Module 18: Implementing Transactions

Contents:

Module Overview

As you continue to move past SELECT statements and into data modification operations with T-SQL, you should consider how to structure batches containing multiple modification statements, and those that might encounter errors. In this module, you will learn how to define transactions to control the behavior of batches of T-SQL statements submitted to Microsoft® SQL Server®. You will also learn how to determine whether a runtime error has occurred after work has begun, and whether the work needs to be undone.

Objectives

After completing this module, you will be able to:

- Describe transactions and the differences between batches and transactions.
- Describe batches and how they are handled by SQL Server.
- Create and manage transactions with transaction control language (TCL) statements.
- Use SET XACT_ABORT to define SQL Server's handling of transactions outside TRY/CATCH blocks.

Lesson 1: Transactions and the Database Engine

In this lesson, you will compare simple batches of T-SQL statements to transactions, which allow you to control the behavior of code submitted to SQL Server. You will decide whether special action is needed to respond to a runtime error after work has begun and whether the work needs to be undone.

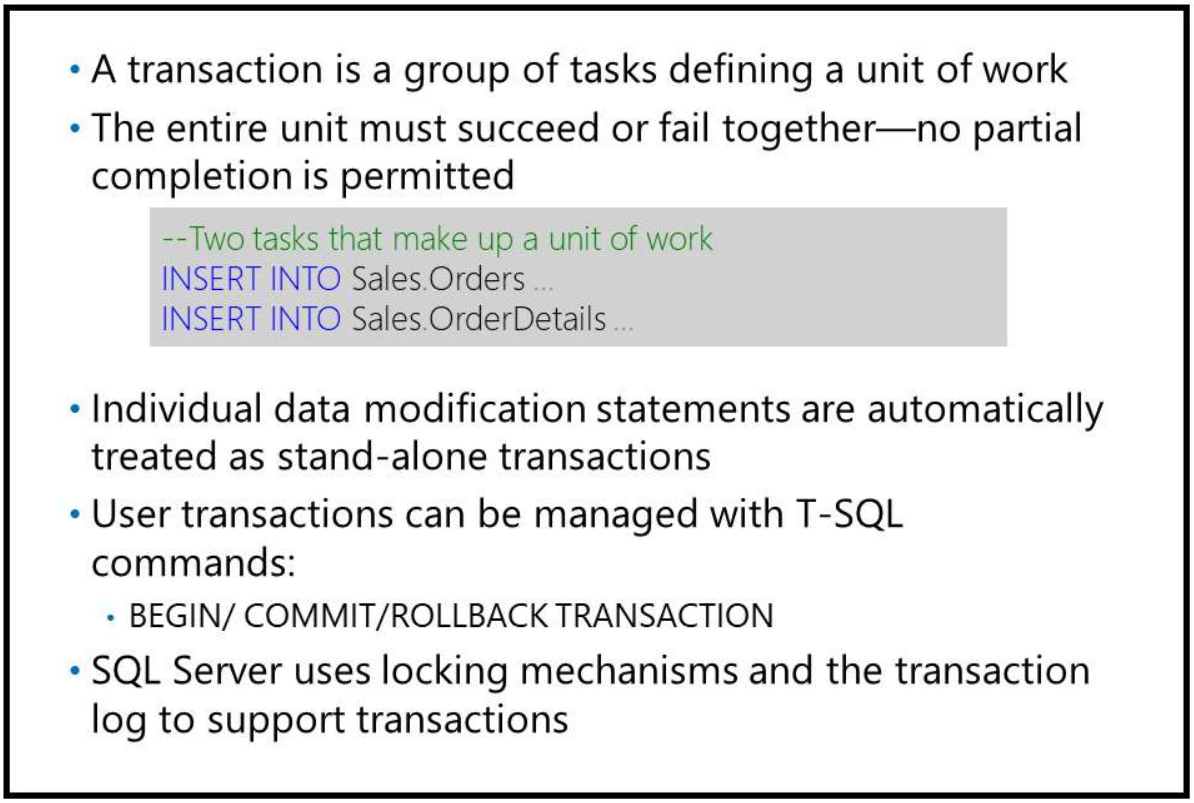
Lesson Objectives

After completing this lesson, you will be able to:

- Describe a SQL Server database transaction.
- Describe the difference between a batch and a transaction.

- Describe how transactions extend batches.

Defining Transactions



Earlier in this course, you learned that a batch was a collection of T-SQL statements sent to SQL Server as a unit for parsing, optimization, and execution. A transaction extends a batch from a unit submitted to the database engine to a unit of work performed by the database engine. A transaction is a sequence of T-SQL statements performed in an all-or-nothing fashion by SQL Server.

Transactions are commonly created in two ways:

- **Autocommit transactions**. Individual data modification statements (for example, INSERT, UPDATE, and DELETE) submitted separately from other commands are automatically wrapped in a transaction by SQL Server. These single-statement transactions are automatically committed when the statement succeeds, or are automatically rolled back when the statement encounters a runtime error.
- **Explicit transactions**. User-initiated transactions are created through the use of TCL commands that begin, commit, or roll back work, based on user-issued code. TCL is a subset of T-SQL.

The primary characteristic of a transaction is that all activity within a transaction's boundaries must either succeed or all fail—no partial completion is permitted. User transactions are typically defined to encapsulate operations that must logically occur together, such as entries into related tables as part of a single business operation.

