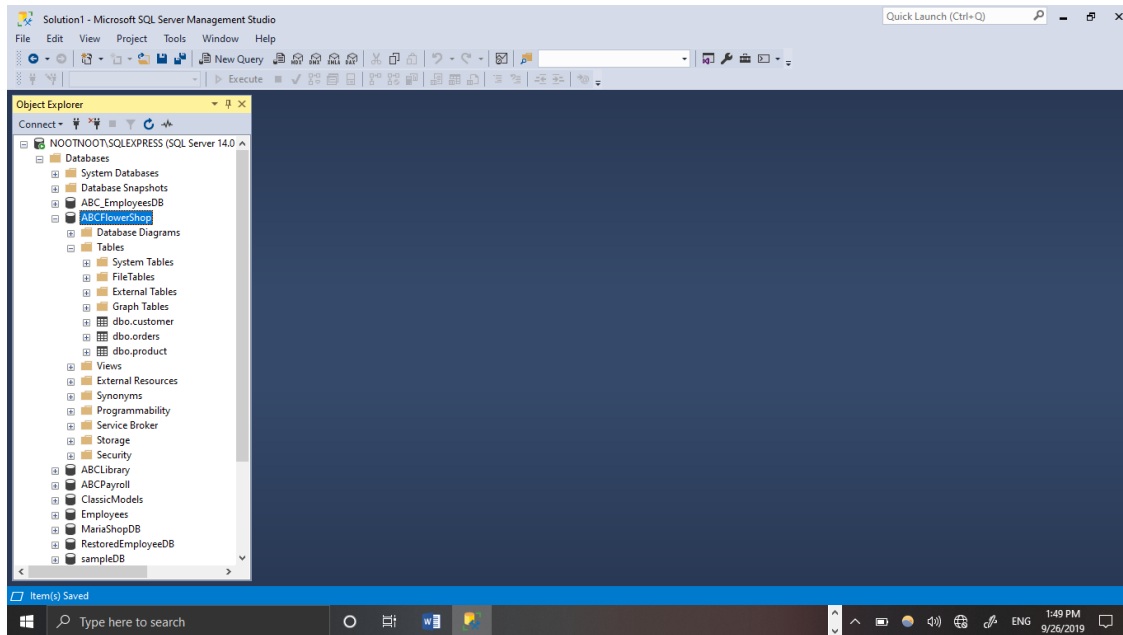


Activity #14 – Transactions	
Name: Cabral, Jose Paulo C.	Date: 09/26/19
Course/Section: CPE011/CPE21FB1	Instructor: Engr. Royce Chua

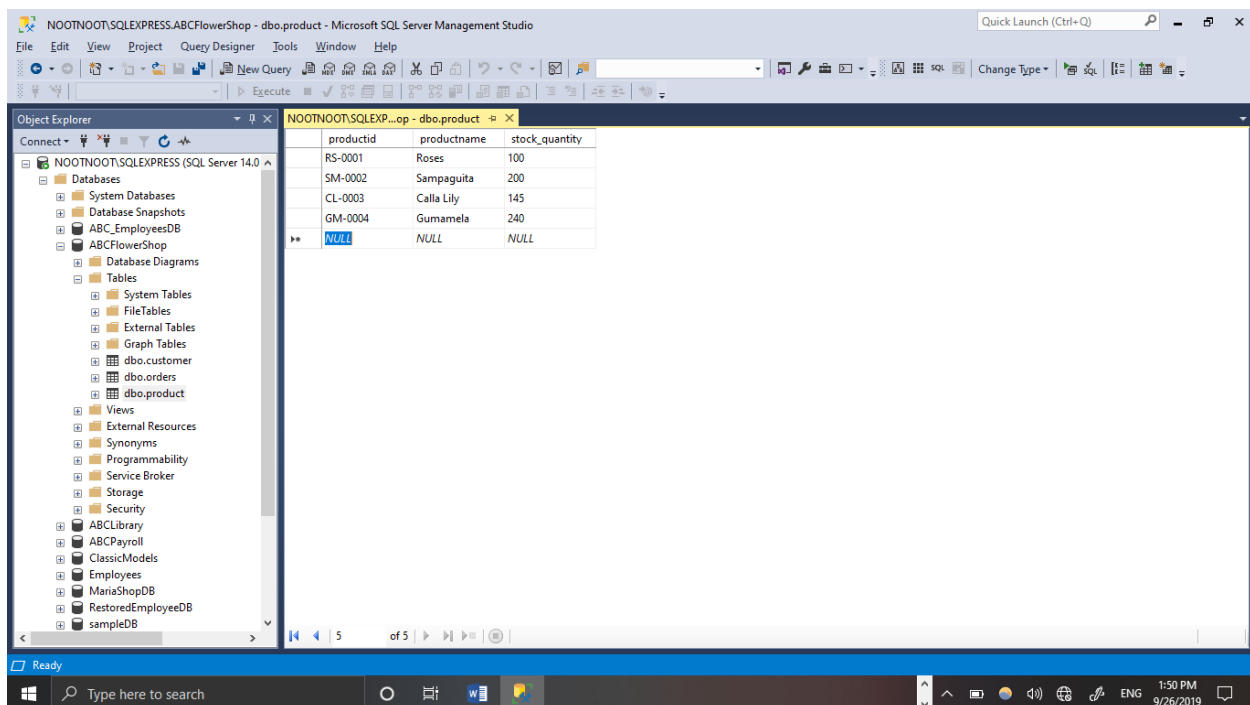
Procedures

Create ABCFlowershop database



ABCFlowershop database is defined with the tables: customer, orders, and product.

