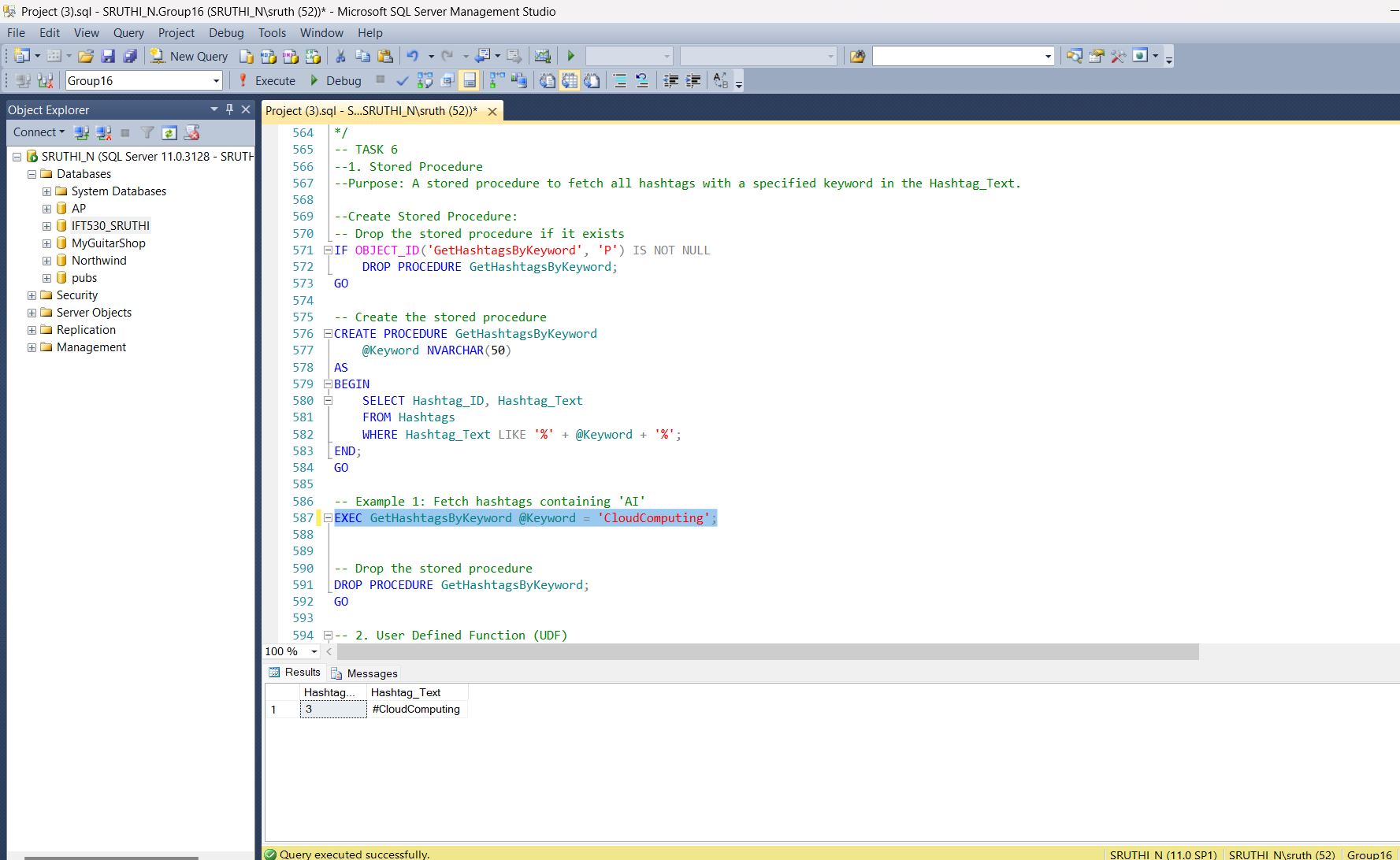
DROP PROCEDURE GetHashtagsByKeyword;