For example, the following batch inserts data into two tables using two INSERT statements that are part of a single order-processing operation:

```
INSERT INTO dbo.SimpleOrders(custid, empid, orderdate)
        VALUES (68,9,'2006-07-12');
INSERT INTO dbo.SimpleOrderDetails(orderid,productid,unitprice,qty)
        VALUES (1, 2,15.20,20);
GO
```

Business rules might dictate that an order is complete only if the data was successfully inserted into both tables. As you will see in the next lesson, a runtime error in this batch might result in data being inserted into one table but not the other. Enclosing both INSERT statements in a user-defined transaction provides the ability to undo the data insertion in one table if the INSERT statement in the other table fails. A simple batch does not provide this capability.

SQL Server manages resources on behalf of transactions while they are active. These resources might include locks and entries in the transaction log to allow SQL Server to undo changes made by the transaction, should a rollback be required.

For more information, see Microsoft Docs:

Transaction Statements (Transact-SQL)

http://aka.ms/H9jd4y

The Need for Transactions: Issues with Batches

- To work around this situation, you will need to direct SQL Server to treat the batch as a transaction; you will learn more about creating transactions in the next topic

```
--Batch without transaction management
BEGIN TRY
        INSERT INTO Sales.Orders ... --Insert succeeds
        INSERT INTO Sales.OrderDetails ... --Insert fails
END TRY
BEGIN CATCH
        --Inserted rows still exist in Sales.Orders Table
        SELECT ERROR_NUMBER()
        ...
END CATCH;
```

While batches of T-SQL statements provide a unit of code submitted to the server, they do not include any logic for dealing with partial success when a runtime error occurs, even with the use of structured exception handling's TRY/CATCH blocks.

Code Without Transaction

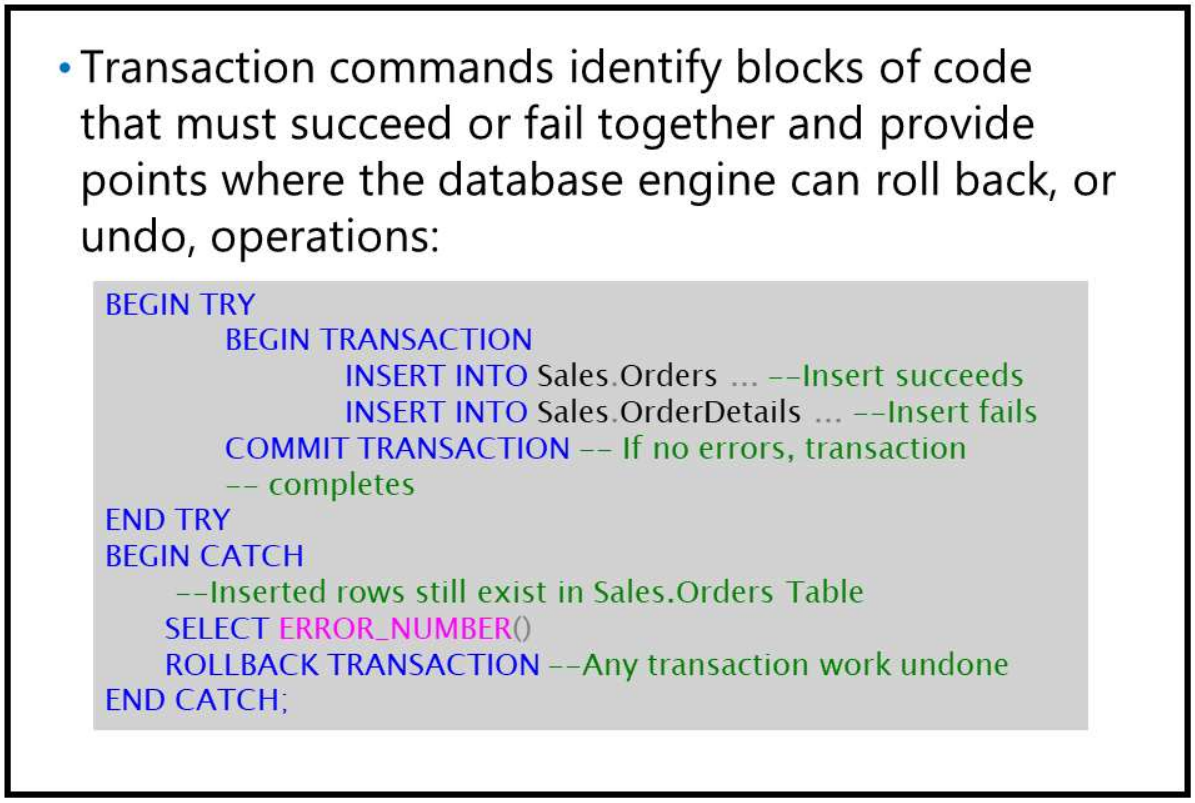
```
BEGIN TRY
        INSERT INTO dbo.SimpleOrders(custid, empid, orderdate)
                VALUES (68,9,'2006-07-12');
        INSERT INTO dbo.SimpleOrders(custid, empid, orderdate)
                VALUES (88,3,'2006-07-15');
        INSERT INTO dbo.SimpleOrderDetails(orderid,productid,unitprice,qty)
                VALUES (1, 2,15.20,20);
        INSERT INTO dbo.SimpleOrderDetails(orderid,productid,unitprice,qty)
                VALUES (999,77,26.20,15);
END TRY
BEGIN CATCH
        SELECT ERROR_NUMBER() AS ErrNum, ERROR_MESSAGE() AS ErrMsg;
END CATCH;
```

If the first INSERT statement succeeds but a subsequent one fails, the new row in the dbo.SimpleOrders table will persist after the end of the batch, even after the execution branches to the CATCH block. This issue applies to any successful statements, if a later statement fails with a runtime error.