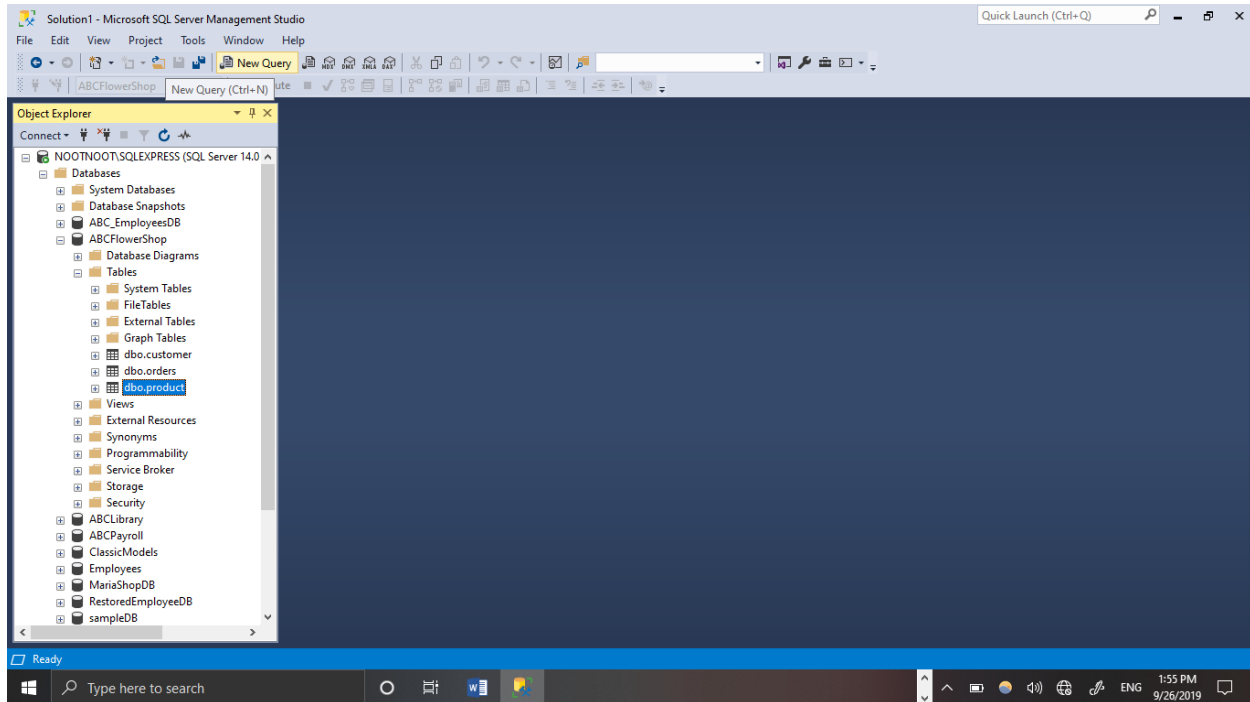
Populated products table



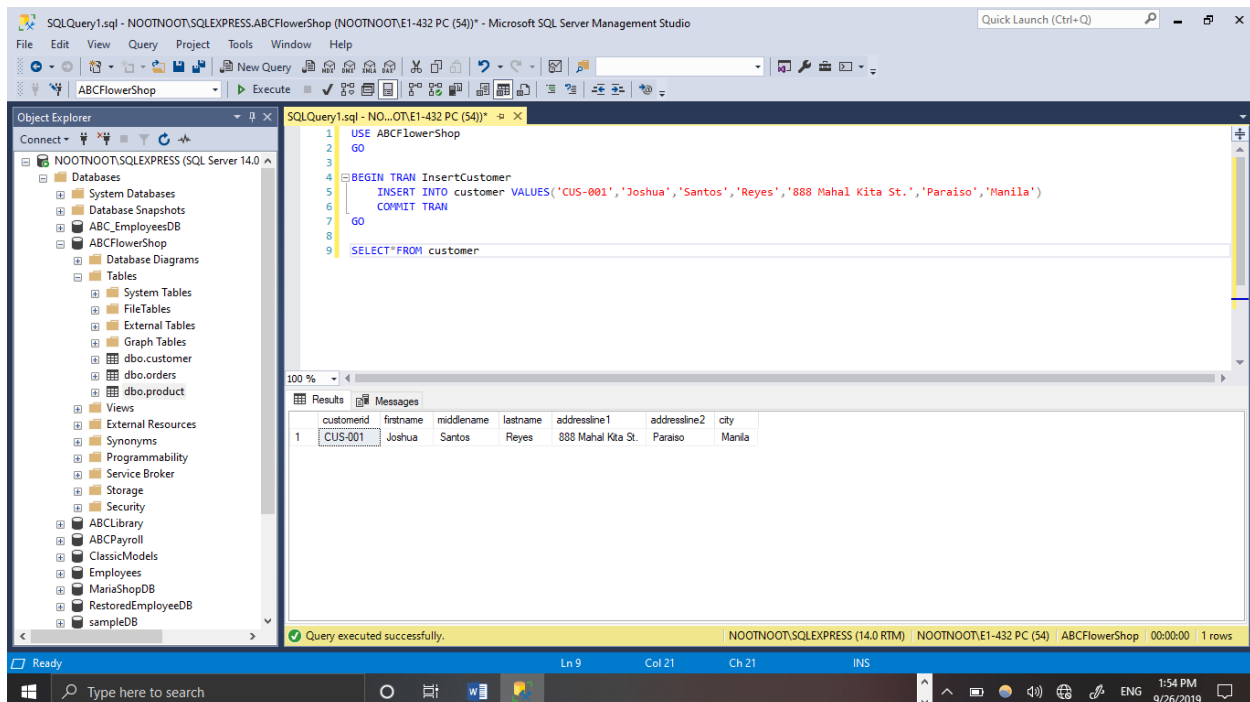
The products table is filled with unique entries.

COMMIT TRANSACTION

A. Using T-SQL

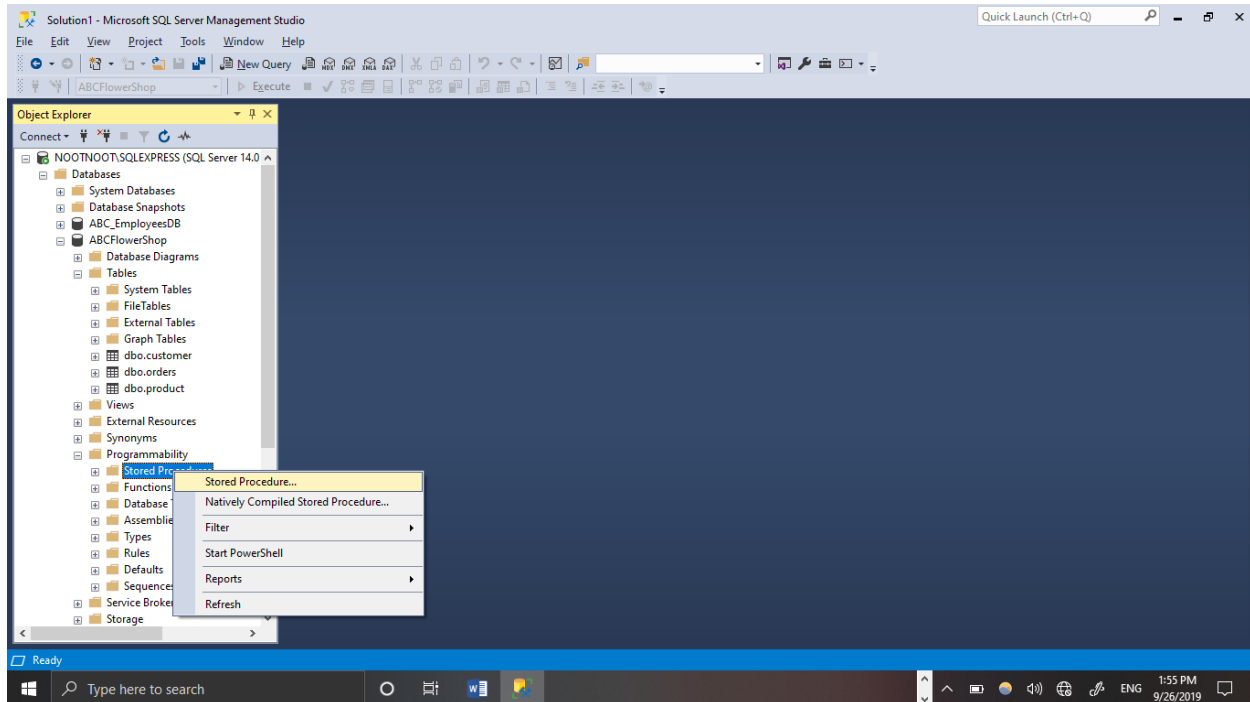


When executing raw SQL commands, the New Query option can be selected to perform T-SQL statements.

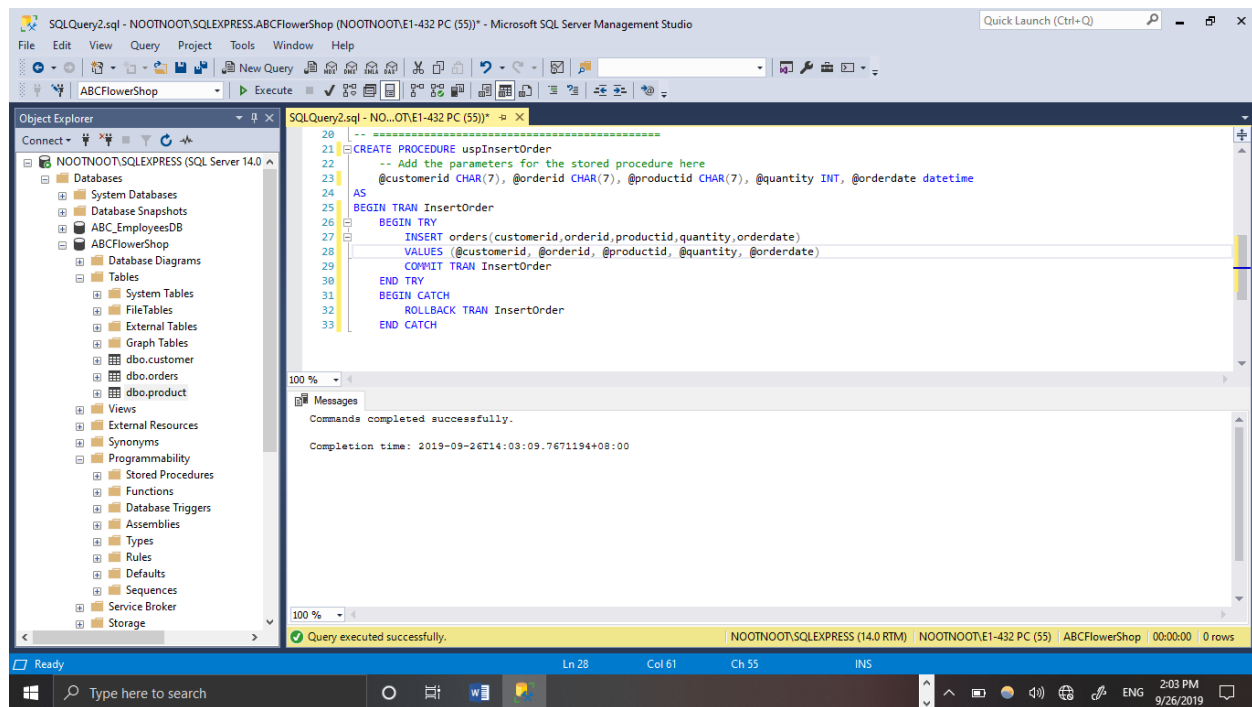


In this scenario, the execution of a transaction was recognized in between the BEGIN TRAN command and the COMMIT TRAN command. These commands define the SQL statement involved in a transaction. In this case, the transaction made was for the INSERT statement. The COMMIT command enables the SQL statement to pass through.