GO

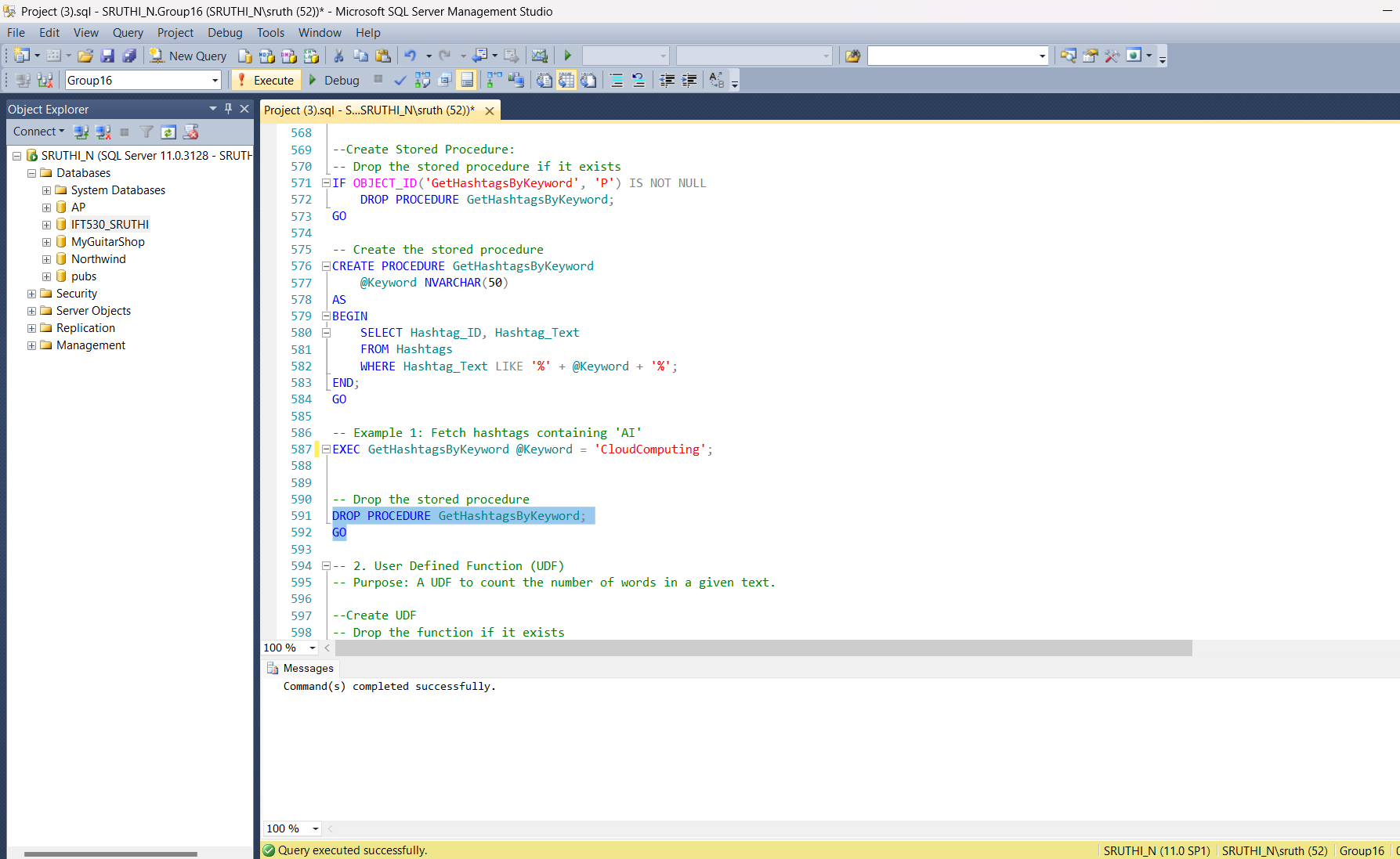
**Created the stored procedure.**

****

**Test case: Fetching hashtags containing ‘CloudComputing’**

****

**Dropping the Stored Procedure.**

****

**2. User Defined Function (UDF)**

**Purpose: A UDF to calculate the engagement rate based on likes, shares, and views.**

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

-- 2. User Defined Function (UDF)

-- Purpose: A UDF to count the number of words in a given text.

--Create UDF

-- Drop the function if it exists

IF OBJECT\_ID('CalculateEngagementRate', 'FN') IS NOT NULL

DROP FUNCTION CalculateEngagementRate;

GO

-- Create the function

CREATE FUNCTION CalculateEngagementRate

(@Likes INT, @Shares INT, @Views INT)

RETURNS DECIMAL(5, 2)

AS

BEGIN

RETURN CASE

WHEN @Views = 0 THEN 0.00

ELSE CAST((@Likes + @Shares) AS DECIMAL(5, 2)) / @Views \* 100

END;

END;

GO

-- Example in combination with Post\_Analytics table

SELECT Post\_ID,

dbo.CalculateEngagementRate(Likes\_Count, Shares\_Count, Views\_Count) AS EngagementRate

