

Module 18

Implementing Transactions

Module Overview

- Transactions and the Database Engine
- Controlling Transactions

Lesson 1: Transactions and the Database Engine

- Defining Transactions
- The Need for Transactions: Issues with Batches
- Transactions Extend Batches
- Demonstration: Transactions and the Database Engine

Defining Transactions

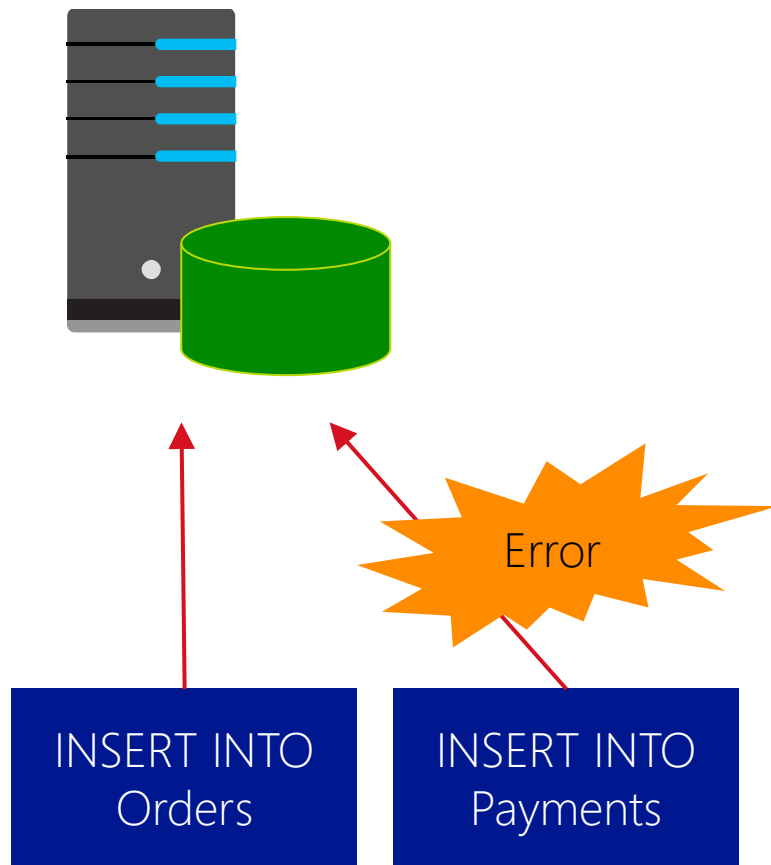
- A transaction is a group of tasks defining a unit of work
- The entire unit must succeed or fail together—no partial completion is permitted

```
--Two tasks that make up a unit of work  
INSERT INTO Sales.Orders ...  
INSERT INTO Sales.OrderDetails ...
```

- Individual data modification statements are automatically treated as stand-alone transactions
- User transactions can be managed with T-SQL commands:
 - BEGIN/ COMMIT/ROLLBACK TRANSACTION
- SQL Server uses locking mechanisms and the transaction log to support transactions

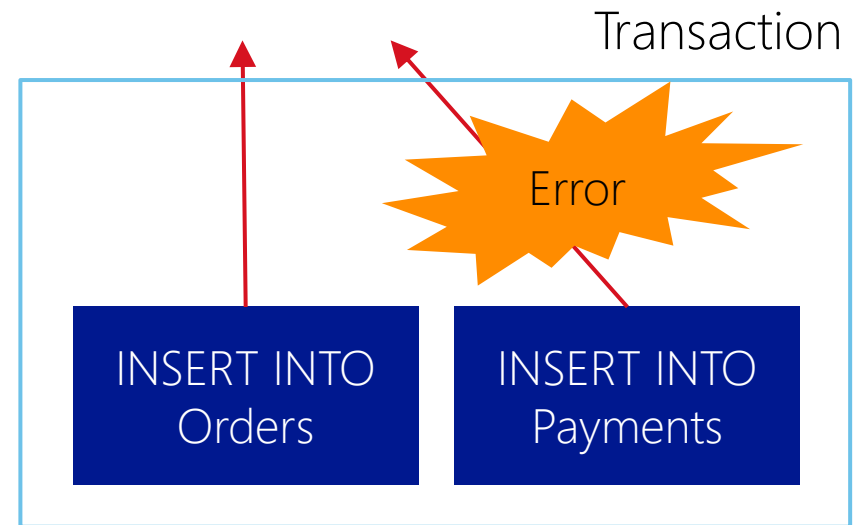
Understanding Transactions

Without Transactions:



Result: Order with no Payment

With Transactions:



Result: No Order and no Payment

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;
```

Transactions Extend Batches

- Transaction commands identify blocks of code that must succeed or fail together and provide points where the database engine can roll back, or undo, operations:

```
BEGIN TRY
    BEGIN TRANSACTION
        INSERT INTO Sales.Orders ... --Insert succeeds
        INSERT INTO Sales.OrderDetails ... --Insert fails
    COMMIT TRANSACTION -- If no errors, transaction
    -- completes
END TRY
BEGIN CATCH
    --Inserted rows still exist in Sales.Orders Table
    SELECT ERROR_NUMBER()
    ROLLBACK TRANSACTION --Any transaction work undone
END CATCH;
```

Demonstration: Transactions and the Database Engine

In this demonstration, you will see how to:

- Use transactions

Lesson 2: Controlling Transactions

- BEGIN TRANSACTION
- COMMIT TRANSACTION
- ROLLBACK TRANSACTION
- Using XACT_ABORT
- Demonstration: Controlling Transactions

BEGIN TRANSACTION

- BEGIN TRANSACTION marks the starting point of an explicit, user-defined transaction
- Transactions last until a COMMIT statement is issued, a ROLLBACK is manually issued, or the connection is broken and the system issues a ROLLBACK
- Transactions are local to a connection and cannot span connections
- In your T-SQL code, mark the start of the transaction's work:

```
BEGIN TRY
    BEGIN TRANSACTION -- marks beginning of work
    INSERT INTO Sales.Orders ... --transacted work
    INSERT INTO Sales.OrderDetails ... --transacted work
    ...
```

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
```

ROLLBACK TRANSACTION

- A ROLLBACK statement undoes all modifications made in the transaction by reverting the data to the state it was in at the beginning of the transaction
- ROLLBACK frees resources, such as locks, held by the transaction
- Before rolling back, you can test the state of the transaction with the XACT_STATE function
- In your T-SQL code, if an error occurs, ROLLBACK to the point of the BEGIN TRANSACTION statement

```
BEGIN CATCH
    SELECT ERROR_NUMBER() --sample error handling
    ROLLBACK TRAN
END CATCH;
```

Using XACT_ABORT

- SQL Server does not automatically roll back transactions when errors occur
- To roll back, either use ROLLBACK statements in error-handling logic or enable XACT_ABORT
- XACT_ABORT specifies whether SQL Server automatically rolls back the current transaction when a runtime error occurs
 - When SET XACT_ABORT is ON, the entire transaction is terminated and rolled back on error, unless occurring in TRY block
 - SET XACT_ABORT OFF is the default setting
- Change XACT_ABORT value with the SET command:

```
SET XACT_ABORT ON;
```

Demonstration: Controlling Transactions

In this demonstration, you will see how to:

- Control transactions

Lab: Implementing Transactions

- Exercise 1: Controlling Transactions with BEGIN, COMMIT, and ROLLBACK
- Exercise 2: Adding Error Handling to a CATCH Block

Logon Information

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Estimated Time: 30 minutes

Lab 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.

Module Review and Takeaways

- Review Question(s)

Course Evaluation

- Your evaluation of this course will help Microsoft understand the quality of your learning experience.
- Please work with your training provider to access the course evaluation form.
- Microsoft will keep your answers to this survey private and confidential and will use your responses to improve your future learning experience. Your open and honest feedback is valuable and appreciated.