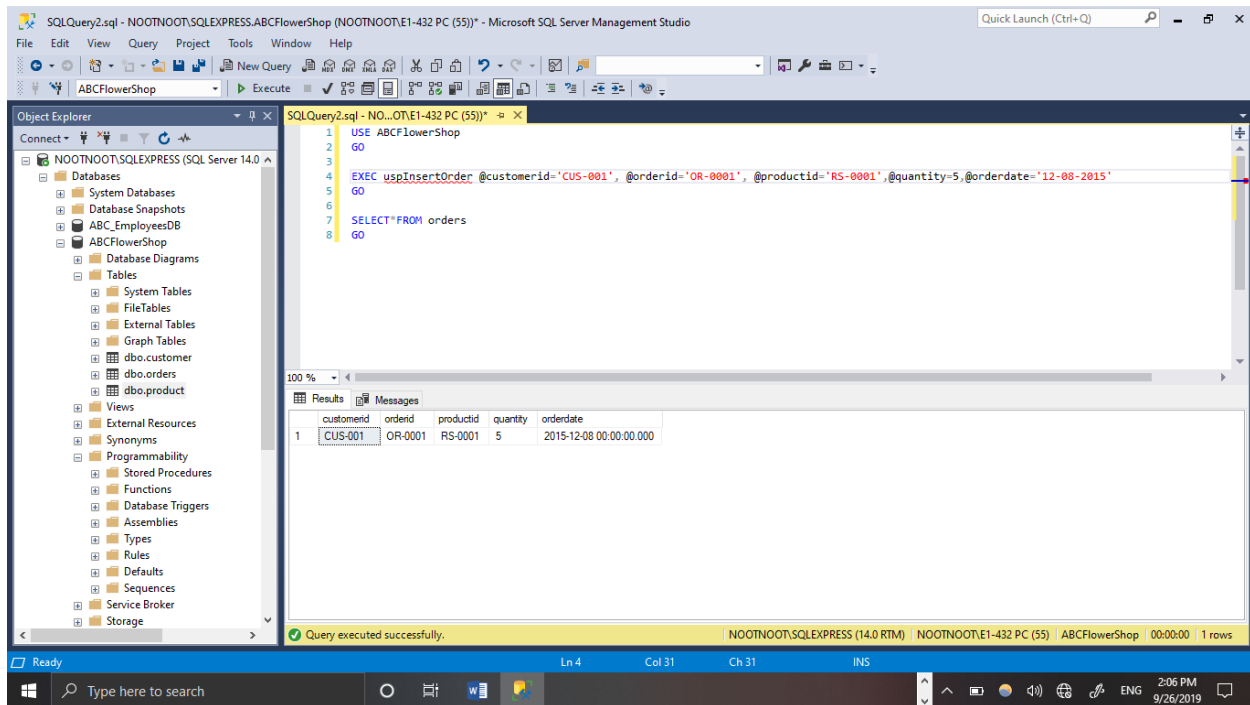
B. Using Stored Procedure



To create a stored procedure, expand the database and navigate to Programmability. Expand the selection and right-click Stored Procedures. Select Stored Procedure option to create a new one.

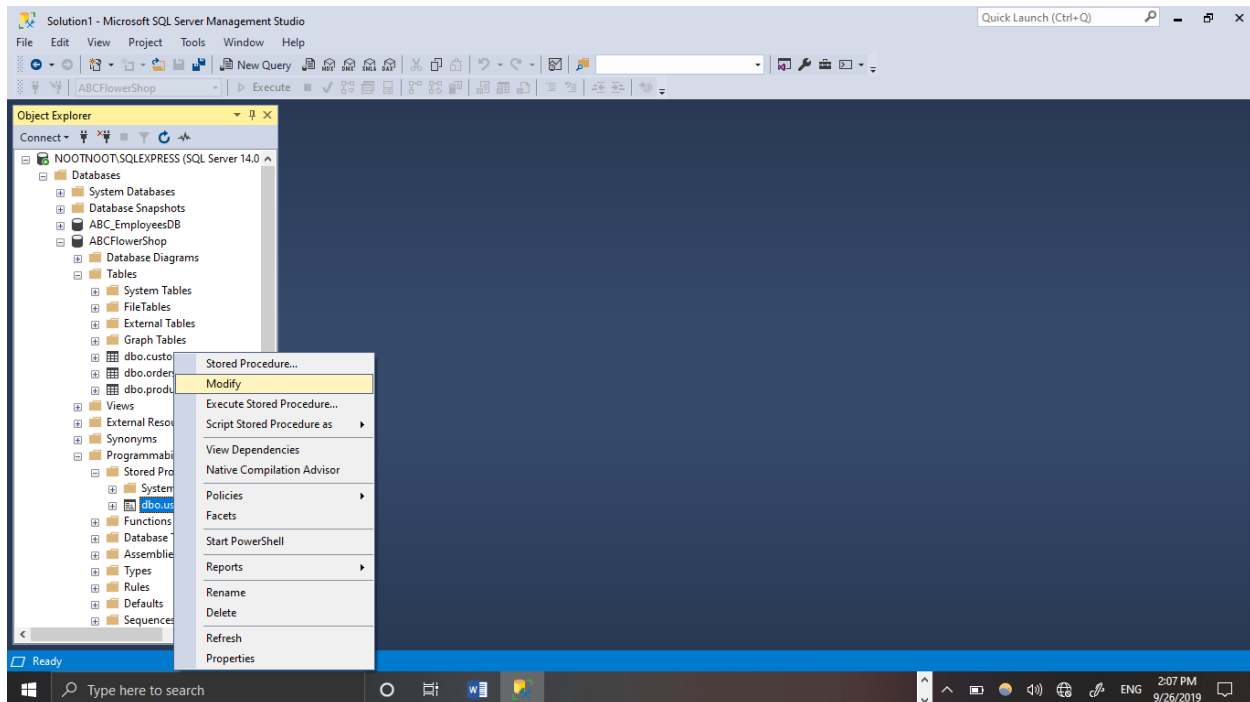


Since stored procedures are essentially SQL statements stored as executable commands in the database, a stored procedure can be used to initiate transaction specific commands when performing an SQL query. In this case, whenever an INSERT statement towards the orders table is executed, the transaction works as well and determine any errors or absence of those. If there is no error, the transaction will be committed by the COMMIT TRAN command. This means that the SQL statement executed will apply permanent effect on the database. If the transaction is rolled back using ROLLBACK TRAN command, the SQL statement executed is pulled back, therefore invalidating its effect, and essentially restoring the state of the database on the instance before the query was executed.

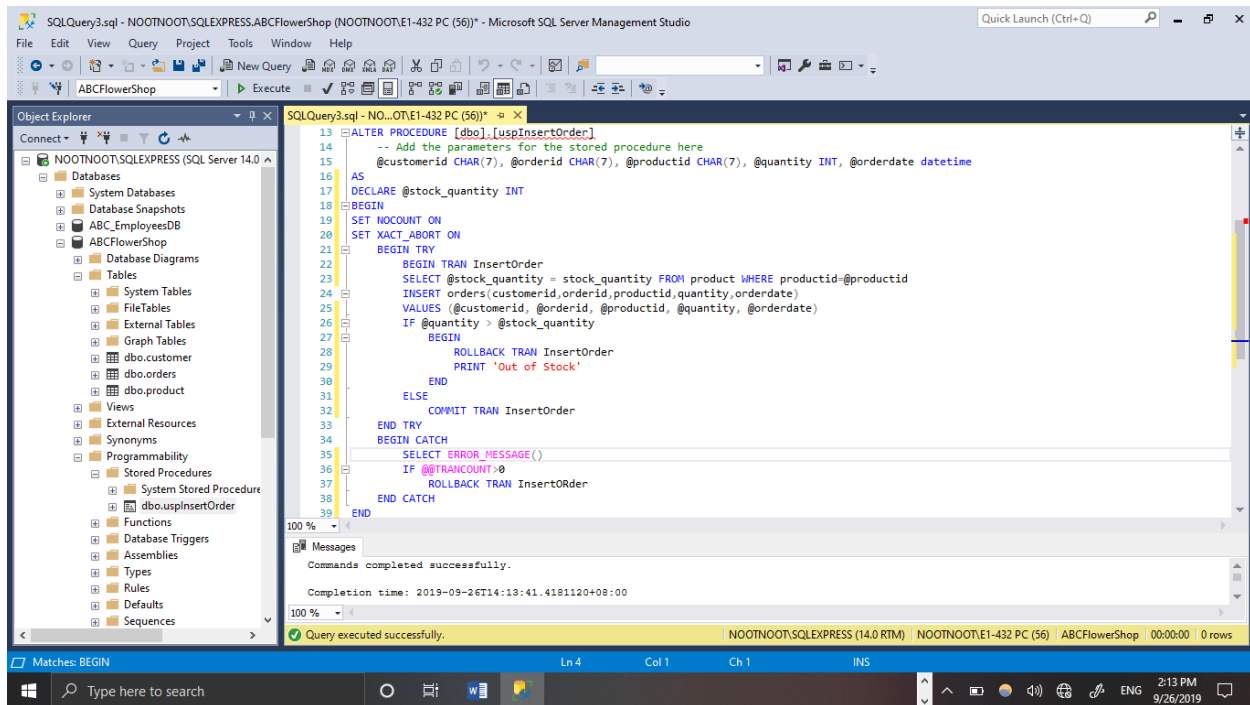


Since there is no problem with the executed INSERT statement, the transaction was committed.