> **Note:** Remember that syntax or name-resolution errors cause the entire batch to return an error, preventing any execution. Runtime errors only occur after the batch has been submitted, parsed, planned, and compiled for execution.

To work around this situation, you will need to direct SQL Server to treat the batch as a transaction. You will learn more about creating transactions in the next topic.

Transactions Extend Batches



As you have seen, runtime errors encountered during the execution of simple batches create the possibility of partial success, which is not typically a desired outcome. To address this, you will add code to identify the batch as a transaction by placing the batch between BEGIN TRANSACTION and COMMIT TRANSACTION statements. You will also add error-handling code to roll back the transaction should an error occur. This error-handling code will undo the partial changes made before the error occurred.

Transaction Example

```
BEGIN TRY
 BEGIN TRANSACTION;
        INSERT INTO dbo.SimpleOrders(custid, empid, orderdate)
                VALUES (68,9,'2006-07-15');
        INSERT INTO dbo.SimpleOrderDetails(orderid,productid,unitprice,qty)
                VALUES (99, 2,15.20,20);
        COMMIT TRANSACTION;
END TRY
BEGIN CATCH
 SELECT ERROR_NUMBER() AS ErrNum, ERROR_MESSAGE() AS ErrMsg;
 ROLLBACK TRANSACTION;
END CATCH;
```

Within the TRY block, the INSERT statements are wrapped by BEGIN TRANSACTION and COMMIT TRANSACTION statements. This identifies the INSERT statements as a single unit of work that must succeed or fail together. If no runtime error occurs, the transaction commits, and the result of each INSERT is allowed to persist in the database.

If an error occurs during the execution of the first INSERT statement, the execution branches to the CATCH block, bypassing the second INSERT statement. The ROLLBACK statement in the CATCH block terminates the transaction, releasing its resources.

If an error occurs during the execution of the second INSERT statement, the execution branches to the CATCH block. Because the first INSERT completed successfully and added rows to the dbo.SimpleOrders table, the ROLLBACK statement is used to undo the successful INSERT operation.

> **Note:** You will learn how to use the BEGIN TRANSACTION, COMMIT TRANSACTION, and ROLLBACK TRANSACTION statements in the next lesson.

Demonstration: Transactions and the Database Engine

In this demonstration, you will see how to use transactions.

Demonstration Steps

Use Transactions

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd.**
2. Run **D:\Demofiles\Mod18\Setup.cmd** as an administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. At the command prompt, type **y**, and then press Enter.
5. Press any key to continue.
6. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
7. On the **File** menu, point to **Open**, and then click **Project/Solution**.
8. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod18\Demo** folder.
9. In Solution Explorer, in the Queries folder, open the **11 - Demonstration A.sql** script file.
10. Select the code under the comment **Step 1**, and then click **Execute**.
11. Select the code under the comment **Step 2**, and then click **Execute**.
12. Select the code under the comment **Step 3**, and then click **Execute**. Note the error message.
13. Select the code under the comment **Step 4**, and then click **Execute**.
14. Select the code under the comment **Step 5**, and then click **Execute**.
15. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Categorize Activity

Place each T-SQL keyword into the appropriate category. Indicate your answer by writing the category number to the right of each item.