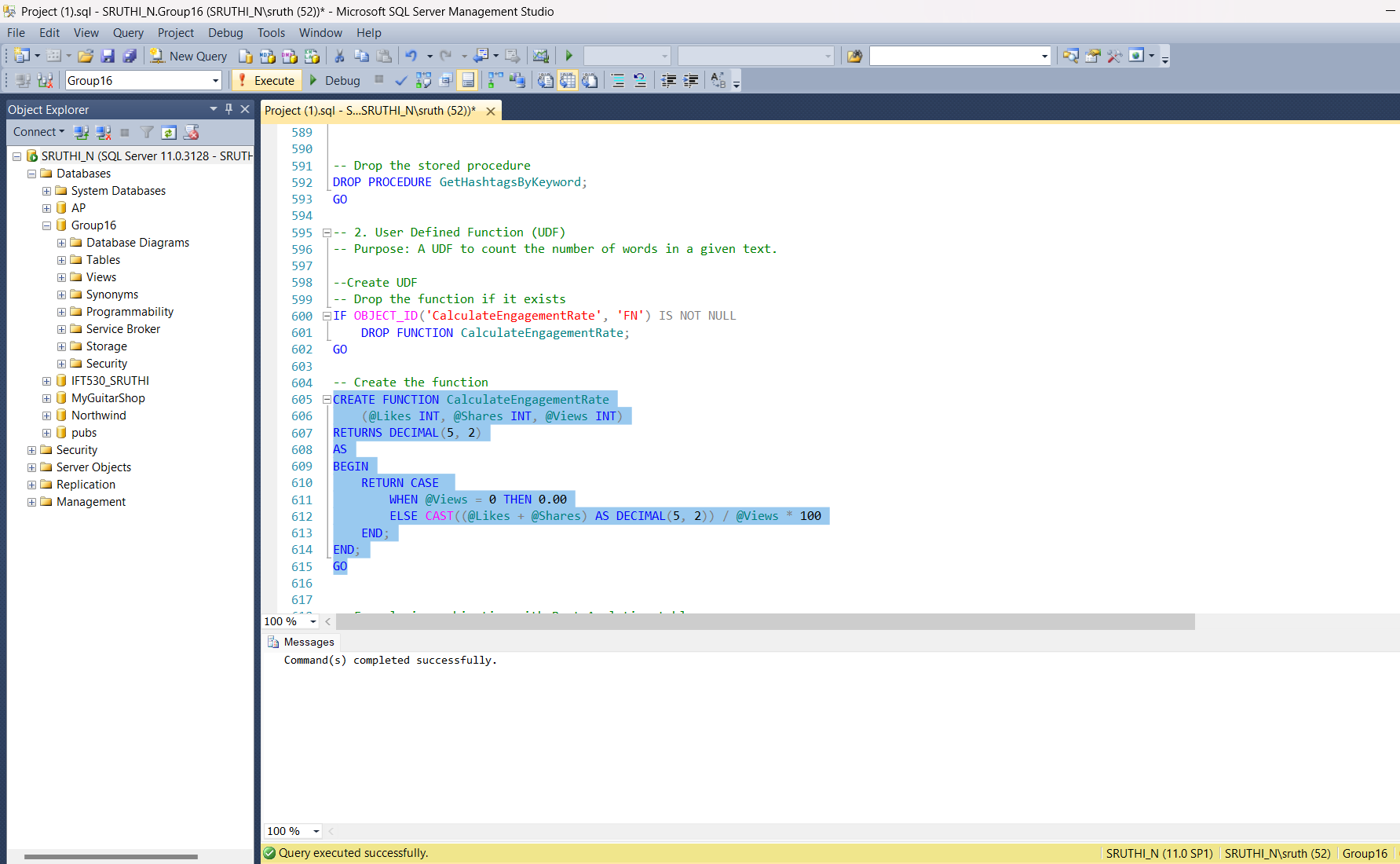
FROM Post\_Analytics;

