Module 2: Introduction to T-SQL Querying

Contents:

Module Overview

Transact-SQL, or T-SQL, is the language you will use to interact with Microsoft® SQL Server®. In this module, you will learn that T-SQL has many elements in common with other computer languages, such as commands, variables, loops, functions, and operators. You will also learn that designing your queries to take sets into account means SQL Server will perform at its best. To make the most of your effort in writing T-SQL, you will also learn the process and order by which SQL Server evaluates your queries. Understanding the logical order for operations of SELECT statements is vital to learning how to write effective queries.

Objectives

After completing this module, you will be able to describe:

- The elements of T-SQL and their role in writing queries.
- The use of sets in SQL Server.
- The use of predicate logic in SQL Server.
- The logical order of operations in SELECT statements.

Lesson 1: Introducing T-SQL

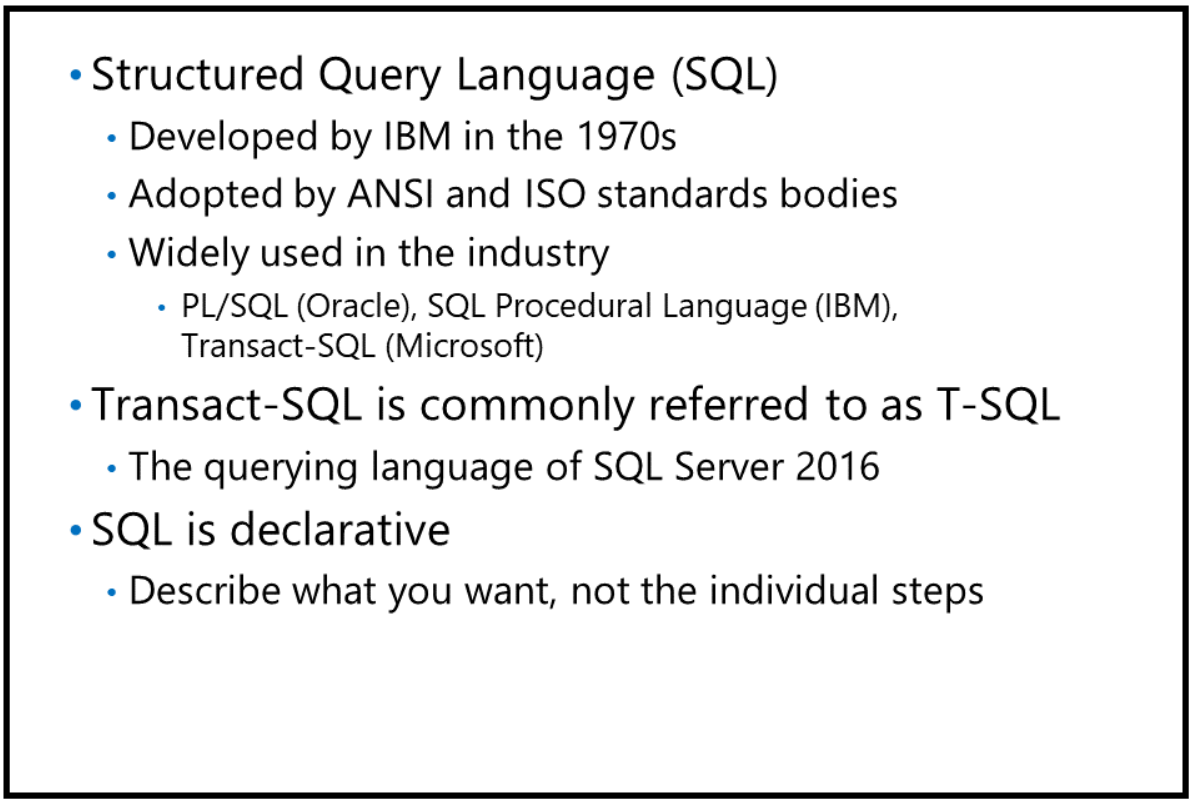
In this lesson, you will learn the role of T-SQL in writing SELECT statements. You will also learn about many of the T-SQL language elements and which ones are useful for writing queries.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe Microsoft's implementation of the standard SQL language.
- Categorize SQL statements into their dialects.
- Identify the elements of T-SQL, including predicates, operators, expressions, and comments.

About T-SQL



T-SQL is Microsoft's implementation of the industry standard Structured Query Language. The language was originally developed to support the new relational data model at International Business Machines (IBM) in the early 1970s. Since then, SQL has become widely adopted in the industry. SQL became a standard of the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) in the 1980s.

The ANSI standard has gone through several revisions, including SQL-89 and SQL-92—the specifications are either fully or partly supported by T-SQL. SQL Server also implements features from later standards, such as ANSI SQL-2008 and ANSI SQL-2011. Microsoft, like many vendors, has also extended the language to include SQL Server-specific features and functions.

Besides Microsoft's implementation as T-SQL in SQL Server, Oracle has PL/SQL, IBM has SQL PL, and Sybase maintains its own T-SQL operation.

An important concept to understand when working with T-SQL is that it is not a procedural language but a set-based and declarative language. When you write a query to retrieve data from SQL Server, you describe the data you wish to display; you do not tell SQL Server exactly how to retrieve it. Instead of supplying a procedural list of steps to take, you provide the attributes of the data you seek.

Procedural Approach

```
Loop through each row in the data.
If the city is Portland, return the row; otherwise do nothing.
Move to next row.
End loop.
```

This procedural code has to contain the logic to retrieve the data—to inspect the data to see if it meets your needs—and will be doing this for all the data in the table, whether or not it is relevant.

With a declarative language such as T-SQL, you will provide the attributes and values that describe the set you wish to retrieve. You do not have to specify how to retrieve the data, but you should identify what the data is.