TCL

- COMMIT TRANSACTION
- END TRANSACTION
- BEGIN TRANSACTION
- ROLLBACK TRANSACTION

Non-TCL

- END TRY
- END CATCH
- BEGIN CATCH
- BEGIN TRY
- INSERT
- RAISERROR

Check answer Show solution Reset

Correct

Lesson 2: Controlling Transactions

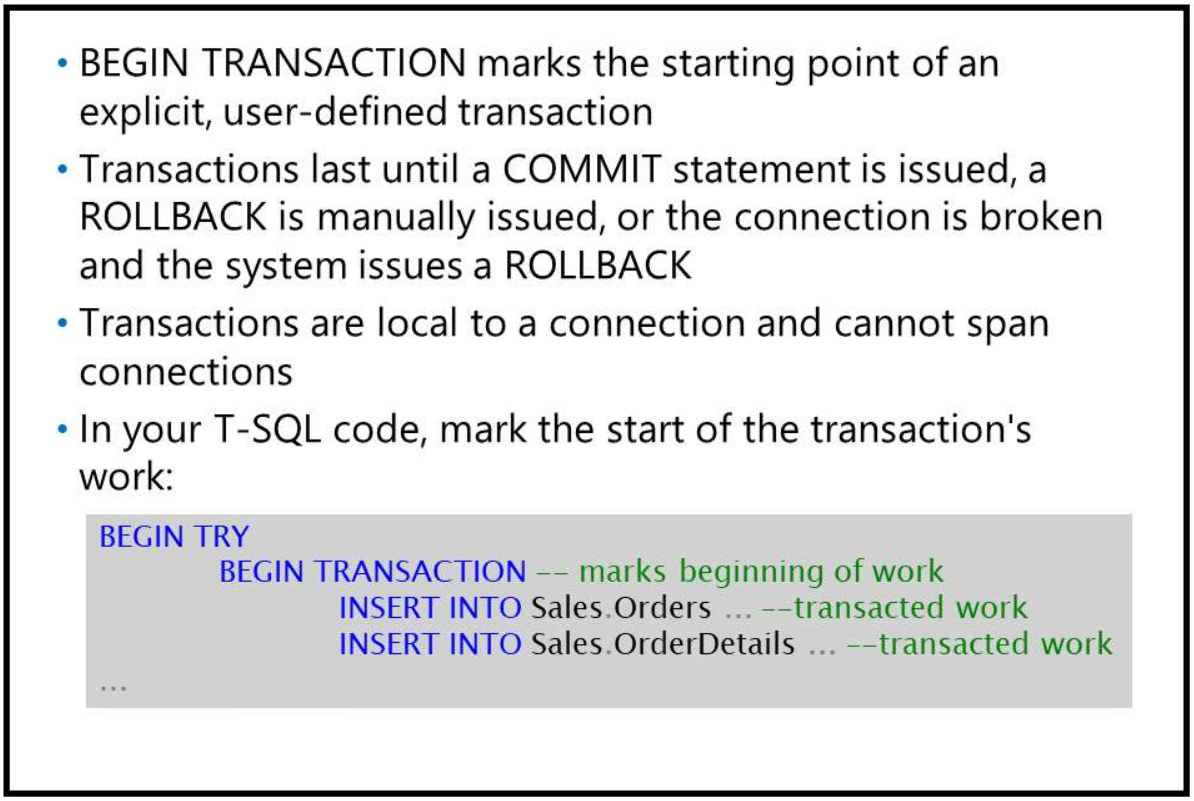
To control how SQL Server treats your data modification statements, you need to use TCL statements. By enclosing batches between BEGIN TRANSACTION and COMMIT or ROLLBACK TRANSACTION statements, you will identify the units of work to be performed together and provide points of recovery in your code.

Lesson Objectives

After completing this lesson, you will be able to:

- Mark the beginning of units of work with BEGIN TRANSACTION.
- Mark successful completion of batches with COMMIT TRANSACTION.
- Undo failed transactions with ROLLBACK TRANSACTION.
- Describe how to use XACT_ABORT to automatically roll back failed T-SQL statements.

BEGIN TRANSACTION



SQL Server will automatically wrap individual data modification statements (for example, INSERT, UPDATE, and DELETE) in their own transactions, which auto-commit on success and auto-rollback on failure. While this behavior is transparent to the user, you have seen the results of this when you have executed a batch of T-SQL statements with partial success. Successful INSERTS have written their values to the target tables, while failed statements have not left values behind.

If you need to identify a group of statements as a transactional unit of work, you cannot rely on this automatic behavior. Instead, you will need to manually specify the boundaries of the unit. To mark the start of a transaction, use the BEGIN TRANSACTION statement, which may also be stated as BEGIN TRAN.

If you are using T-SQL structured exception handling, you will want to begin the transaction inside a TRY block. Within the exception handler, you may decide whether to COMMIT or ROLLBACK the transaction, depending on its outcome.

When you identify your own transactions with BEGIN TRANSACTION, consider the following:

- Once you initiate a transaction, you must properly end it. Use COMMIT TRANSACTION on success or ROLLBACK TRANSACTION on failure.

- While transactions may be nested, inner transactions will be rolled back, even if committed, if the outer transaction rolls back. Therefore, nested transactions are not typically useful in user code.
- Transactions last until a COMMIT TRANSACTION or a ROLLBACK TRANSACTION is issued, or until the originating connection is dropped, at which point SQL Server will roll back the transaction automatically.
- A transaction's scope is the connection in which it was started. Transactions cannot span connections (except by bound sessions, a deprecated feature that is beyond the scope of this course).
- SQL Server may take and hold locks on resources during the lifespan of the transaction. To reduce concurrency issues, consider keeping your transactions as short as possible. For more information on locking in SQL Server, see course 20762C: *Developing Microsoft SQL Server Databases*.

For more information on BEGIN TRANSACTION statements, see Microsoft Docs:

BEGIN TRANSACTION (Transact-SQL)

http://aka.ms/E3u6jb

For more information on nested transactions, see the SQL Server Technical Documentation:

Nesting Transactions

http://go.microsoft.com/fwlink/?LinkID=402857

COMMIT TRANSACTION

- COMMIT ensures all of the transaction's modifications are made a permanent part of the database
- COMMIT frees resources, such as locks, used by the transaction
- In your T-SQL code, if a transaction is successful, commit it

```
BEGIN TRY
        BEGIN TRAN -- marks beginning of work
                INSERT INTO Sales.Orders ...
                INSERT INTO Sales.OrderDetails ...
        COMMIT TRAN -- mark the work as complete
END TRY
```

When the statements in your transaction have completed without error, you need to instruct SQL Server to end the transaction, making the modifications permanent and releasing resources that were held on behalf of the transaction. To do this, use the COMMIT TRANSACTION (or COMMIT TRAN) statement.