-- Drop the function

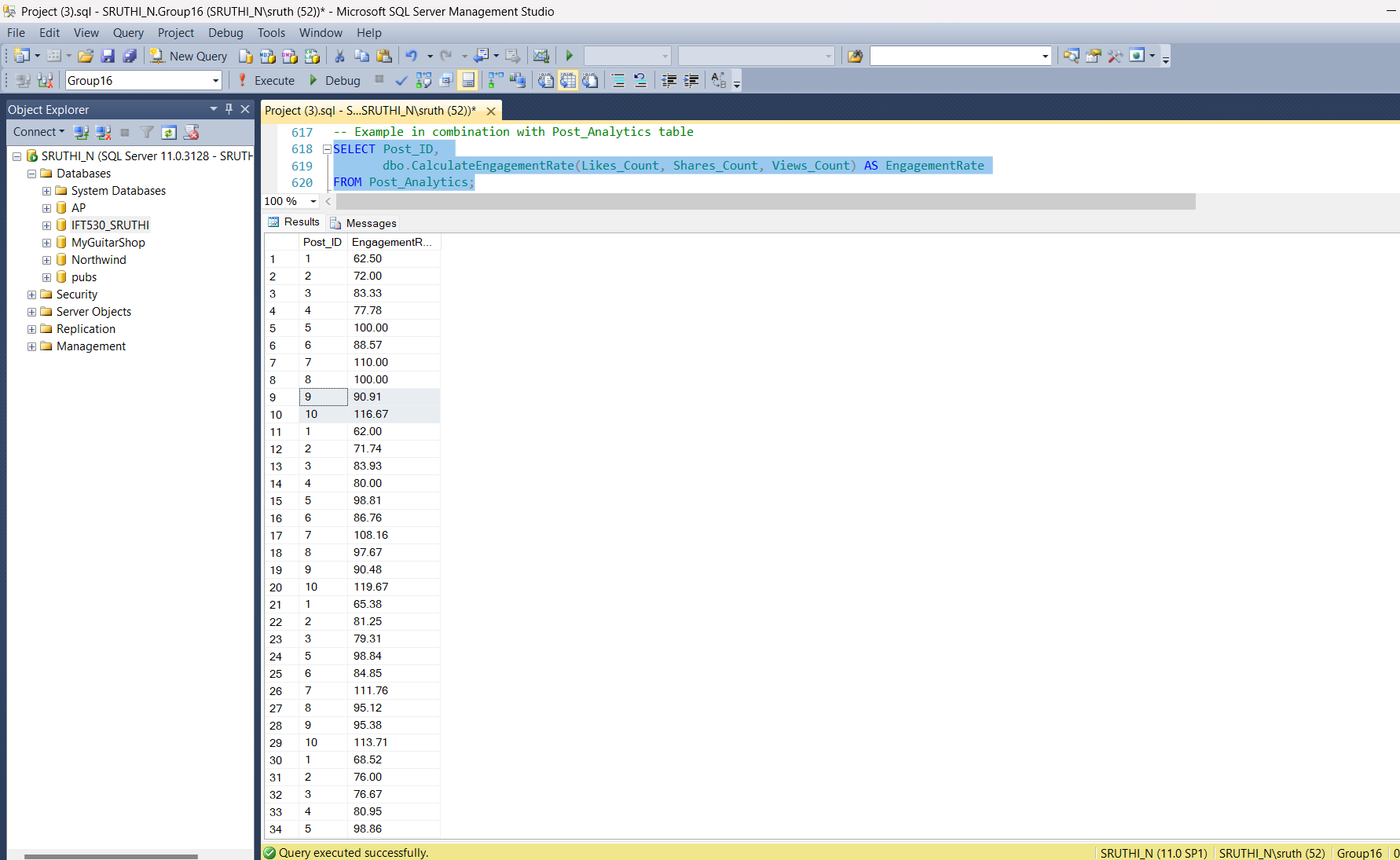
DROP FUNCTION CalculateEngagementRate;