Declarative Language

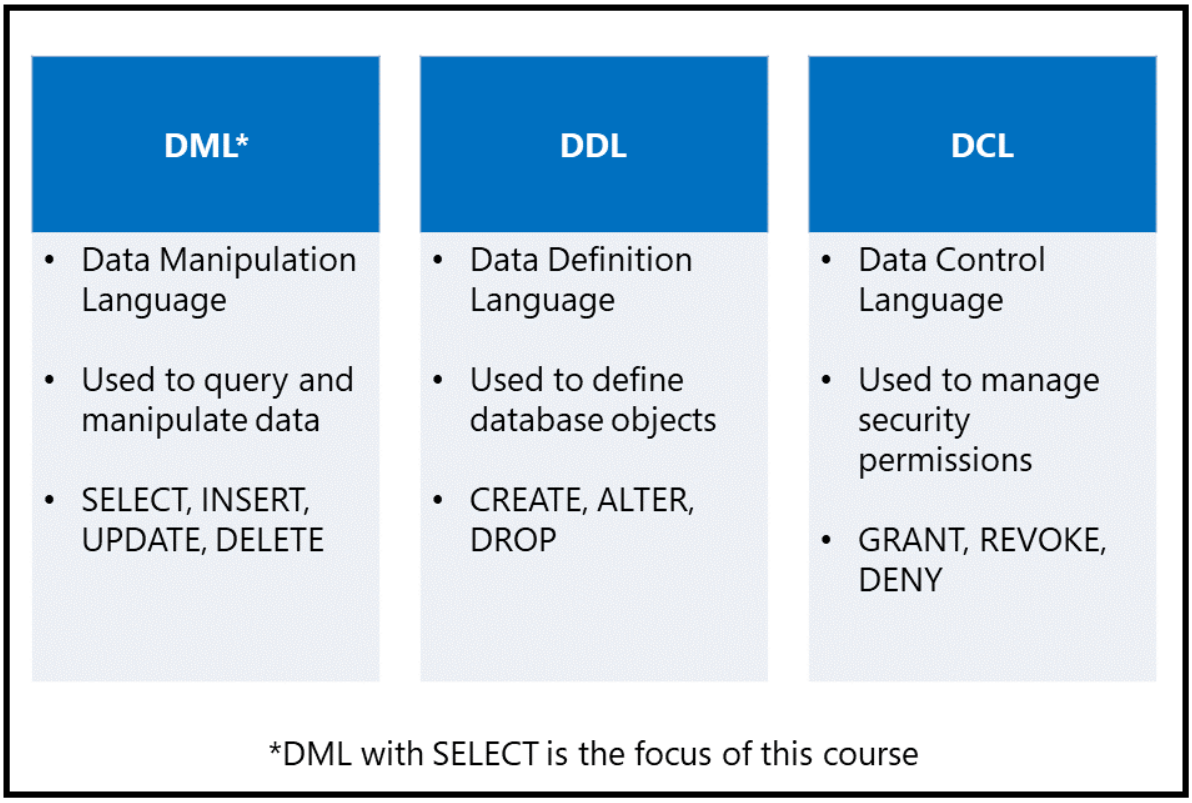
```
Return all the customers whose city is Portland
```

With T-SQL, the SQL Server Database Engine will determine the optimal path to access the data and return a matching set. Your role is to learn to write efficient and accurate T-SQL code so you can properly describe the set you wish to retrieve.

If you have a background in other programming environments, adopting a new mindset may present a challenge. This course has been designed to help you bridge the gap between procedural and set-based, declarative T-SQL.

> **Note:** Sets will be discussed later in this module.

Categories of T-SQL Statements



T-SQL statements can be organized into three categories:

- **Data Manipulation Language** (DML) is the set of T-SQL statements that focuses on querying and modifying data. This includes SELECT, the primary focus of this course, and modification statements such as INSERT, UPDATE, and DELETE. You will learn about SELECT statements throughout this course.
- **Data Definition Language** (DDL) is the set of T-SQL statements that handles the definition and life cycle of database objects, such as tables, views, and procedures. This includes statements such as CREATE, ALTER, and DROP.
- **Data Control Language** (DCL) is the set of T-SQL statements used to manage security permissions for users and objects. DCL includes statements such as GRANT, REVOKE, and DENY.

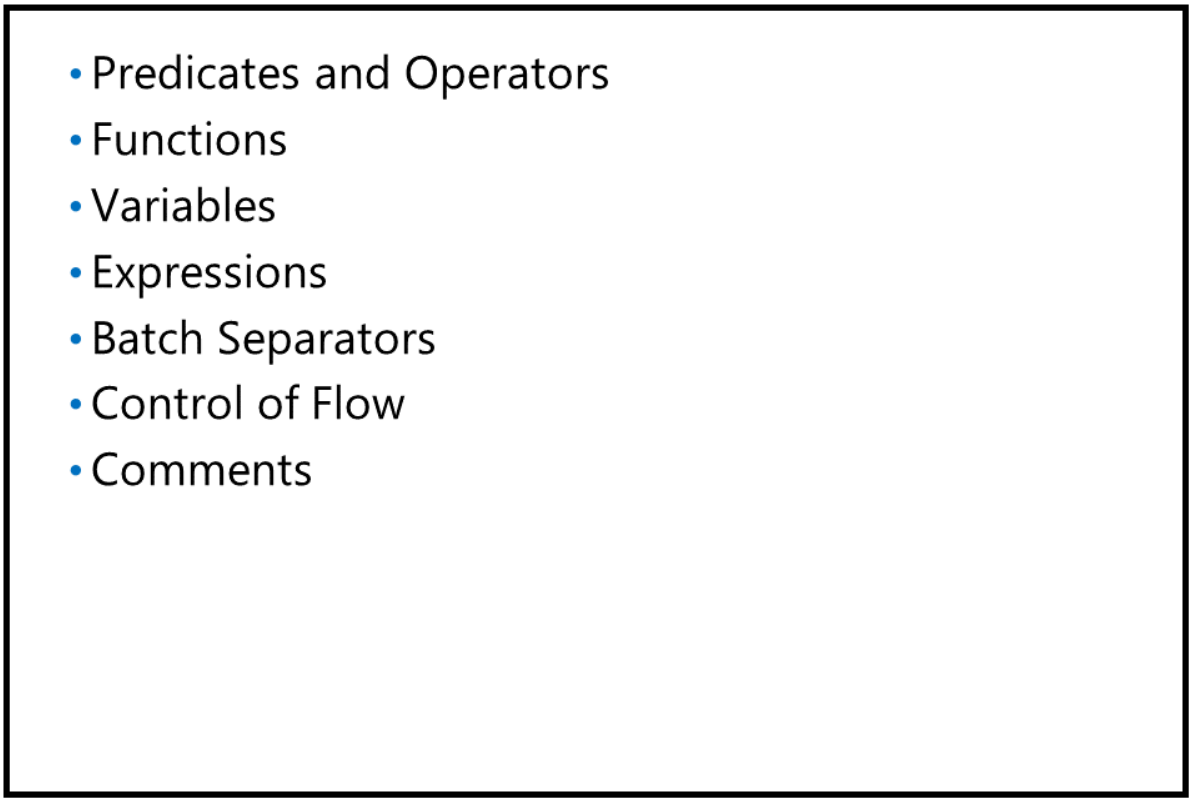
> **Note:** DCL commands are beyond the scope of this course. For more information about SQL Server security, including DCL, see the Microsoft Official Course 20473-2*: Administering a SQL Database Infrastructure*.