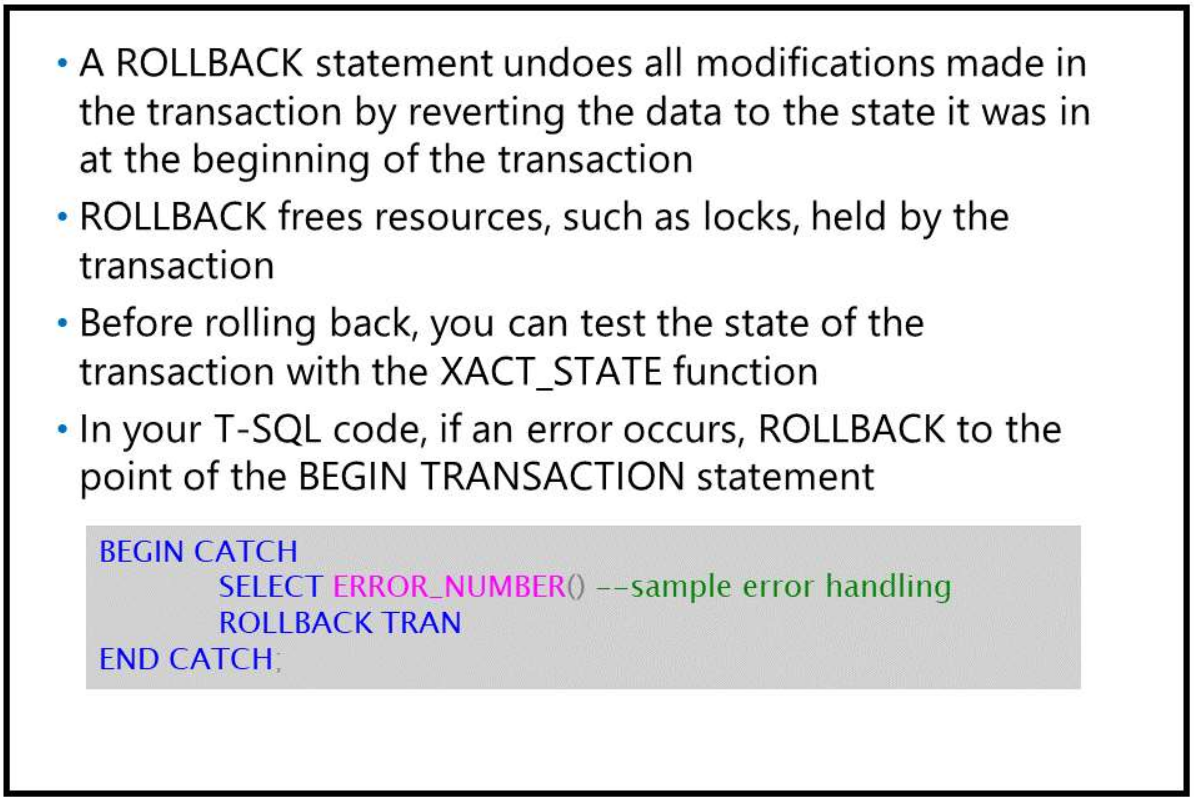
If you are using T-SQL structured exception handling, you will want to COMMIT the transaction inside the TRY block in which you began it.

COMMIT TRANSACTION Example

```
BEGIN TRY
 BEGIN TRANSACTION
        INSERT INTO dbo.SimpleOrders(custid, empid, orderdate)
        VALUES (68,9,'2006-07-12');
        INSERT INTO dbo.SimpleOrderDetails(orderid,productid,unitprice,qty)
        VALUES (1, 2,15.20,20);
 COMMIT TRANSACTION
END TRY
```

Note: The previous example does not contain logic to determine if the transaction should be committed or rolled back. It is relying on the success of the statements to provide the logic to implement error handling.

ROLLBACK TRANSACTION



To end a failed transaction, you will use the ROLLBACK command. ROLLBACK undoes any modifications made to data during the transaction, reverting it to the state it was in when the transaction started. This includes rows

inserted, deleted, or updated, in addition to objects created. ROLLBACK also allows SQL Server to release resources, such as locks, held during the transaction's lifespan.

If you are using T-SQL structured exception handling, you will want to ROLLBACK the transaction inside the CATCH block that follows the TRY block containing the BEGIN and COMMIT statements.

ROLLBACK TRANSACTION Example

```
BEGIN TRY
 BEGIN TRANSACTION;
        INSERT INTO dbo.SimpleOrders(custid, empid, orderdate)
        VALUES (68,9,'2006-07-12');
        INSERT INTO dbo.SimpleOrderDetails(orderid,productid,unitprice,qty)
        VALUES (1, 2,15.20,20);
 COMMIT TRANSACTION;
END TRY
BEGIN CATCH
        SELECT ERROR_NUMBER() AS ErrNum, ERROR_MESSAGE() AS ErrMsg;
        ROLLBACK TRANSACTION;
END CATCH;
```

Before issuing a ROLLBACK command, you might wish to test to see if a transaction is active. You can use the T-SQL XACT_STATE function to determine if there is an active transaction to be rolled back. This can help avoid errors being raised inside the CATCH block.

XACT_STATE returns the following values:

XACT_STATE Results	Description
0	There is no active user transaction.
1	The current request has an active, committable, user transaction.
-1	The current request has an active user transaction, but an error has occurred. The transaction can only be rolled back.