GO

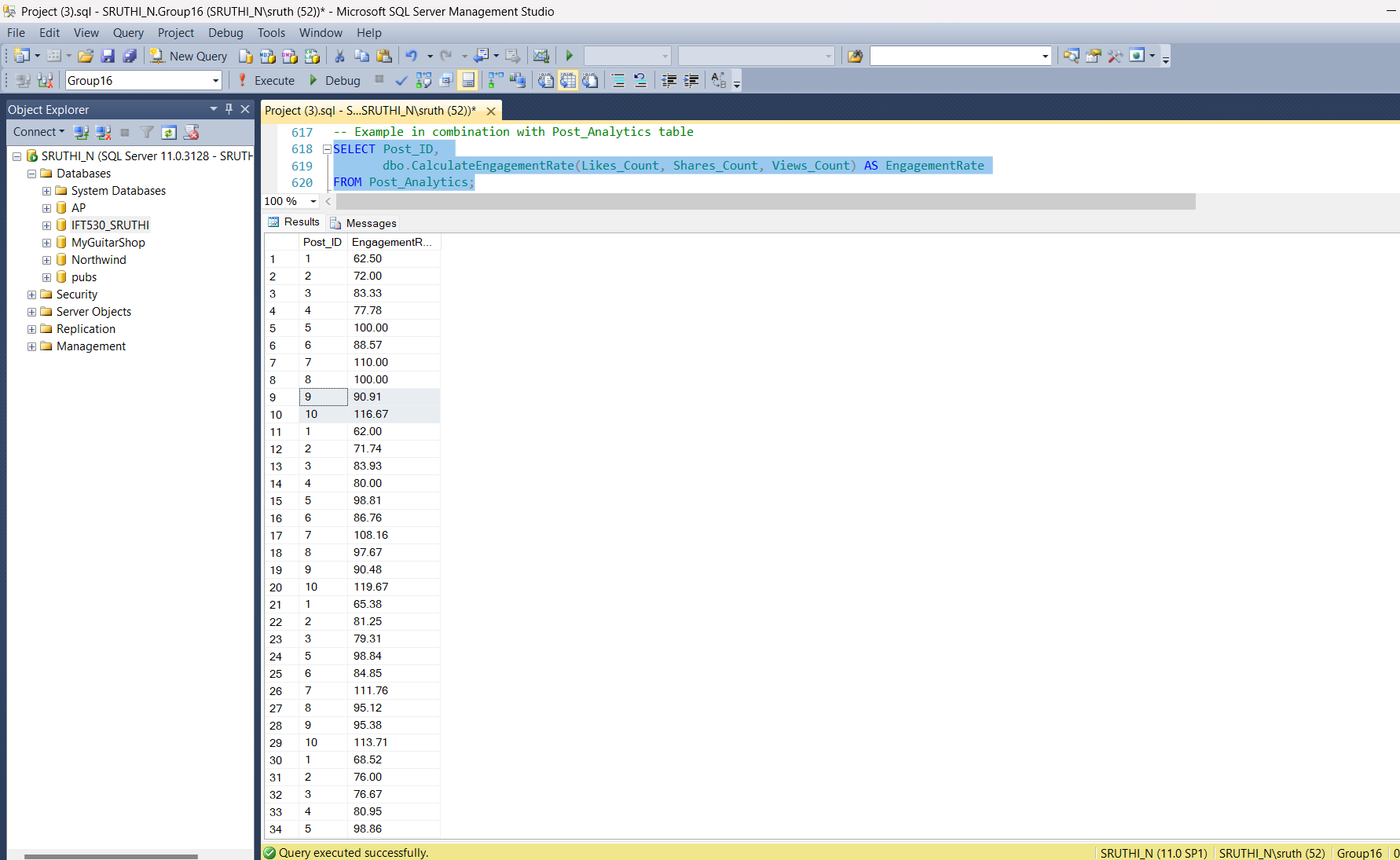
**Created the User Defined Function:**

****

**Example in Combination with Post\_Analytics Table:**

****

**Dropping the Function.**

****

**Task 7: Demonstrate the use of one cursor for your database. Create and drop script for cursor.**

#### **Purpose of the Cursor:**

The cursor is designed to process each row of data in the Sentiment\_Scores table, joined with the Posts table, to dynamically update engagement rates in the Post\_Analytics table based on sentiment thresholds. This approach is helpful when you need to apply business logic row-by-row, which cannot easily be achieved with a single set-based query.

#### **Step-by-Step Breakdown of our implementation:**

1. Variable Declaration  
   The variables @Post\_ID and @Sentiment\_Score are declared to hold values for each row fetched by the cursor. These correspond to the post's unique identifier and its associated sentiment score.
2. Cursor Declaration
   * The SentimentCursor is declared to retrieve Post\_ID and Sentiment\_Score from a join of the Sentiment\_Scores and Posts tables.
   * This join ensures only valid posts with sentiment scores are processed.
3. Cursor Operation
   * Open the Cursor: The OPEN statement initializes the cursor, making it ready to fetch rows.
   * Fetch Rows: The FETCH NEXT command retrieves the first row of data and assigns values to the variables. This process repeats until there are no more rows to fetch (@@FETCH\_STATUS = 0).
   * Business Logic: For each fetched row:
     + If the sentiment score is greater than 0.5 (positive sentiment), the engagement rate for the corresponding post is increased by 5.
     + If the sentiment score is less than -0.5 (negative sentiment), the engagement rate is decreased by 5.
     + The UPDATE statement ensures these changes are reflected in the Post\_Analytics table.
   * Iteration: The cursor iterates through the dataset row-by-row, applying the logic to each post.
4. Closing and Deallocating the Cursor
   * The CLOSE statement releases the cursor from use, and DEALLOCATE removes it from memory. This step is essential for maintaining database efficiency and avoiding resource leaks.