For additional information on DML, DDL, and DCL commands, see Microsoft Docs:

Transact-SQL Reference (Database Engine)

http://aka.ms/hjmhuj

T-SQL Language Elements



Like many programming languages, T-SQL contains elements that you will use in queries. You will use predicates to filter rows; operators to perform comparisons; functions and expressions to manipulate data or retrieve system information; and comments to document your code. If you need to go beyond writing SELECT statements to create stored procedures, triggers, and other objects, you might use elements such as control-of-flow statements, variables to temporarily store values, and batch separators. The next several topics in this lesson will introduce you to many of these elements.

> **Note:** The purpose of this lesson is to introduce common elements of the T-SQL language, which will be presented here at a high conceptual level. Subsequent modules in this course will show more detailed explanations.

T-SQL Language Elements: Predicates and Operators

Elements:	Predicates and Operators:
Predicates	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
Comparison Operators	=, >, <, >=, <=, <>, !=, !>, !<
Logical Operators	AND, OR, NOT
Arithmetic Operators	*, /, %, +, -,
Concatenation	+

The T-SQL language provides elements for specifying and evaluating logical expressions. In SELECT statements, you can use logical expressions to define filters for WHERE and HAVING clauses. You will write these expressions using predicates and operators.

Predicates supported by T-SQL include the following:

- **IN**: used to determine whether a value matches any value in a list or subquery.For example, WHERE day IN (1,5,6,10).
- **BETWEEN**: used to specify a range of values.For example, WHERE rate BETWEEN 3 AND 7.
- **LIKE**: used to match characters against a pattern.For example, WHERE surname LIKE '%mith%'.

Operators include several common categories:

- **Comparison**. For equality and inequality tests: =, <, >, >=, <=, !=, !>, !< (Note that !>, !< and != are not ISO standard—it is best practice to use standard options when they exist.)
- **Logical**. For testing the validity of a condition: AND, OR, NOT.
- **Arithmetic**. For performing mathematical operations: +, -, *, /, % (modulo).
- **Concatenation**. For combining character strings: +.
- **Assignment**. For setting a value: =.

As with other mathematical environments, operators are subject to rules governing precedence. The following table describes the order in which T-SQL operators are evaluated:

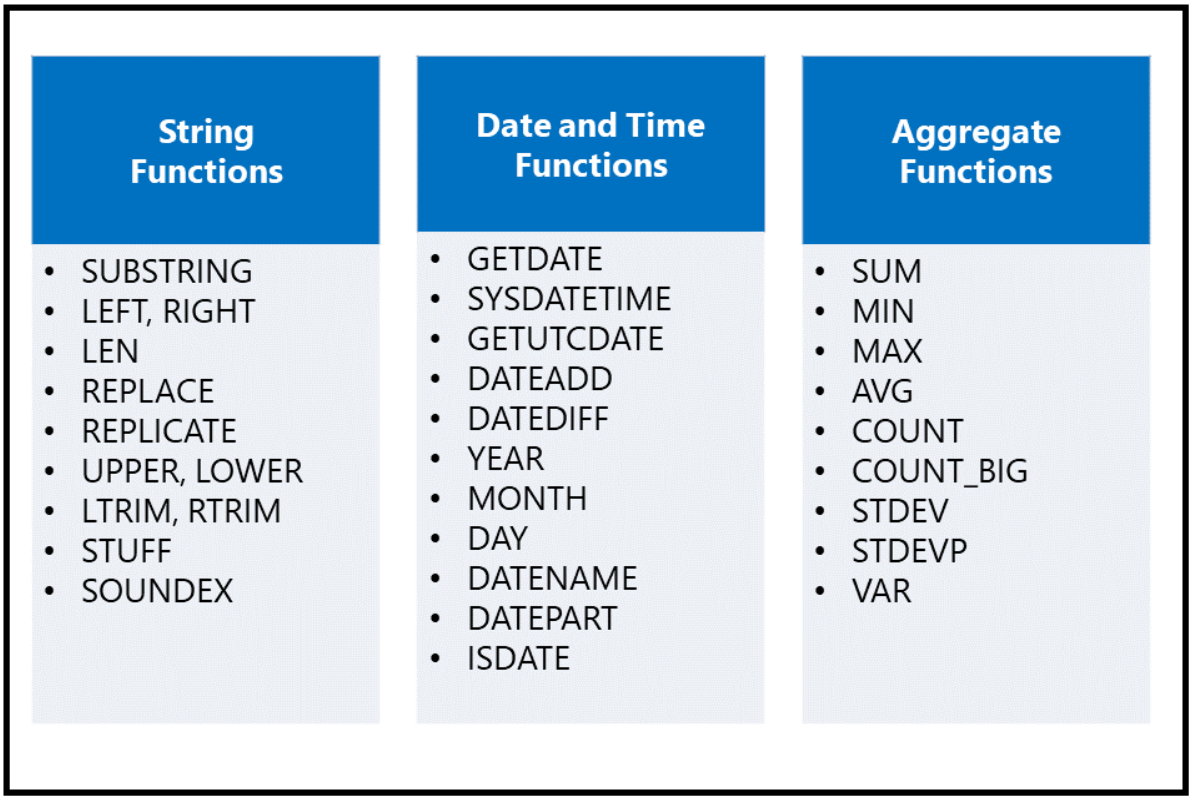
Order of Evaluation	Operator
1	~ (Bitwise NOT)
2	/, *, % (Division, Multiply, Modulo)
3	+, -, &, ^, | (Positive/Add/Concatenate, Negative/Subtract, Bitwise AND, Bitwise Exclusive OR, Bitwise OR)
4	=, >, <, <=, <, !=, !, !<, !> (Comparisons)
5	NOT
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (Assignment)