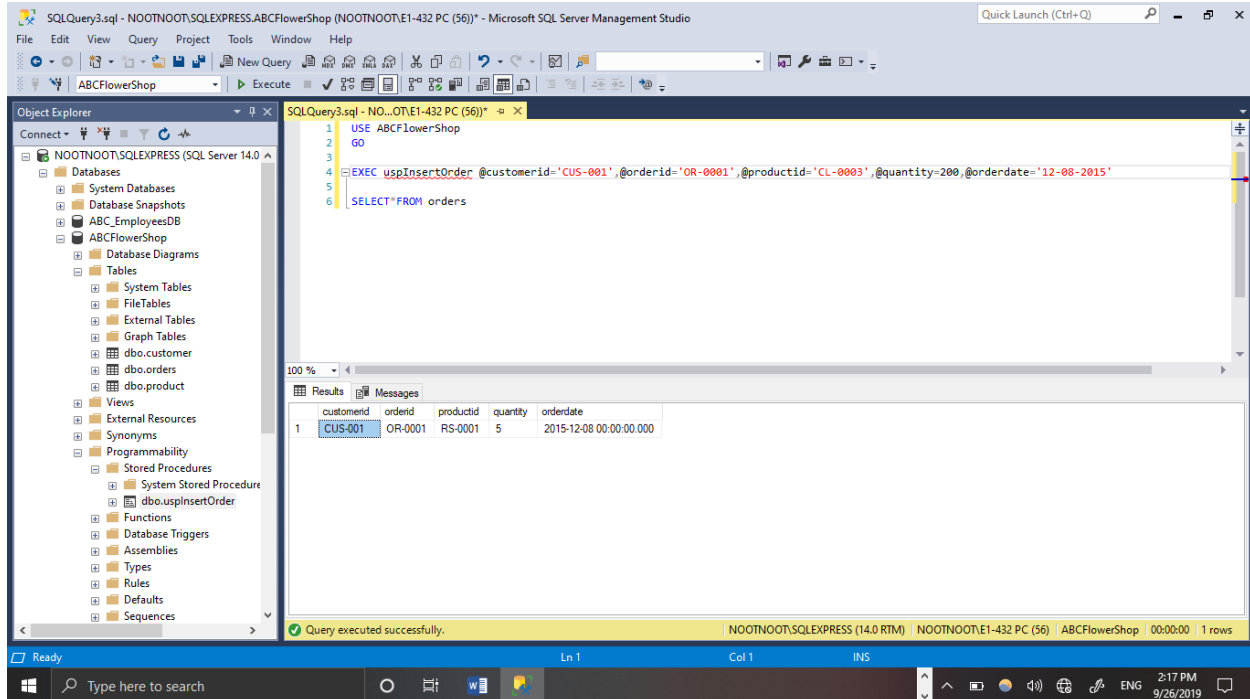
ROLLBACK TRANSACTION



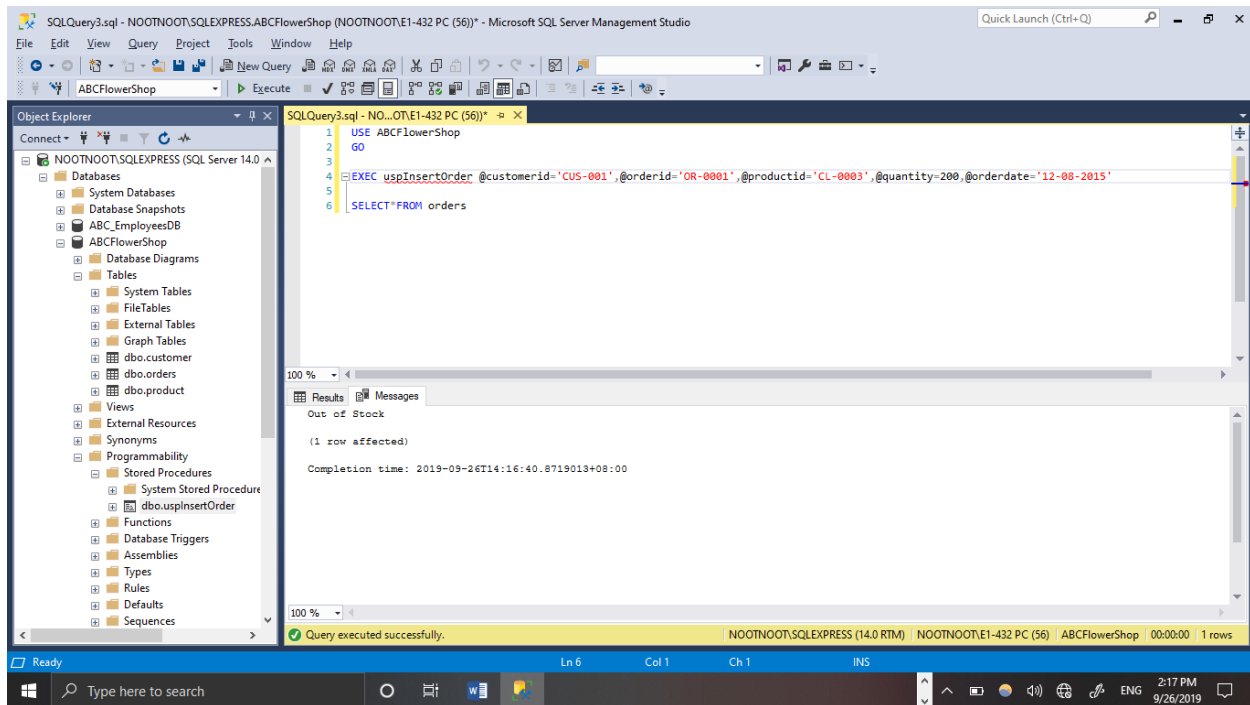
To modify a stored procedure, right-click the name of the stored procedure of interest, under the Programmability folder, further into the Stored Procedures folder.



Aside from mainly executing COMMIT and ROLLBACK commands upon execution of a transaction, other queries can be executed alongside these commands as well. For instance, when a rollback was executed upon catching an error, a PRINT statement can be used to deliver the error message as an indicator.



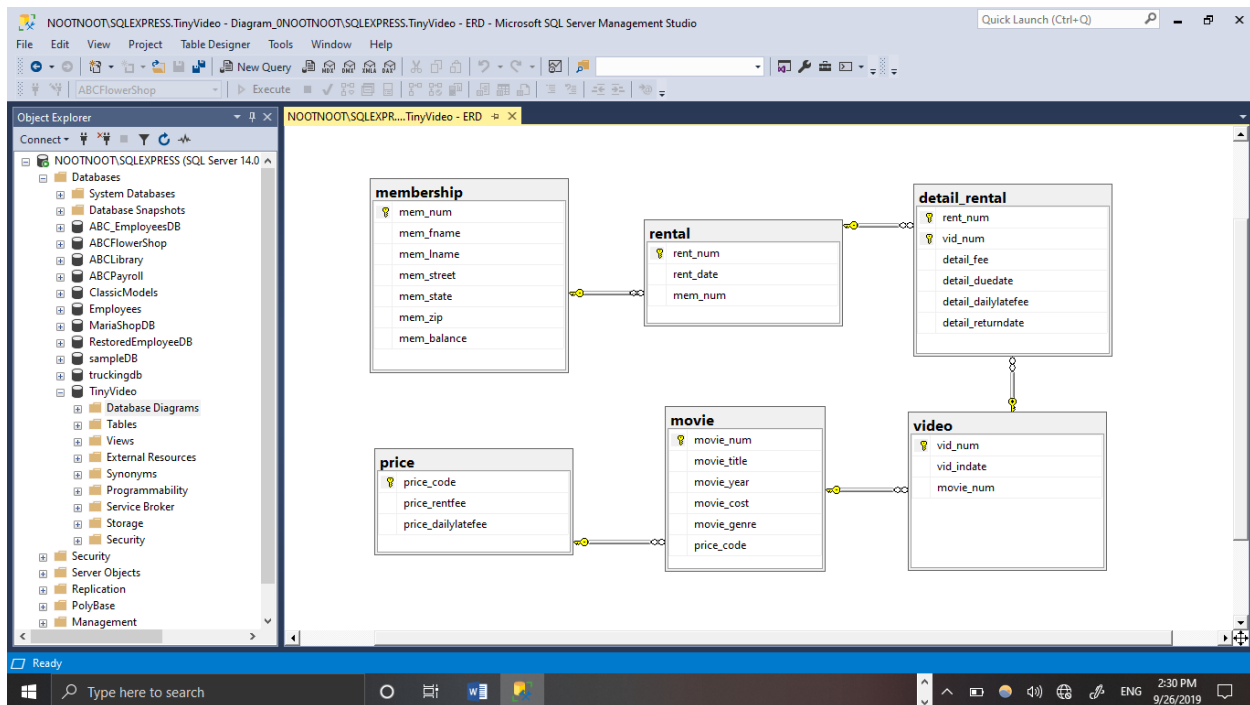
In this case, the error was catch and no changes in the database was made. This was due to the rollback command executed after the transaction.