**Code:**

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

--Task 7

-- Declare variables to hold data

DECLARE @Post\_ID INT, @Sentiment\_Score DECIMAL(3, 2);

-- Declare the cursor

DECLARE MyCursor CURSOR FOR

SELECT Post\_ID, Sentiment\_Score

FROM Sentiment\_Scores

WHERE Sentiment\_Score <> 0; -- Only rows with a non-zero sentiment score

-- Open the cursor

OPEN MyCursor;

-- Fetch the first row

FETCH NEXT FROM MyCursor INTO @Post\_ID, @Sentiment\_Score;

-- Loop through each row

WHILE @@FETCH\_STATUS = 0

BEGIN

-- Print the fetched data (or apply business logic)

PRINT 'Post ID: ' + CAST(@Post\_ID AS VARCHAR) + ', Sentiment Score: ' + CAST(@Sentiment\_Score AS VARCHAR);

-- Fetch the next row

FETCH NEXT FROM MyCursor INTO @Post\_ID, @Sentiment\_Score;

END;

-- Close and deallocate the cursor

CLOSE MyCursor;

DEALLOCATE MyCursor;

/\*

\*\* Author: Group 16 (Jahnavi Lasyapriya Vavilala, Sruthi Keerthana Nuttakki)

\*\* Course: IFT/530

\*\* SQL Server Version: Microsoft SQL Server 2012 (SP1)

\*\* History

\*\* Date Created Comments

\*\* 12/01/2024

\*/

--Drop Cursor Script

-- Close and deallocate cursor if it exists

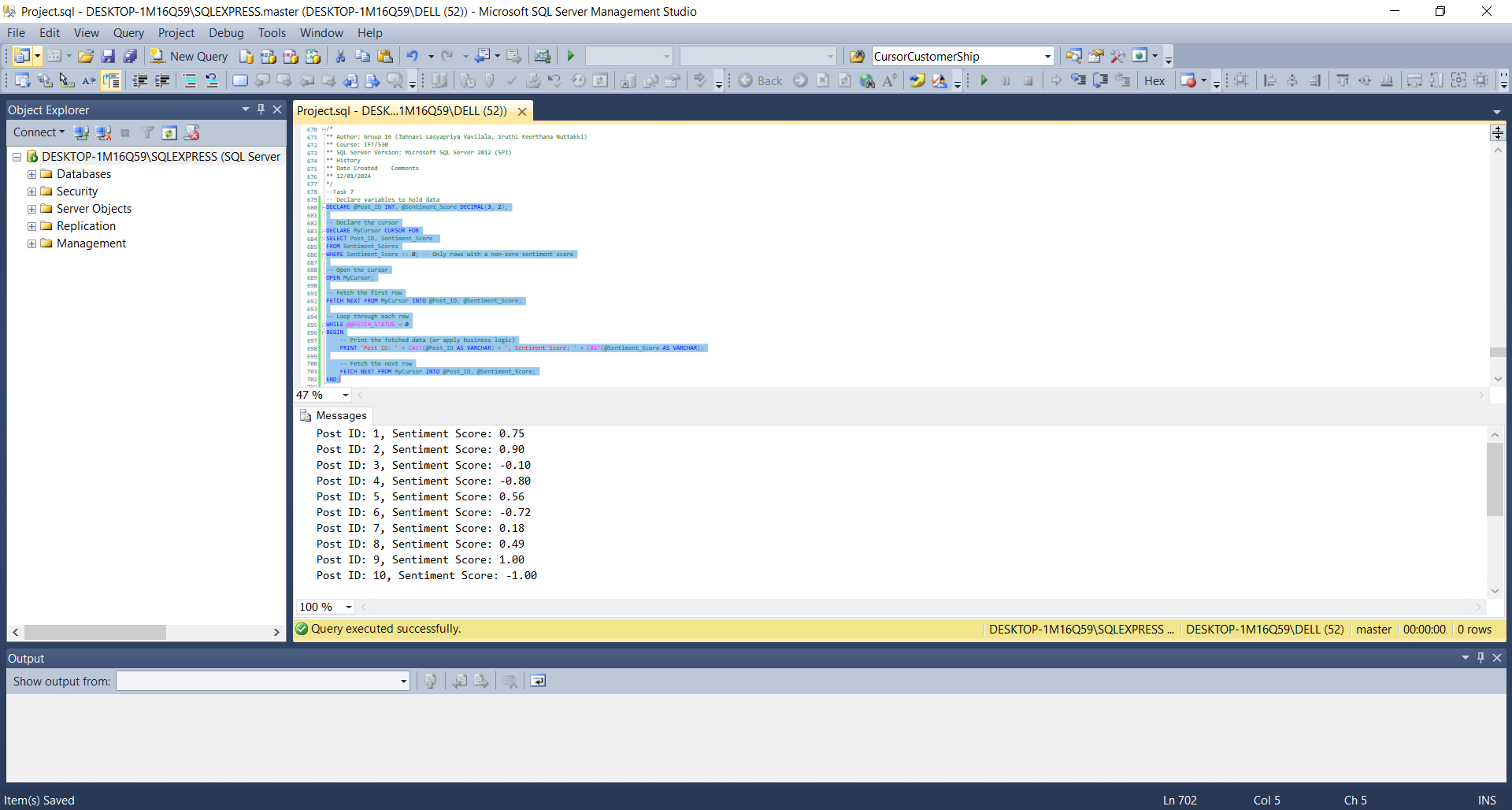
IF CURSOR\_STATUS('global', 'SentimentCursor') >= 0