For more information on operator precedence, see Microsoft Docs:

Operator Precedence

http://aka.ms/y6ylxo

T-SQL Language Elements: Functions



SQL Server provides a wide variety of functions for your T-SQL queries. They range from scalar functions, such as SYSDATETIME, which return a single-valued result, to others that operate on and return entire sets, such as the windowing functions you will learn about later in this course.

As with operators, SQL Server functions can be organized into categories. Here are some common categories of scalar (single-value) functions available to you for writing queries:

- String functions
  - SUBSTRING, LEFT, RIGHT, LEN, DATALENGTH
  - REPLACE, REPLICATE
  - UPPER, LOWER, RTRIM, LTRIM
  - STUFF
  - SOUNDEX
- Date and time functions
  - GETDATE, SYSDATETIME, GETUTCDATE
  - DATEADD, DATEDIFF
  - YEAR, MONTH, DAY
  - DATENAME, DATEPART
  - ISDATE
- Aggregate functions
  - SUM, MIN, MAX, AVG
  - COUNT, COUNT_BIG
  - STDEV, STDEVP
  - VAR
- Mathematical functions
  - RAND, ROUND, POWER, ABS
  - CEILING, FLOOR

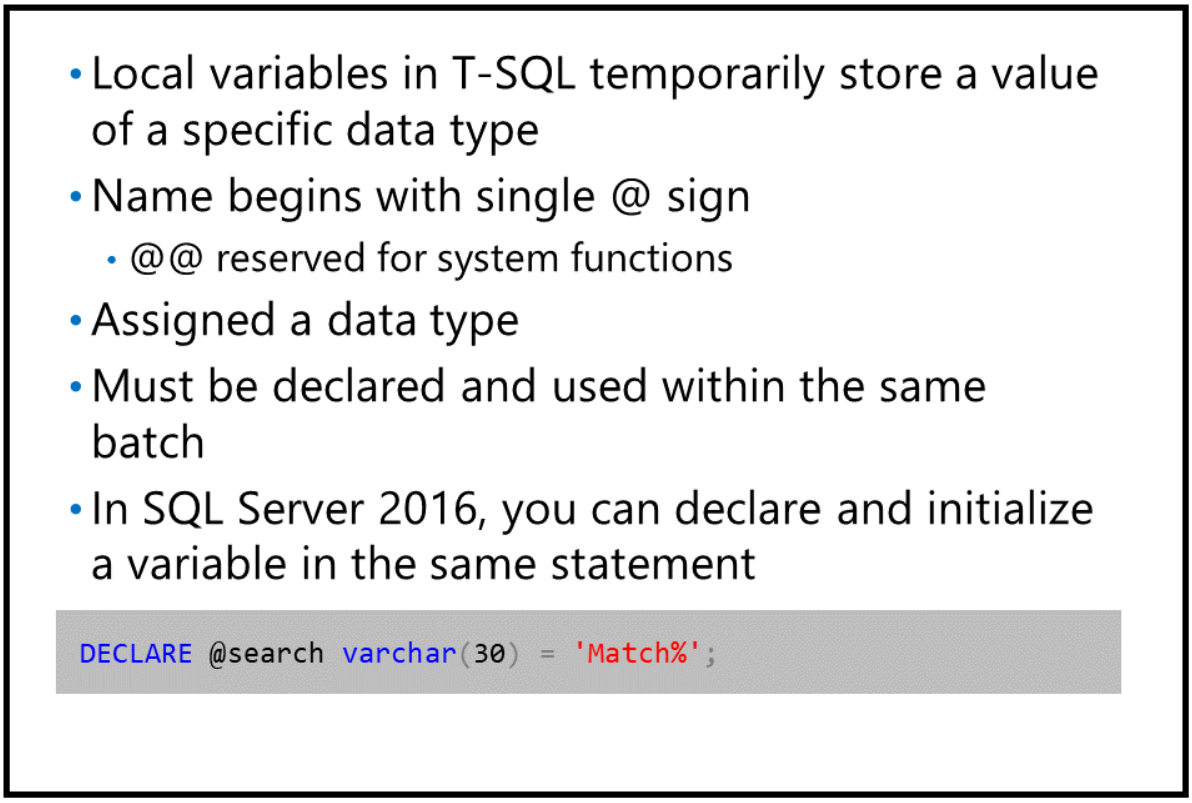
> **Note:** The purpose of this lesson is to introduce many elements of the T-SQL language, which is presented here at a high conceptual level. Subsequent modules in this course will show more detailed explanations.