This can be further confirmed by the error message specified using the PRINT statement.

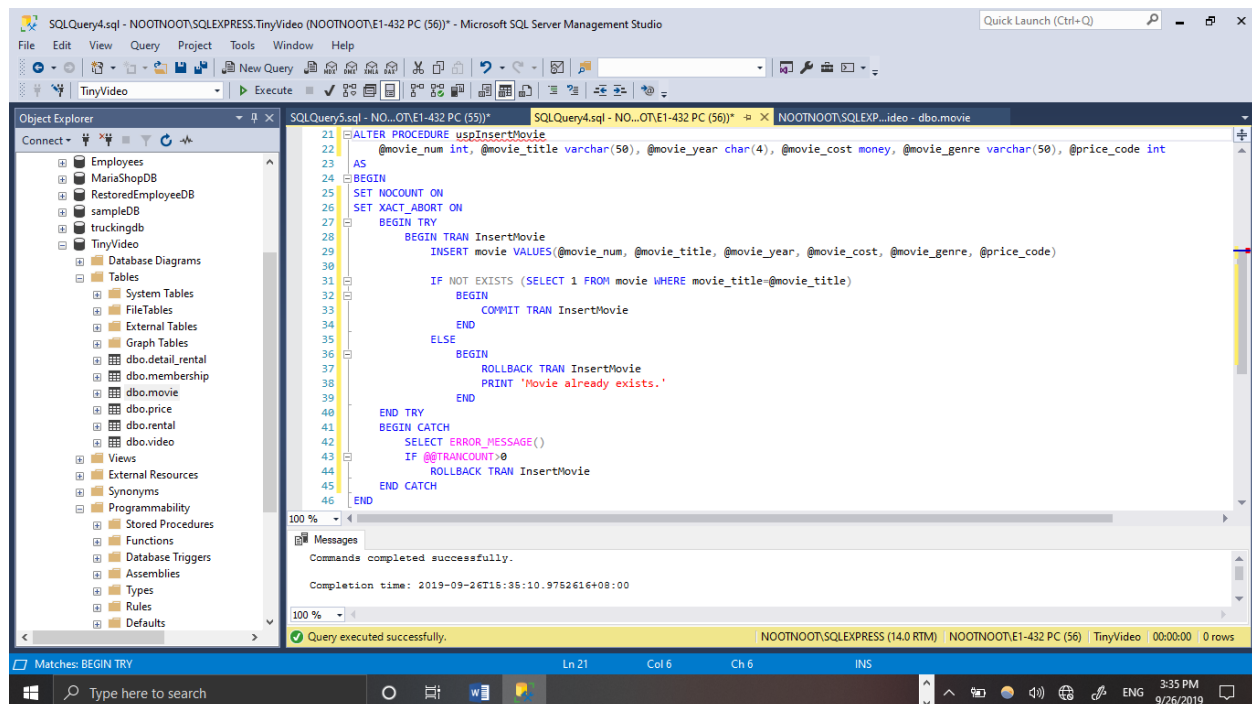
Supplementary Activity

Creation TinyVideo database



The TinyVideo database is composed of the membership, rental, detail rental, price, movie, and video tables.

Transaction to insert a new movie



The screenshot displays the Microsoft SQL Server Management Studio interface. The Object Explorer on the left shows the database structure for 'TinyVideo', including tables like 'dbo.detail_rental', 'dbo.membership', 'dbo.movie', 'dbo.price', 'dbo.rental', and 'dbo.video'. The main query window shows the execution of an ALTER PROCEDURE statement for 'uspInsertMovie'. The procedure logic includes a BEGIN TRY block with an INSERT statement, a check for existing movie titles, a COMMIT, and a ROLLBACK if an error occurs. A PRINT statement displays 'Movie already exists.' if the transaction count is greater than 0. The Messages pane at the bottom shows 'Commands completed successfully.' and the completion time.

```
21 ALTER PROCEDURE uspInsertMovie
22     @movie_num int, @movie_title varchar(50), @movie_year char(4), @movie_cost money, @movie_genre varchar(50), @price_code int
23 AS
24 BEGIN
25     SET NOCOUNT ON
26     SET XACT_ABORT ON
27     BEGIN TRY
28         BEGIN TRAN InsertMovie
29         INSERT movie VALUES(@movie_num, @movie_title, @movie_year, @movie_cost, @movie_genre, @price_code)
30
31         IF NOT EXISTS (SELECT 1 FROM movie WHERE movie_title=@movie_title)
32             BEGIN
33                 COMMIT TRAN InsertMovie
34             END
35         ELSE
36             BEGIN
37                 ROLLBACK TRAN InsertMovie
38                 PRINT 'Movie already exists.'
39             END
40     END TRY
41     BEGIN CATCH
42         SELECT ERROR_MESSAGE()
43         IF @@TRANCOUNT > 0
44             ROLLBACK TRAN InsertMovie
45     END CATCH
46 END
```

Messages
Commands completed successfully.
Completion time: 2019-09-26T15:35:10.9752616+08:00

Query executed successfully.

This transaction requires to display an error message when an error was caught upon insertion of a new movie entry in the database. The conflicting logic that the error needs to catch is the duplication among movie entries upon the execution of the INSERT statement in the movies table. Therefore, the condition checks whether the inserted movie title already exists among the entries in the database. If the movie title already exists, therefore the transaction would not commit and the database will roll back on the instance before the command was executed.

SQLQuery5.sql - NOOTNOOT\SQLEXPRESS.TinyVideo (NOOTNOOT\E1-432 PC (55)) - Microsoft SQL Server Management Studio

```

1  USE TinyVideo
2  GO
3
4  EXEC uspInsertMovie @movie_num=10, @movie_title='The Rise of Skywalker', @movie_year='2019', @movie_cost=1000, @movie_genre='Sci-Fi', @price_code=3
5  GO
6
7  SELECT * FROM movie
8  GO

```

movie_num	movie_title	movie_year	movie_cost	movie_genre	price_code
1	The Phantom Menace	1999	1000.00	Sci-Fi	3
2	Attack of the Clones	2002	1000.00	Sci-Fi	3
3	Revenge of the Sith	2005	1000.00	Sci-Fi	3
4	A New Hope	1977	1000.00	Sci-Fi	3
5	The Empire Strikes Back	1980	1000.00	Sci-Fi	3
6	Return of the Jedi	1983	1000.00	Sci-Fi	3
7	The Force Awakens	2015	1000.00	Sci-Fi	3
8	The Last Jedi	2017	1000.00	Sci-Fi	3
9	The Rise of Skywalker	2019	1000.00	Sci-Fi	3

Query executed successfully. NOOTNOOT\SQLEXPRESS (14.0 RTM) NOOTNOOT\E1-432 PC (55) TinyVideo 00:00:00 9 rows

For instance, the list of existing movie entries are presented.

SQLQuery5.sql - NOOTNOOT\SQLEXPRESS.TinyVideo (NOOTNOOT\E1-432 PC (55)) - Microsoft SQL Server Management Studio

```

1  USE TinyVideo
2  GO
3
4  EXEC uspInsertMovie @movie_num=10, @movie_title='The Rise of Skywalker', @movie_year='2019', @movie_cost=1000, @movie_genre='Sci-Fi', @price_code=3
5  GO
6
7  SELECT * FROM movie
8  GO

```

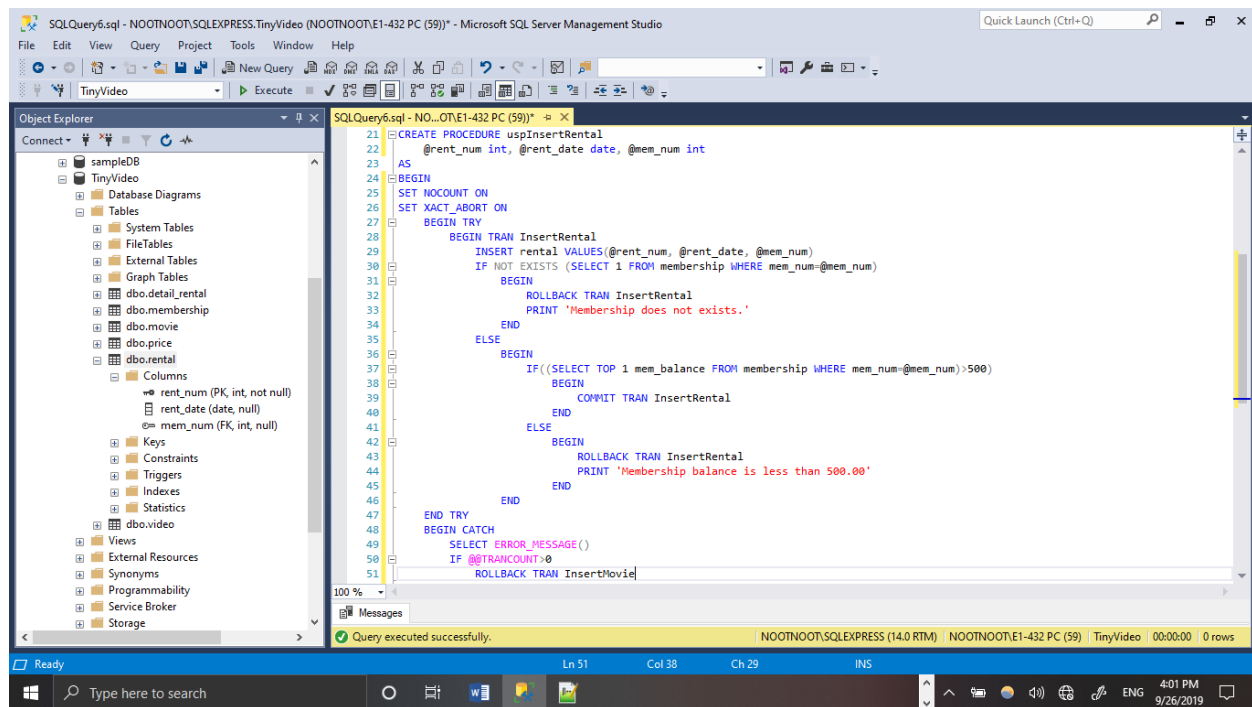
Movie already exists.
(9 rows affected)