The following example shows the use of XACT_STATE to issue a ROLLBACK statement only if the transaction is active but cannot be committed:

```
BEGIN TRY
 BEGIN TRANSACTION;
        INSERT INTO dbo.SimpleOrders(custid, empid, orderdate)
        VALUES (68,9,'2006-07-12');
        INSERT INTO dbo.SimpleOrderDetails(orderid,productid,unitprice,qty)
        VALUES (1, 2,15.20,20);
 COMMIT TRANSACTION;
END TRY
BEGIN CATCH
```

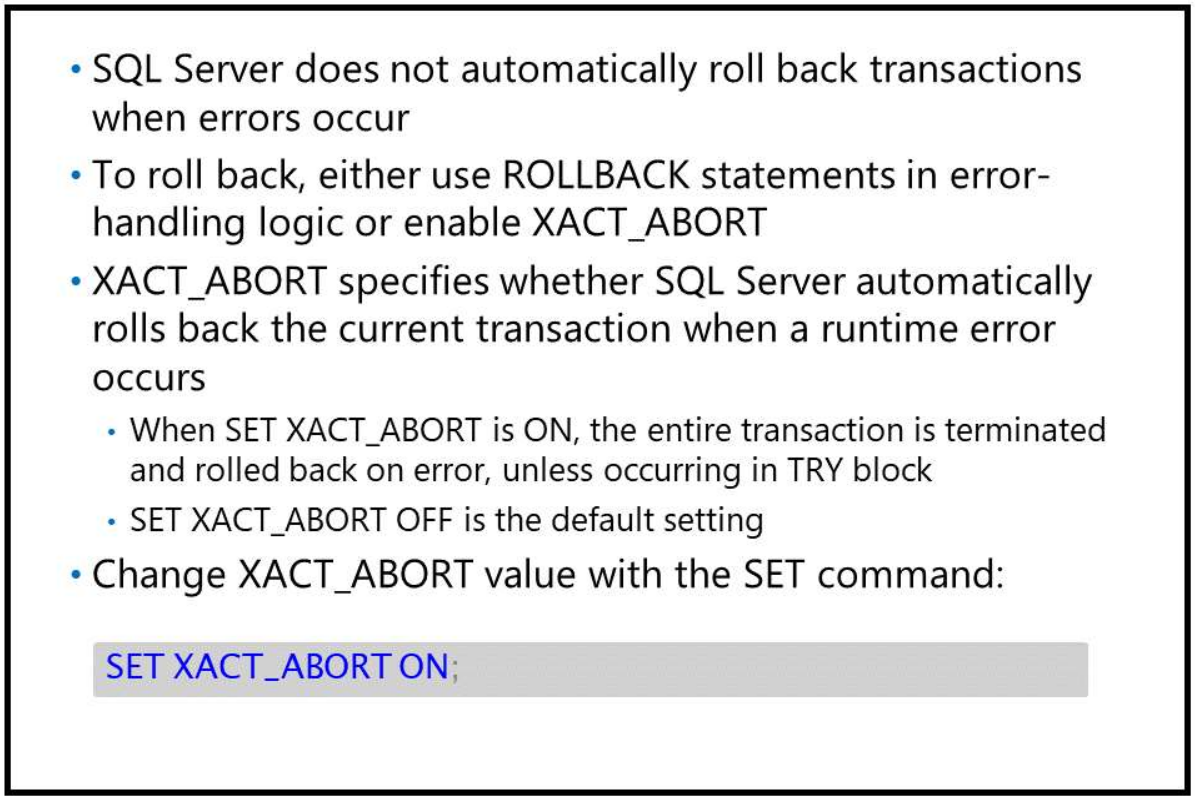
```
        SELECT ERROR_NUMBER() AS ErrNum, ERROR_MESSAGE() AS ErrMsg;
        IF (XACT_STATE()) = -1
        BEGIN
        ROLLBACK TRANSACTION;
        END;
        ELSE .... -- provide for other outcomes of XACT_STATE()
END CATCH;
```


For more information, see *Transaction Statements (Transact-SQL)* in Microsoft Docs:

Transaction Statements (Transact-SQL)

http://aka.ms/h9jd4y

Using XACT_ABORT



As you have seen, SQL Server does not automatically roll back transactions when errors occur. In this module, most of the discussion about controlling transactions has assumed the use of TRY/CATCH blocks to perform the logic and either commit or roll back a transaction. For situations in which you are not using TRY/CATCH blocks, another option exists for automatically rolling back a transaction when an error occurs. The XACT_ABORT setting can be used to specify whether SQL Server rolls back the current transaction when a runtime error occurs during the execution of T-SQL code.

By default, XACT_ABORT is off. Change the XACT_ABORT setting with the SET command:


```
SET XACT_ABORT ON;
```

When SET XACT_ABORT is ON, the entire transaction is terminated and rolled back on error, unless the error occurs in a TRY block. An error in a TRY block leaves the transaction open but not committable, despite the setting of XACT_ABORT.

For more information, see Microsoft Docs:

SET XACT_ABORT (Transact-SQL)

http://aka.ms/Qehdhl

Demonstration: Controlling Transactions

In this demonstration, you will see how to control transactions.