For more information, including code samples, see Microsoft Docs:

Built-in Functions

http://aka.ms/jw8w5j

T-SQL Language Elements: Variables



Like many programming languages, T-SQL provides a means of temporarily storing a value of a specific data type. However, unlike other programming environments, all user-created variables are local to the T-SQL batch that created them—and are visible only to that batch. There are no global or public variables available to SQL Server users.

To create a local variable in T-SQL, you must give a name, data type, and initial value. The name must start with a single @ (at) symbol, and the data type must be system-supplied or user-defined, and stored in the database your code will run against.

> **Note:** You may find references in SQL Server literature, websites, and so on, to so-called "system variables," named with a double @@, such as @@ERROR. It is more accurate to refer to these as system functions, because users may not assign a value to them. This course will differentiate user variables prefixed with a single @ from system functions prefixed with @@.

You can name and initialize a variable in the same statement. If your variable is not initialized in the DECLARE statement, it will be created with a value of NULL, and you can subsequently assign a value with the SET statement.

Character Variable

```
DECLARE @search varchar(30) = 'Match%';
```

Date Variable

```
DECLARE @CurrentDate date;
SET @CurrentDate = GETDATE();
```

You will learn more about different data types—including dates—and T-SQL variables later in this course.

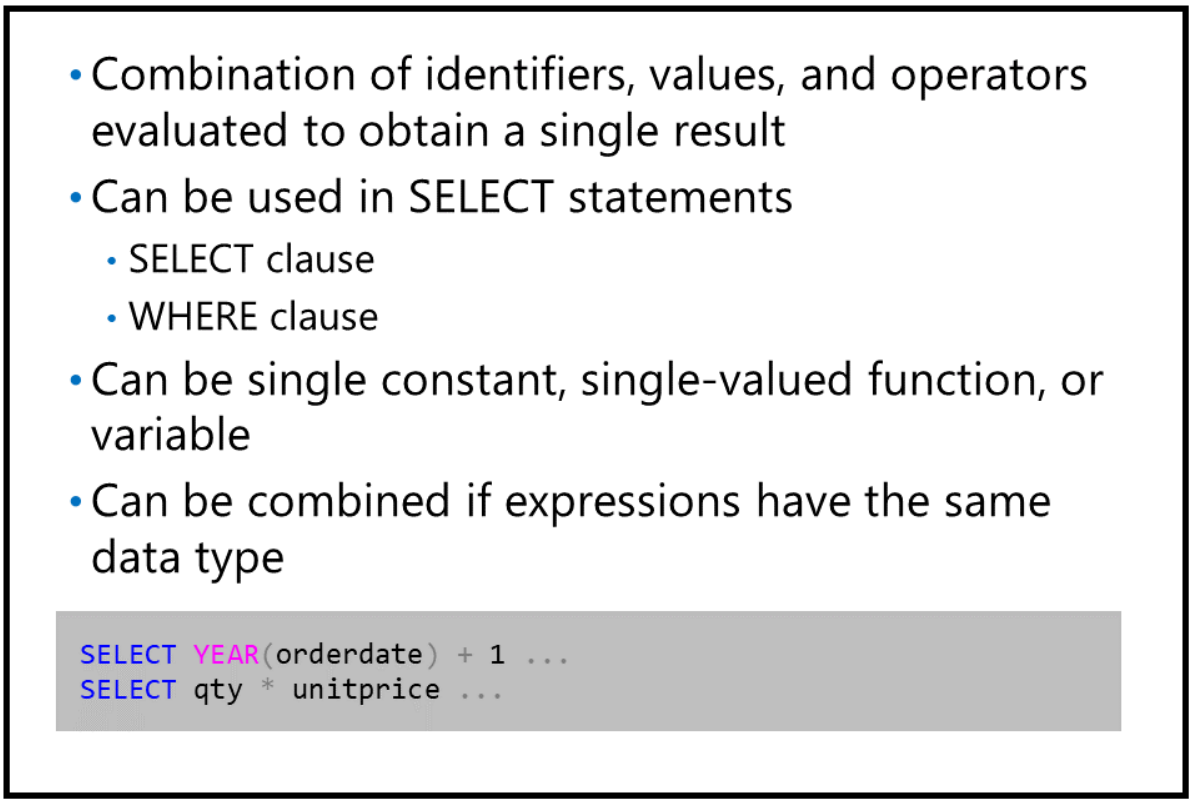
If persistent storage or global visibility for a value is needed, consider creating a table. SQL Server provides both temporary and permanent storage in databases.

For more information on different types of tables, see Microsoft Docs:

Tables

http://aka.ms/quew7f

T-SQL Language Elements: Expressions



T-SQL provides combinations of identifiers, symbols, and operators that are evaluated by SQL Server to return a single result. These combinations are known as expressions, offering a useful and powerful tool for your queries. In SELECT statements, you may use expressions:

- In the SELECT clause to operate on and/or manipulate columns.
- As CASE expressions to replace values matching a logical expression with another value.
- In the WHERE clause to construct predicates for filtering rows.
- As table expressions to create temporary sets used for further processing.

> **Note:** The purpose of this lesson is to introduce many elements of the T-SQL language, which will be presented here at a high conceptual level.