Completion time: 2019-09-26T15:36:46.2481405+08:00

Query executed successfully. NOOTNOOT\SQLEXPRESS (14.0 RTM) NOOTNOOT\E1-432 PC (55) TinyVideo 00:00:00 9 rows

Since the movie, “The Rise of Skywalker” is already present, despite the difference in the movie number (primary key), the movie entry was still not added since the movie already exists under different row entry.

Transaction to insert a rental



The condition of these procedure determines whether the transaction will commit or rollback on a previous instance. The specified condition checks the INSERT statements for the rental table. The condition determines whether a membership number exists, and therefore should be allowed for a rental. If not, non-members will not be granted the ability to rent a movie. Furthermore, identified members must have a balance of 500 and above to be eligible. To achieve this condition, nested IF statements were employed. The first pass through of IF statements checks whether the membership number exists on the database. If not, the transaction would automatically not proceed. If the condition was satisfied however, the balance of the specific membership number is then checked.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure for 'sampleDB' and 'TinyVideo'. The 'membership' table is highlighted. The query window on the right contains the following SQL code:

```

1 USE TinyVideo
2 GO
3
4 SELECT * FROM membership

```

The Results pane at the bottom shows the output of the query, displaying two rows of data from the 'membership' table:

mem_num	mem_fname	mem_lname	mem_street	mem_state	mem_zip	mem_balance
1	George	Lucas	NULL	New York	NULL	1000000.00
2	Pau	Cabral		Markina	NULL	333.00

The status bar at the bottom indicates that the query was executed successfully, returning 2 rows.

For instance, two existing members are present in the database. One meets the criteria, and the other does not.

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure for 'sampleDB' and 'TinyVideo'. The 'rental' table is highlighted. The query window on the right contains the following SQL code:

```

1 USE TinyVideo
2 GO
3
4 EXEC uspInsertRental @rent_num=1, @rent_date='2019-09-26', @mem_num=1
5 GO
6
7 SELECT * FROM rental
8 GO

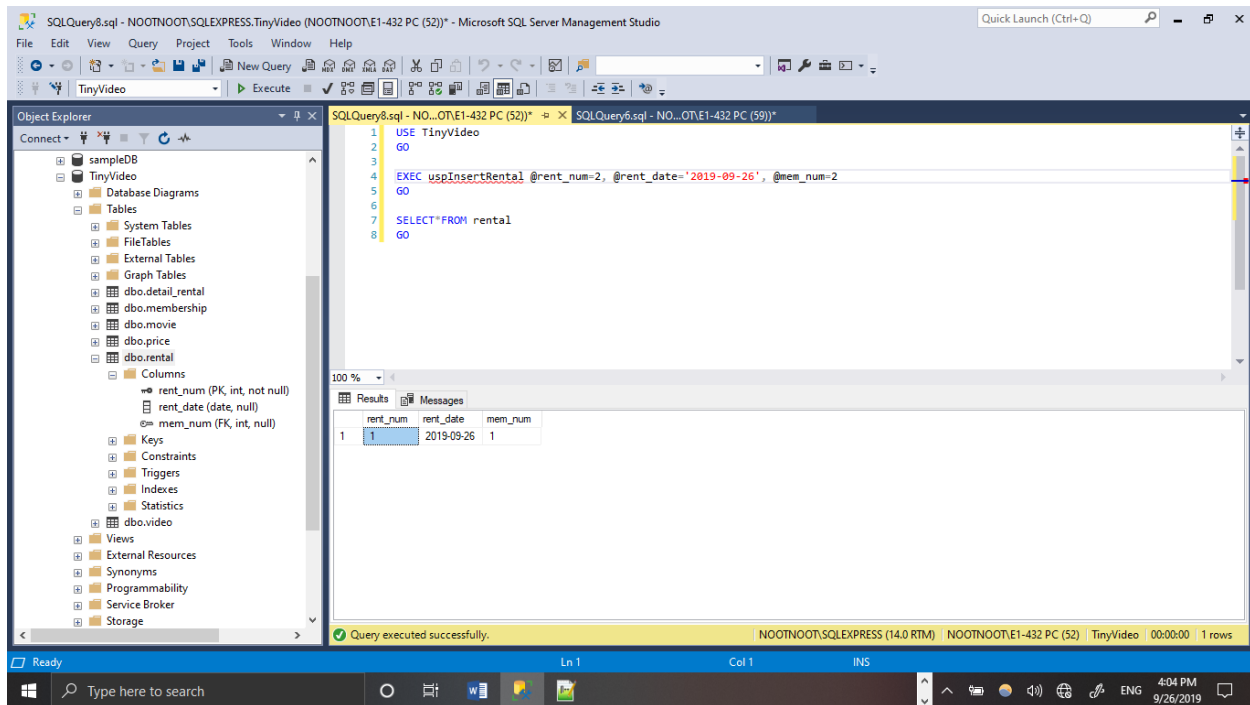
```

The Results pane at the bottom shows the output of the query, displaying one row of data from the 'rental' table:

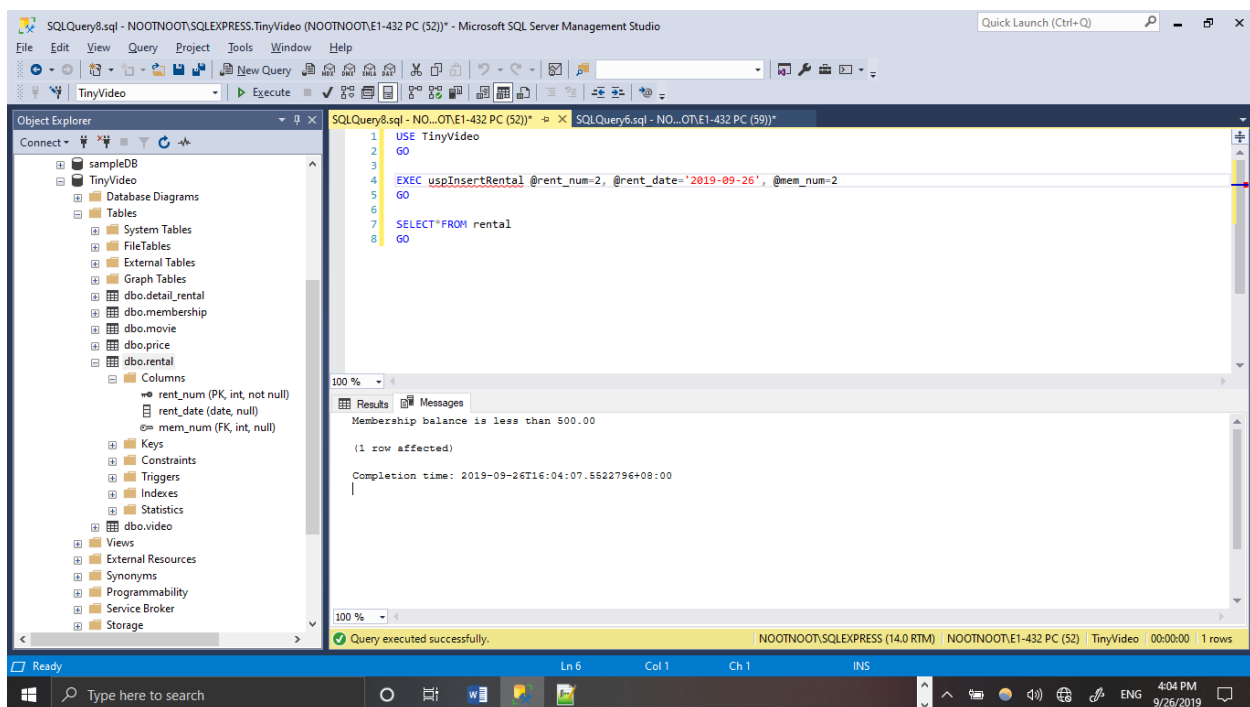
rent_num	rent_date	mem_num
1	2019-09-26	1

The status bar at the bottom indicates that the query was executed successfully, returning 1 row.