Demonstration Steps

Control Transactions

1. In Solution Explorer, open the **21 - Demonstration B.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4**, and then click **Execute**.
6. Select the code under the comment **Step 5**, and then click **Execute**.
7. Select the code under the comment **Step 6**, and then click **Execute**. Note the error message.
8. Select the code under the comment **Step 7**, and then click **Execute**.
9. Select the code under the comment **Step 8**, and then click **Execute**.
10. Close SQL Server Management Studio without saving any files.

Check Your Knowledge

Discovery

You have executed the following batch of T-SQL statements:

```
BEGIN TRY
  BEGIN TRANSACTION;
    INSERT INTO dbo.SimpleOrders(custid, empid, orderdate)
      VALUES (68,9,'2006-07-12');
    INSERT INTO dbo.SimpleOrderDetails(orderid,productid,unitprice,qty)
      VALUES (1, 2,15.20,20);
END TRY
BEGIN CATCH
```

```
    SELECT ERROR_NUMBER() AS ErrNum, ERROR_MESSAGE() AS ErrMsg;
    ROLLBACK TRANSACTION;
END CATCH;
```

A fellow query writer is now receiving errors resulting from locks on database records. What can you do to troubleshoot this problem?

Show solution Reset

Issue a COMMIT TRANSACTION statement. In the script, you have forgotten to issue a COMMIT TRANSACTION statement. Therefore, the transaction remains open and the database has locked records in the SimpleOrders and SimpleOrderDetails tables.

Lab: Implementing Transactions

Scenario

As a junior database developer for Adventure Works, you will be creating stored procedures using corporate databases stored in SQL Server. To create more robust procedures, you will be implementing transactions in your code.

Objectives

After completing this lab, you will be able to:

- Control transactions.
- Add error handling to a CATCH block.

Lab Setup

Estimated Time: 30 minutes

Virtual machine: **20761C-MIA-SQL**

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Controlling Transactions with BEGIN, COMMIT, and ROLLBACK

Scenario

The IT department has supplied different examples of INSERT statements to practice executing multiple statements inside one transaction. You will practice how to start a transaction, commit or abort it, and return the database to its state before the transaction.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment

2. Commit a Transaction
3. Delete the Previously Inserted Rows from the HR.Employees Table
4. Open a Transaction and Use the ROLLBACK Statement
5. Clear the Modifications Against the HR.Employees Table

Detailed Steps ▲

Task 1: Prepare the Lab Environment

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **Setup.cmd** in the **D:\Labfiles\Lab18\Starter** folder as Administrator.

Detailed Steps ▲

Task 2: Commit a Transaction

1. In SQL Server Management Studio, open the project file **D:\Labfiles\Lab18\Starter\Project\Project.ssmssln** and the T-SQL script 51 - Lab Exercise 1.sql. Ensure that you are connected to the TSQL database.
2. The IT department has provided the following T-SQL code:

```
INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate,
hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Johnson', N'Test 1', N'Sales Manager', N'Mr.', '19700101', '20110101', N'Some
Address 18', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 113322', 2);

INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate,
hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Robertson', N'Test 2', N'Sales Representative', N'Mr.', '19850101',
'20110601', N'Some Address 22', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386)
553344', 10);
```

3. This code inserts two rows into the HR.Employees table. By default, SQL Server treats each individual statement as a transaction. In other words, by default, SQL Server automatically commits the transaction at the end of each individual statement. In this case, the default behavior would be two transactions because you have two INSERT statements. (Do not worry about the details of the INSERT statements because they are only meant to provide sample code for the transaction scenario.)

4. In this example, you would like to control the transaction and execute both INSERT statements inside one transaction.

5. Before the supplied T-SQL code, write a statement to open a transaction. After the supplied INSERT statements, write a statement to commit the transaction. Highlight all of the T-SQL code and execute it.

6. Observe and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab18\Solution\52 - Lab Exercise 1 - Task 1_1 Result.txt.

7. Write a SELECT statement to retrieve the empid, lastname, and firstname columns from the HR.Employees table. Order the employees by the empid column in a descending order. Execute the SELECT statement.

8. Observe and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab18\Solution\53 - Lab Exercise 1 - Task 1_2 Result.txt. Notice the two new rows in the result set.

Detailed Steps ▲

Task 3: Delete the Previously Inserted Rows from the HR.Employees Table

1. Execute the provided T-SQL code to delete rows inserted from the previous task:

```
DELETE HR.Employees
WHERE empid IN (10, 11);
DBCC CHECKIDENT ('HR.Employees', RESEED, 9);
```

Note that this is cleanup code that will not be explained in this course.

Detailed Steps ▲

Task 4: Open a Transaction and Use the ROLLBACK Statement

1. The IT department has provided T-SQL code (which happens to the same code as in task 1). Before the provided T-SQL code, write a statement to start a transaction.

2. Highlight the written statement and the provided T-SQL code, and execute it.

3. Write a SELECT statement to retrieve the empid, lastname, and firstname columns from the HR.Employees table. Order the employees by the empid column.

4. Execute the written SELECT statement and notice the two new rows in the result set.

5. Observe and compare the results that you achieved with the desired results shown in the file

D:\Labfiles\Lab18\Solution\54 - Lab Exercise 1 - Task 3_1 Result.txt.