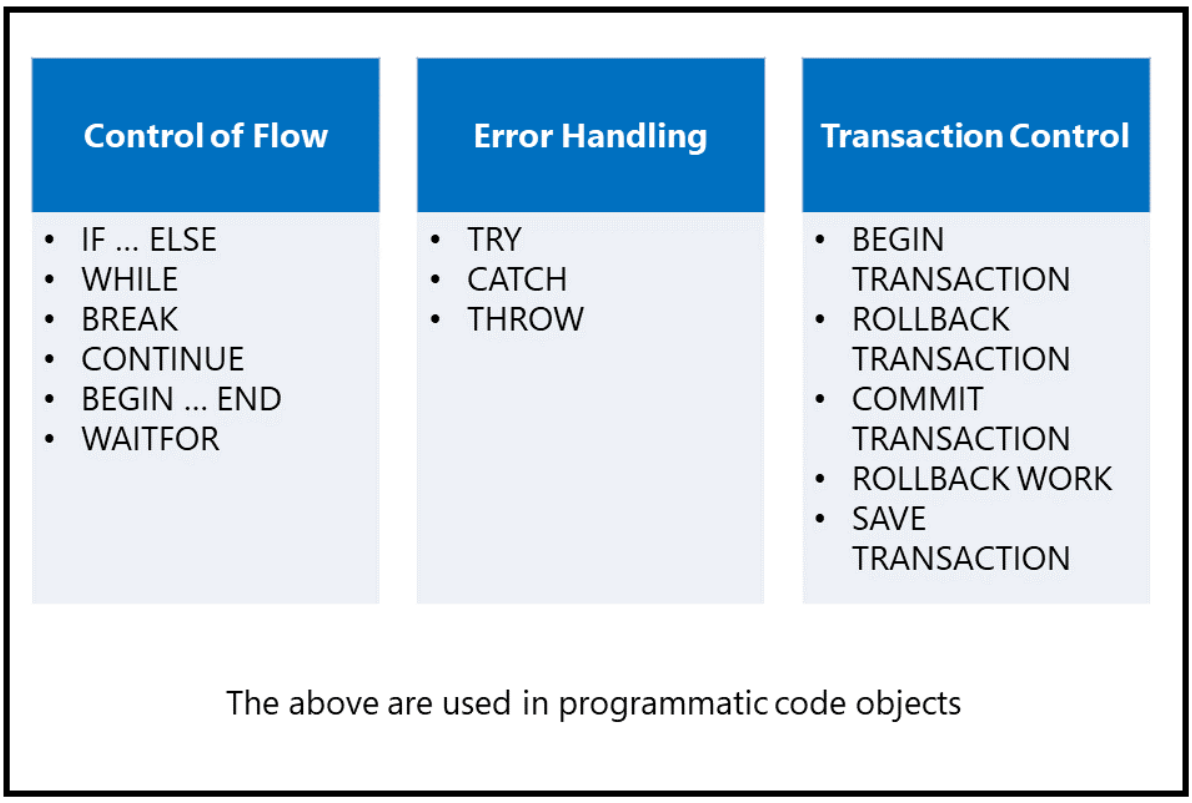
Expressions may be based on a scalar (single-value) function, on a constant value, or on variables. Multiple expressions may be joined using operators if they have the same data type, or if the data type can be converted from a lower precedence to a higher precedence (for example, int to money).

Expression

```
SELECT YEAR(orderdate) AS currentyear, YEAR(orderdate) + 1 AS nextyear
FROM Sales.Orders;
```

> **Note:** The preceding example uses T-SQL techniques, such as column aliases and date functions, which will be covered later in this course.

T-SQL Language Elements: Control of Flow, Errors, and Transactions



While T-SQL is primarily a data retrieval language and not a procedural language, it does support a limited set of statements that provide some control of flow during execution.

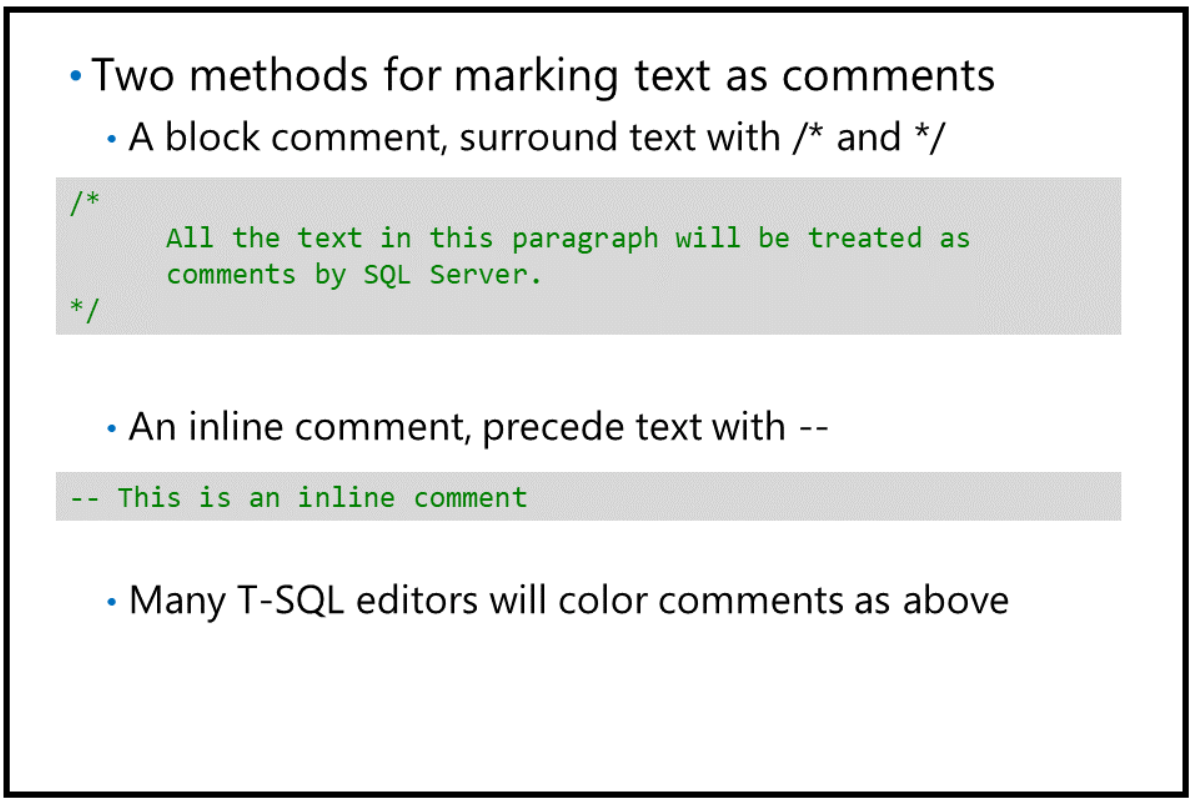
Some of the commonly-used control-of-flow statements include:

- **IF ... ELSE**, for providing branching control based on a logical test.
- **WHILE**, for repeating a statement or block of statements while a condition is true.