Since member number 1 has a load balance of 1000, which is definitely greater than 500, the member was allowed to rent a movie.

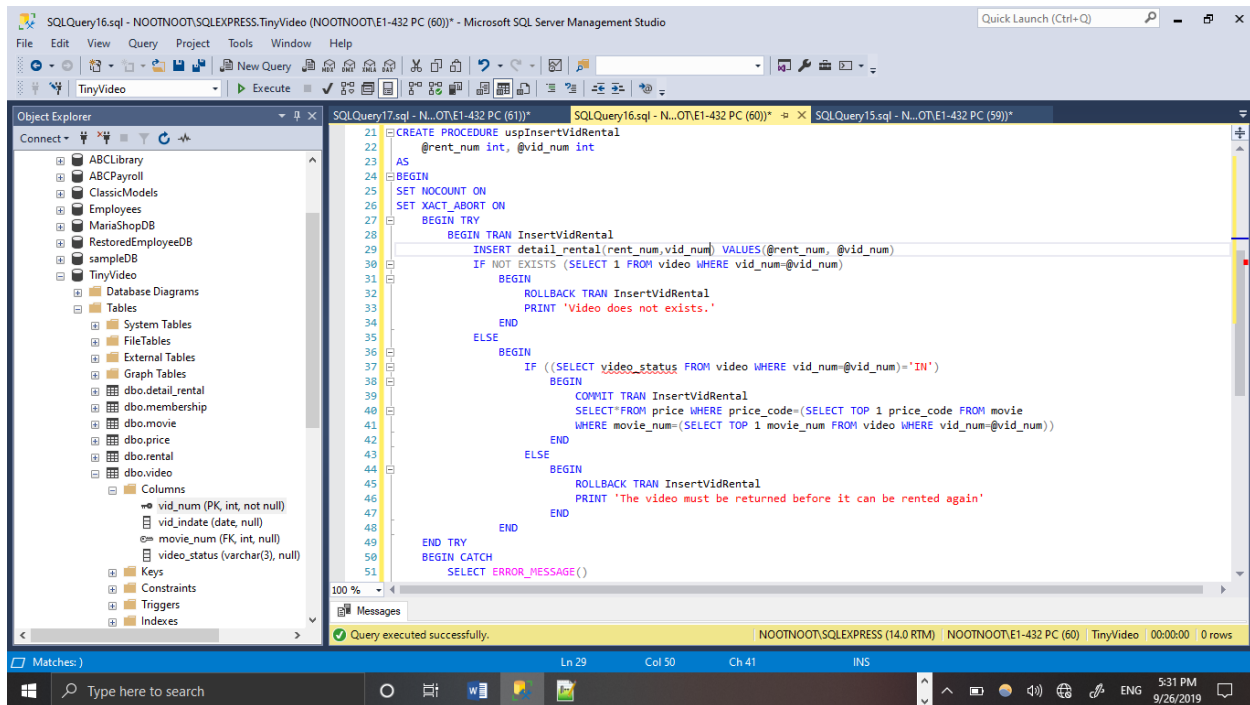


On the other hand, the other member which has less than 500 in his load balance was denied of renting transaction.



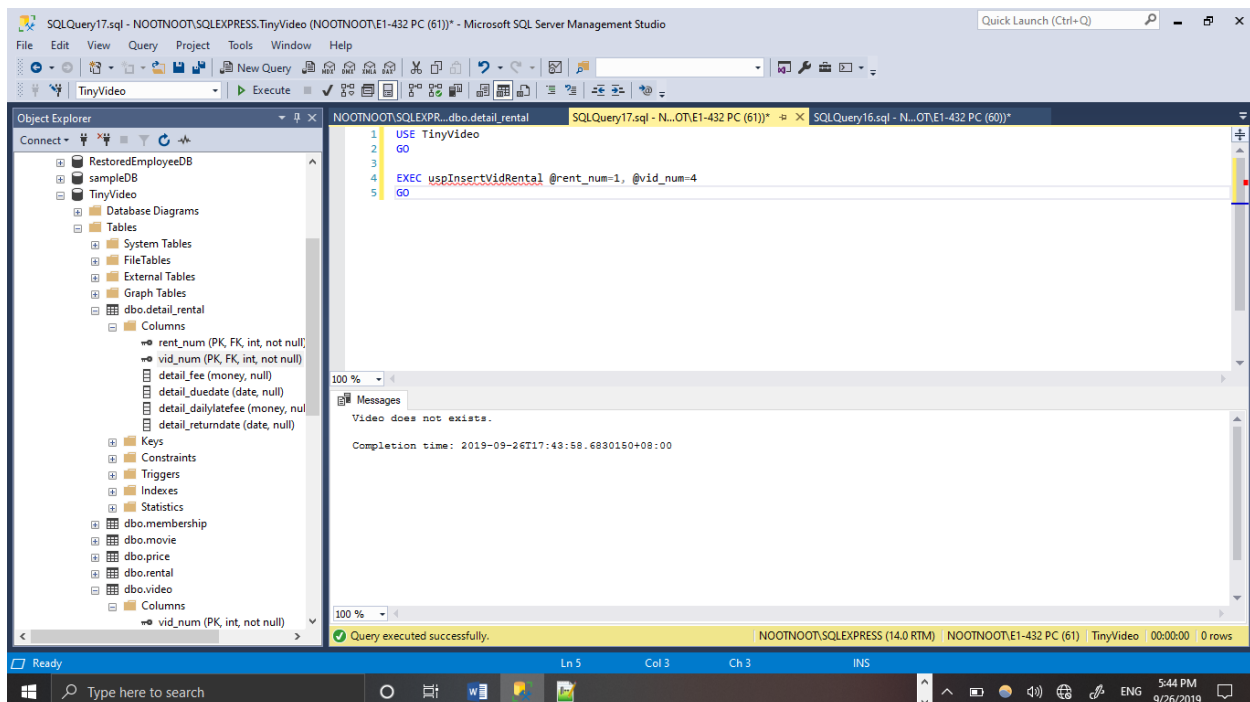
This can be confirmed further through the error message specified by the PRINT statement.

Transaction to insert a video



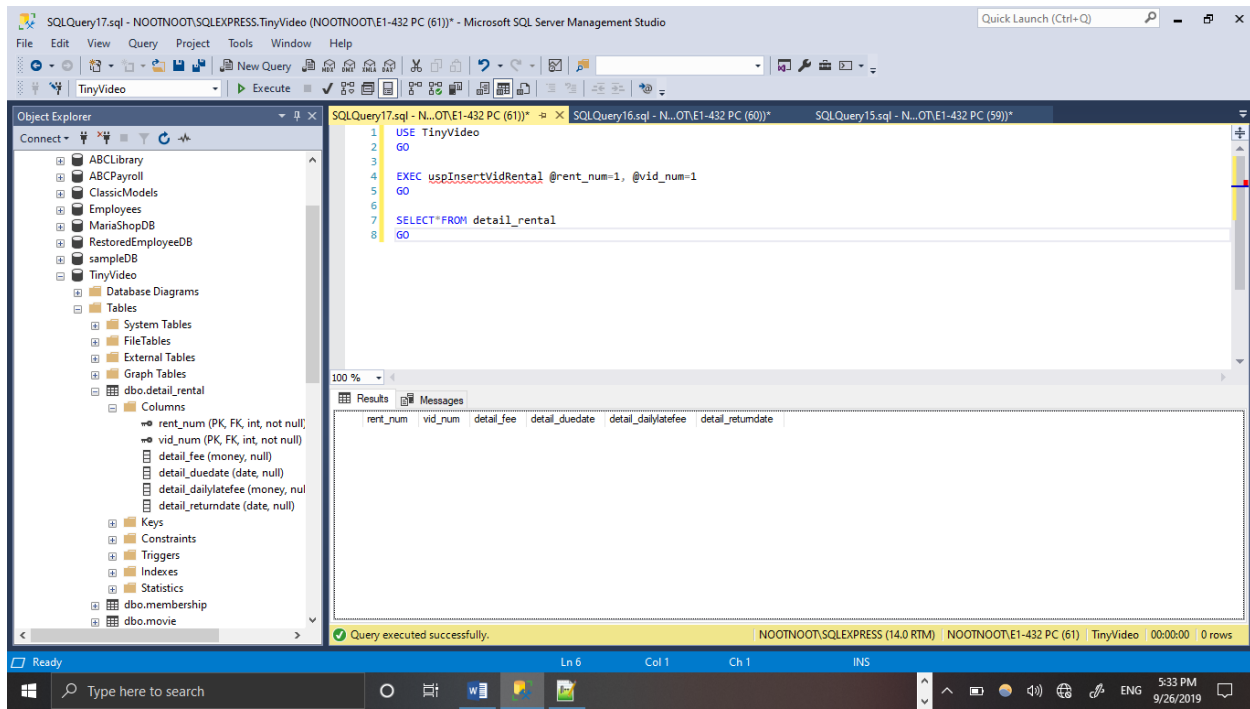
This transaction checks if a video rental would be eligible based on the existence of the video and/or its availability. If the video exists in the database, the procedure would further determine the availability of the video based on its status. If the video does not exist, no transaction will be made. If it does, and the status of the video is eligible for rent, then the transaction will pass through, otherwise, an error message will be displayed as well.