6. After the written SELECT statement, write a ROLLBACK statement to cancel the transaction. Only execute the ROLLBACK statement.
7. Highlight this and execute the written SELECT statement against the HR.Employees table again.
8. Observe and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab18\Solution\55 - Lab Exercise 1 - Task 3_2 Result.txt. Notice that the two new rows are no longer present in the table.

Detailed Steps ▲

Task 5: Clear the Modifications Against the HR.Employees Table

- Execute the provided T-SQL code:

```
DBCC CHECKIDENT ('HR.Employees', RESEED, 9);
```

Result: After this exercise, you should be able to control a transaction using the BEGIN TRAN, COMMIT, and ROLLBACK statements.

Exercise 2: Adding Error Handling to a CATCH Block

Scenario

In the previous module, you learned how to add error handling to T-SQL code. Now you will practice how to properly control a transaction by testing to see if an error occurred.

The main tasks for this exercise are as follows:

1. Observe the Provided T-SQL Code
2. Delete the Previously Inserted Row in the HR.Employees Table
3. Abort Both INSERT Statements If an Error Occurs
4. Clear the Modifications Against the HR.Employees Table

Detailed Steps ▲

Task 1: Observe the Provided T-SQL Code

1. Open the T-SQL script 61 - Lab Exercise 2.sql. Ensure that you are connected to the TSQL database.
2. The IT department has provided T-SQL code that is similar to the code in the previous exercise:

```
SELECT empid, lastname, firstname
FROM HR.Employees
ORDER BY empid DESC;

GO

BEGIN TRAN;

INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate,
hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Johnson', N'Test 1', N'Sales Manager', N'Mr.', '19700101', '20110101', N'Some
Address 18', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 113322', 2);

INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate,
hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Robertson', N'Test 2', N'Sales Representative', N'Mr.', '19850101',
'10110601', N'Some Address 22', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386)
553344', 10);

COMMIT TRAN;
```

3. Execute only the SELECT statement.
4. Observe and compare the results that you achieved with the desired results shown in the file 62 - Lab Exercise 2 - Task 1_1 result.txt. Notice the number of employees in the HR.Employees table.
5. Execute the part of the T-SQL code that starts with a BEGIN TRAN statement and ends with the COMMIT TRAN statement. You will get a conversion error in the second INSERT statement.
6. Again, execute only the SELECT statement.
7. Observe and compare the results that you achieved with the desired results shown in the file 63 - Lab Exercise 2 - Task 1_2 Result.txt. Notice that, although an error showed inside the transaction block, one new row was added to the HR.Employees table based on the first INSERT statement.

Detailed Steps ▲

Task 2: Delete the Previously Inserted Row in the HR.Employees Table

- Execute the provided T-SQL code to delete the row inserted from the previous task:

```
DELETE HR.Employees
WHERE empid IN (10, 11);DBCC CHECKIDENT ('HR.Employees', RESEED, 9);
```

Detailed Steps ▲

Task 3: Abort Both INSERT Statements If an Error Occurs

1. Modify the provided T-SQL code to include a TRY/CATCH block that rolls back the entire transaction if any of the INSERT statements throws an error:

```
BEGIN TRAN;

INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate,
hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Johnson', N'Test 1', N'Sales Manager', N'Mr.', '19700101', '20110101', N'Some
Address 18', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386) 113322', 2);

INSERT INTO HR.Employees (lastname, firstname, title, titleofcourtesy, birthdate,
hiredate, address, city, region, postalcode, country, phone, mgrid)
VALUES (N'Robertson', N'Test 2', N'Sales Representative', N'Mr.', '19850101',
'10110601', N'Some Address 22', N'Ljubljana', NULL, N'1000', N'Slovenia', N'(386)
553344', 10);

COMMIT TRAN;
```

2. In the CATCH block, include a PRINT statement that prints the message “Rollback the transaction…” if an error occurred and the message “Commit the transaction…” if no error occurred.
3. Execute the modified T-SQL code.
4. Observe and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab18\Solution\64 - Lab Exercise 2 - Task 3_1 Result.txt.
5. Write a SELECT statement against the HR.Employees table to see if any new rows were inserted (like you did in exercise 1). Execute the SELECT statement.
6. Observe and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab18\Solution\65 - Lab Exercise 2 - Task 3_2 Result.txt.

Detailed Steps ▲

Task 4: Clear the Modifications Against the HR.Employees Table

- Execute the provided T-SQL code:

```
DBCC CHECKIDENT ('HR.Employees', RESEED, 9);
```


Result: After this exercise, you should have a basic understanding of how to control a transaction inside a TRY/CATCH block to efficiently handle possible errors.

Module Review and Takeaways

In this module, you have learned how to:

- Describe transactions and the differences between batches and transactions.
- Describe batches and how they are handled by SQL Server.
- Create and manage transactions with transaction control language (TCL) statements.
- Use SET XACT_ABORT to define SQL Server's handling of transactions outside TRY/CATCH blocks.

Review Question(s)

Check Your Knowledge

Discovery

What happens to a nested transaction when the outer transaction is rolled back?

Show solution Reset

The inner transaction is also rolled back, so nested transactions are not typically useful in user code.

Check Your Knowledge

Discovery

When a runtime error occurs in a transaction and SET XACT_ABORT is ON, is the transaction always automatically rolled back?

Show solution Reset

No, not if the error occurs within a TRY block.