- **BEGIN ... END**, for defining the extents of a block of T-SQL statements.
- **TRY ... CATCH**, for defining the structure of exception handling (error handling).
- **THROW**, for raising an exception and transferring execution to a CATCH block.
- **BEGIN TRANSACTION**, for marking a block of statements as part of an explicit transaction. Ended by COMMIT TRANSACTION or ROLLBACK TRANSACTION.

> **Note:** Control-of-flow operators are not used in stand-alone queries. For example, if your primary role is as a report writer, it is unlikely that you will need to use them. However, if your responsibilities include creating objects such as stored procedures and triggers, you will use these elements.

T-SQL Language Elements: Comments



T-SQL has two methods for documenting code or instructing the database engine to ignore certain statements. Which method you use will typically depend on the number of lines of code you want ignored:

- For single lines, or very few lines of code, use the -- (double dash) to precede the text to be marked as a comment. Any text following the dashes will be ignored by SQL Server.
- For longer blocks of code, enclose the text between /* and */ characters. Any code between the characters will be ignored by SQL Server.

The following example uses the -- (double dash) method to mark comments:

Single Line Comments

```
-- This whole line is a comment
DECLARE @search varchar(30) = 'Match%'; -- end of a line
```

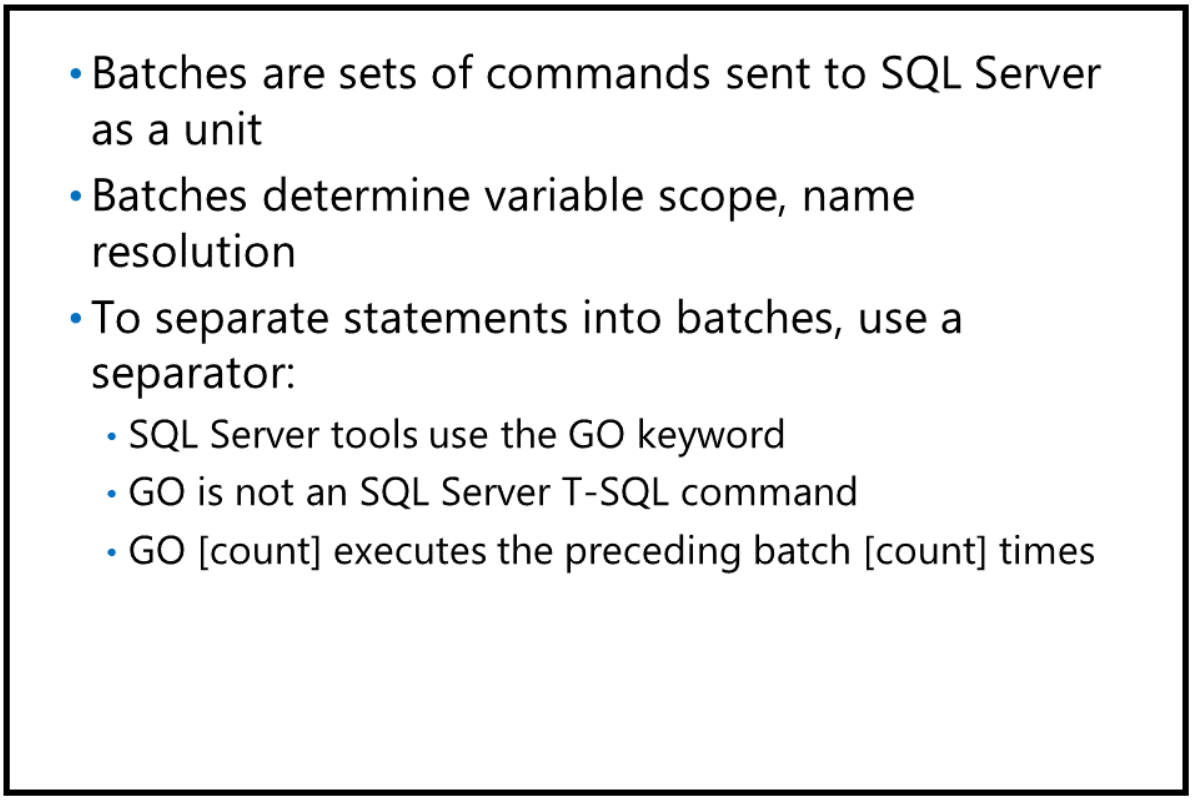
The following example uses the /* comment block */ method to mark comments:

Block Comment

```
/*
      All the text in this paragraph will be treated as comments
      by SQL Server.
*/
```

Many query editing tools, such as SQL Server Management Studio (SSMS), Visual Studio®, or SQLCMD, will color-code commented text in a different color from the surrounding T-SQL code. In SSMS, use the Tools, Options dialog box to customize the colors and fonts in the T-SQL script editor.

T-SQL Language Elements: Batch Separators



SQL Server client tools, such as SSMS, send commands to the database engine in sets called batches. If you are manually executing code, such as in a query editor, you can choose whether to send all the text in a script as one batch. You may also choose to insert separators between certain sections of code.

The specification of a batch separator is handled by your client tool. For example, the keyword GO is the default batch separator in SSMS. You can change this for the current query in Query | Query Options or globally in Tools | Options | Query Execution.

For most simple query purposes, batch separators are not used, because you will be submitting a single query at a time. However, when you need to create and manipulate objects, you might need to separate statements into distinct batches. For example, a CREATE VIEW statement might not be included in the same batch as other statements.