Video does not exist

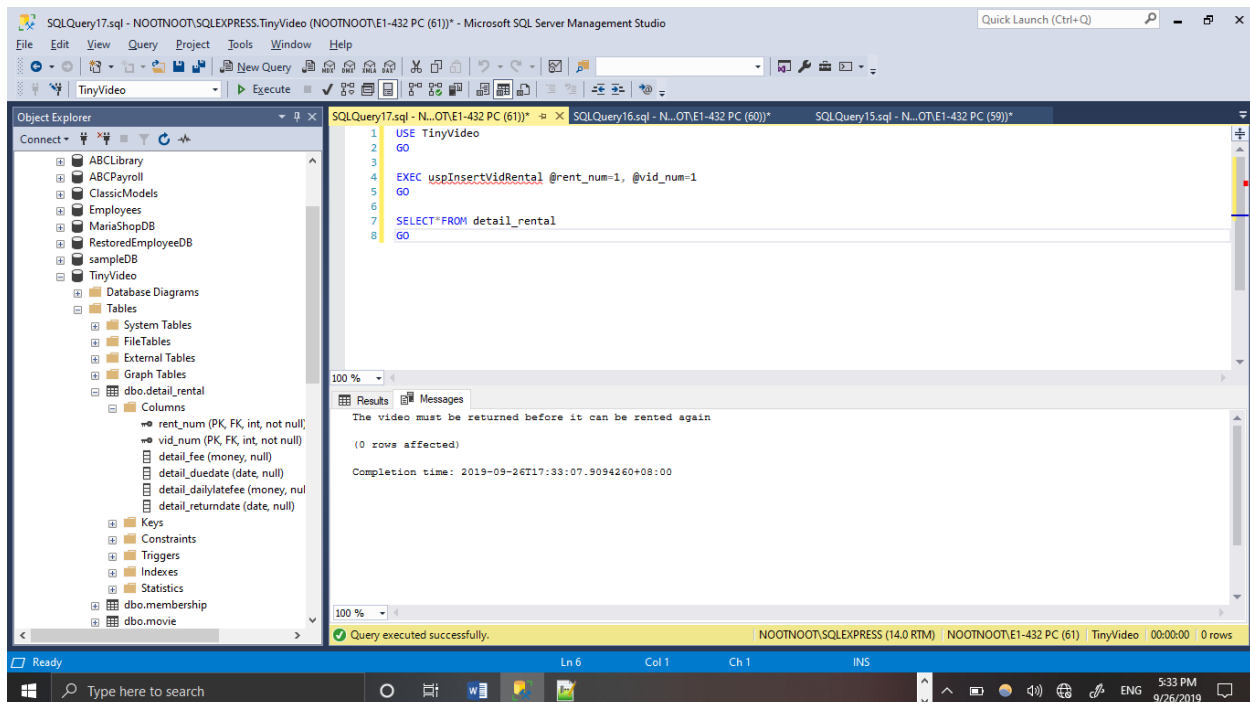


For instances of inserts where video entry does not exist, an error message indicating the absence of the video in the database is displayed.

Video Exists but not 'IN'

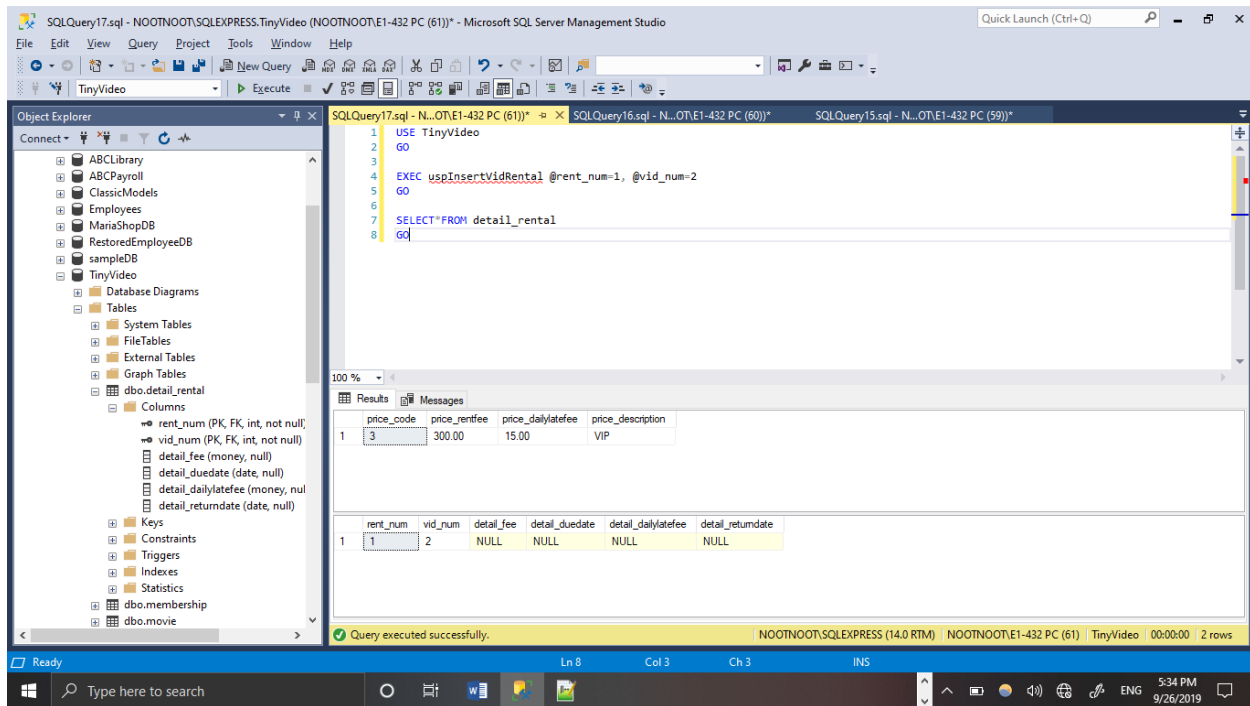


When a video is found in the database but not available for rent, no transaction will be made either.



On the messages tab, the displayed error explains that the no available copy of the video is present at the time of request.

Video exists and 'IN'



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure, including the 'TinyVideo' database and its tables. The SQL Query window in the center shows the following code:

```
1 USE TinyVideo
2 GO
3
4 EXEC uspInsertVidRental @rent_num=1, @vid_num=2
5 GO
6
7 SELECT * FROM detail_rental
8 GO
```

The Messages tab at the bottom shows the execution results. The first table displays price rental properties:

	price_code	price_rentfee	price_dailylatefee	price_description
1	3	300.00	15.00	VIP

The second table displays rental details:

	rent_num	vid_num	detail_fee	detail_duedate	detail_dailylatefee	detail_returndate
1	1	2	NULL	NULL	NULL	NULL

The status bar at the bottom indicates that the query was executed successfully.

If the video exists and is available for rent, the rental details will be filled with an entry as such. The price information of the video is presented to indicate the price rental properties of the video.

Summary

This activity discussed how transactions operate and how these can be used to control the effect of executed SQL queries. In full effect, the activity discussed how a transaction operate when committed or rolled back. The effects of these transaction commands are discussed as well as their effects on the present instance of the database.

Conclusion

Transaction refers to the data modifications made in the database. These transactions can be classified based on their execution and effect on the instance of the database. A transaction for instance may be implicitly or explicitly declared. All database statements that make changes to the database are essentially transactions. Explicitly defined transactions however, are statements with indication of a start of the transaction, and its end, whether ending among a commit command or a rollback command. When a transaction is committed, the changes that the transaction made to the database will be applied in permanent effect in the database. However, if a rollback is called, the executed transaction will be disregarded and the database will be rolled back to the instance before the transaction was made.

Recommendations

Explicit declaration of transactions, along with its implementation with stored procedures allow for better monitoring and code optimization, as well as a huge step for error handling. Catching an error through explicitly defined transactions enables features such as transaction counts and the very valuable commit and rollback commands. These commands should be taken note of all the time as they essentially are responsible for the state of the database after execution of commands.

What I Have Learned

The activity discussed the use of transactions and how they affect the database. In this activity, I have learned how to explicitly declare and execute a transaction, and implement its functionality with the use of stored procedures. The effect of commit and rollback commands are emphasized as well, and viewed in the overall instance of the database after the full effect of transaction.