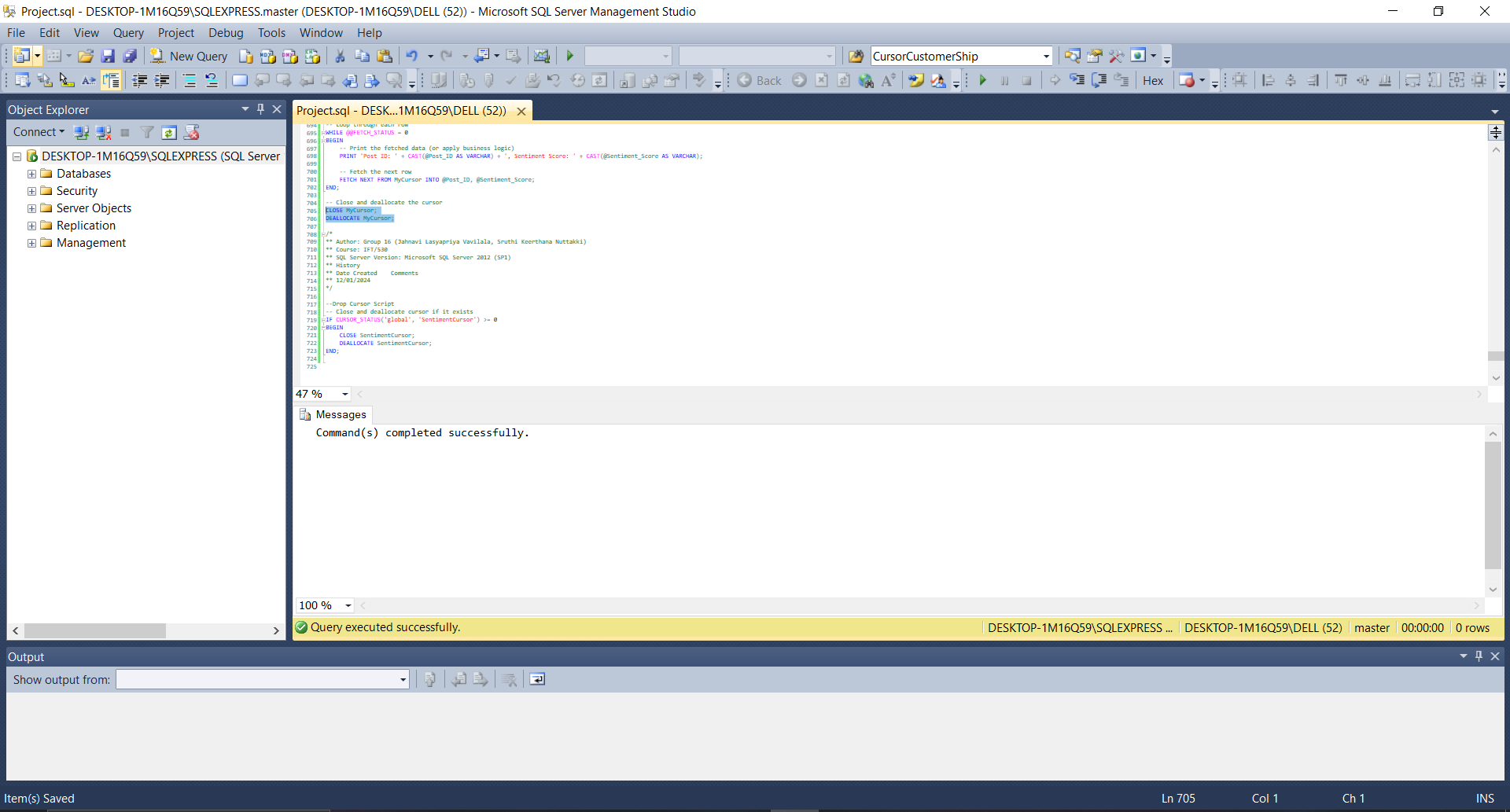
BEGIN

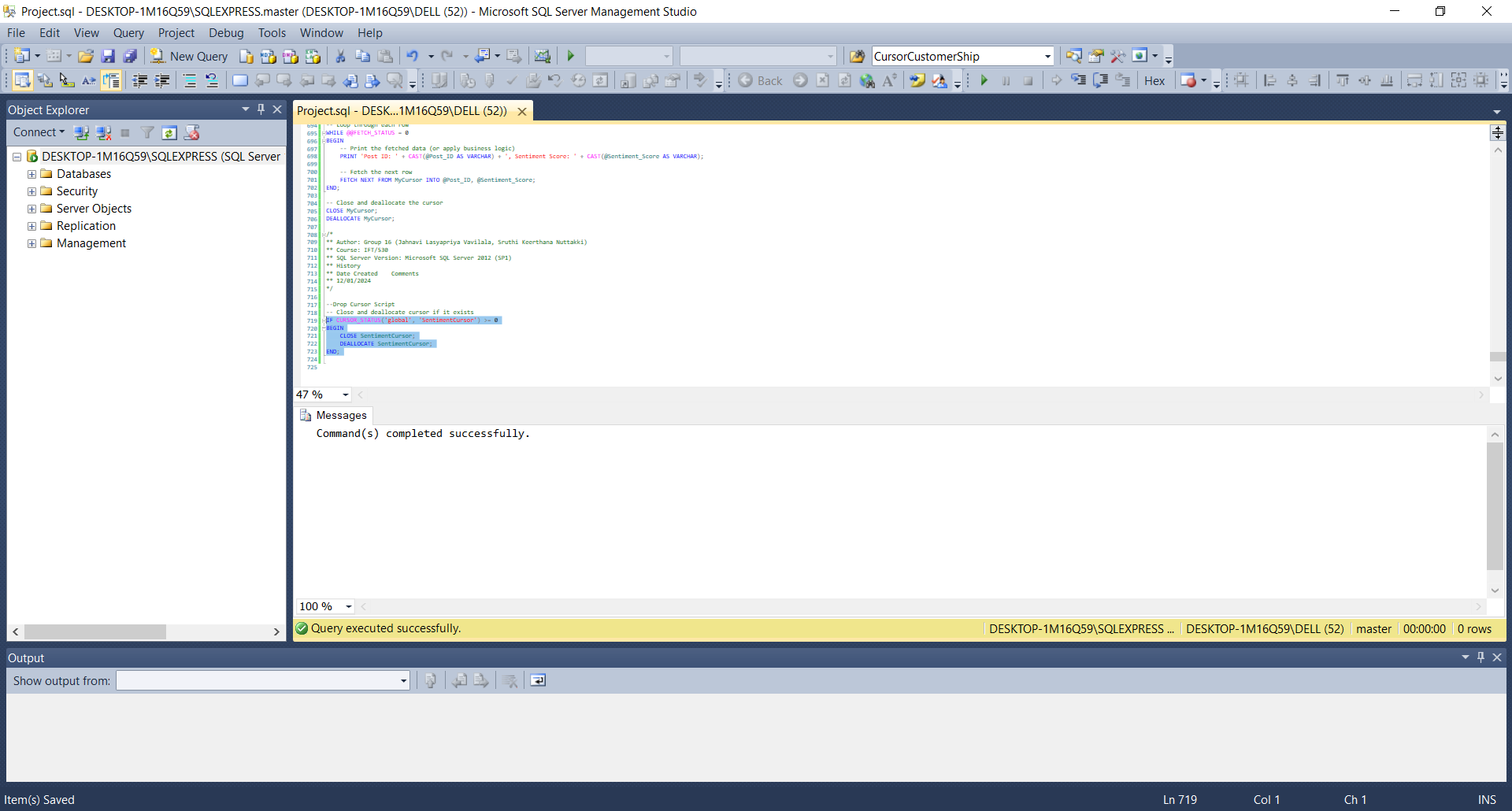
CLOSE SentimentCursor;

DEALLOCATE SentimentCursor;

END;

****

****

****