Code That Requires Multiple Batches

```
CREATE TABLE table1 (col1 int);
CREATE VIEW view1 as SELECT * FROM table1;
```

The previous example returns the following error:

```
Msg 111, Level 15, State 1, Line 2
'CREATE VIEW' must be the first statement in a query batch.
```

Note that user-declared variables are considered local to the batch in which they are declared. If a variable is declared in one batch and referenced in another, the second batch would fail. Insert a GO batch separator between the two CREATE statements to resolve the previous error.

Local Variable

```
DECLARE @cust int = 5;

SELECT custid, companyname, contactname
FROM Sales.Customers
WHERE custid = @custid;
```

However, if a batch separator was inserted between the variable declaration and the query in which the variable is used, an error would occur.

Variable Separated by Batch

```
DECLARE @cust int = 5;
GO
SELECT custid, companyname, contactname
FROM Sales.Customers
WHERE custid = @custid;
```

The previous example returns the following error:

```
Msg 137, Level 15, State 2, Line 5
Must declare the scalar variable "@custid".
```

Demonstration: T-SQL Language Elements

In this demonstration, you will see how to use T-SQL language elements.

Note that some elements will be covered in more depth in later modules.

Demonstration Steps

Use T-SQL Language Elements

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. In File Explorer, browse to **D:\Demofiles\Mod02**, right-click **Setup.cmd**, and then click **Run as administrator**.
3. In the **User Account Control** dialog box, click **Yes**.
4. When the script has finished, press any key.
5. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows® Authentication.
6. On the **File** menu, point to **Open**, and then click **Project/Solution**.
7. In the **Open Project** dialog box, browse to the **D:\Demofiles\Mod02\Demo** folder, and then double-click **Demo.ssmssln**.
8. In Solution Explorer, expand **Queries**, and then double-click the **11 - Demonstration A.sql** script file.
9. Select the code under **Step 1**, and then click **Execute**.
10. Select the code under **Step 2**, and then click **Execute**.
11. Select the code under **Step 3**, and then click **Execute**.
12. Select the code under **Step 4**, and then click **Execute**.
13. Select the code under **Step 5**, and then click **Execute**.
14. Select the code under **Step 6**, and then click **Execute**.
15. Select the code under **Step 7**, and then click **Execute**.
16. Select the code under **Step 8**, and then click **Execute**.
17. Select the code under the comment **Cleanup task if needed**, and then click **Execute**.

18. Close SQL Server Management Studio.

Check Your Knowledge

Select the best answer

From the following T-SQL elements, select the one that does not contain an expression:

```
SELECT FirstName, LastName, SkillName AS Skill, GetDate() - DOB AS Age

WHERE HumanResources.Department.ModifiedDate > (SYSDATETIME() - 31)

JOIN HumanResources.Skills ON Employees.ID = Skills.EmployeeID

WHERE Skill.Level + Skill.Confidence > 10
```

Check answer Show solution Reset

An expression is a combination of identifiers, symbols, and operators that return a single result. The only element that does not contain a symbol and an operator is Option 3.

Lesson 2: Understanding Sets

This lesson introduces the concepts of the set theory, one of the mathematical underpinnings of relational databases, and helps you apply it to how you think about querying SQL Server.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the role of sets in a relational database.
- Understand the impact of sets on your T-SQL queries.
- Describe attributes of sets that may require special treatment in your queries.

Set Theory and SQL Server

Characteristics of a Set	Example
Elements of a set called Members	Customer as a member of set called Customers
Elements of a set are described by attributes	First name, Last name, Age
Elements must be unique	Customer ID

Set theory does not specify the order of its members

Set theory is one of the mathematical foundations of the relational model and is fundamental to working with SQL Server. While you might be able to make progress writing queries in T-SQL without an appreciation of sets, you may eventually have difficulty expressing some of them in a single, well-performing statement.

This lesson will set the stage for you to begin "thinking in sets" and understanding their nature. In turn, this will make it easier for you to:

- Take advantage of set-based statements in T-SQL.
- Understand why you still need to sort your query output.
- Understand why a set-based, declarative approach, rather than a procedural one, works best with SQL Server.

For our purposes, without delving into the mathematics supporting set theory, a set is defined as "a collection of definite, distinct objects considered as a whole." In terms applied to SQL Server databases, you can think of a set as a single unit (such as a table) containing zero or more members of the same type. For example, a Customer table represents a set—specifically, the set of all customers. You will see that the results of a SELECT statement also form a set, which will have important ramifications when learning about subqueries and table expressions.

As you learn more about certain T-SQL query statements, it is important to think of the entire set, instead of individual members, at all times. This will better equip you to write set-based code, instead of thinking one row at a time. Working with sets requires thinking in terms of operations that occur "all at once" instead of one at a time. Depending on your background, this may require an adjustment.

After "collection," the next critical term in our definition is "distinct," or unique. All members of a set must be unique. In SQL Server, uniqueness is typically implemented using keys, such as a primary key column.

However, when you start working with subsets of data, it's important to know how you can uniquely address each member of a set.

This brings us back to the consideration of the set as a "whole." Noted SQL language author Joe Celko suggests mentally adding the phrase "Set of all…" in front of the names of SQL objects that represent sets ("set of all customers," for example). This will help you remember that, when you write T-SQL code, you are addressing a collection of elements, not just one element at a time.

One important consideration is what is omitted from the set theory—any requirement regarding the order of elements in a set. In short, there is no predefined order in a set. Elements may be addressed (and retrieved) in any order. Applied to your queries this means that, if you need to return results in a certain order, you must use the ORDER BY clause in your SELECT statements. You will learn more about ORDER BY later in this course.

Set Theory Applied to SQL Server Queries

Application of Set Theory	Comments
Acts on all elements at once	Query the whole table
Use set-based processing	Tell the engine what you want to retrieve
Avoid cursors or loops	Do not process each item individually
Members of a set must be unique	Define unique keys in a table
No defined order to result set	Use ORDER BY clause if results need to be ordered

Given the set-based foundation of databases, there are a few considerations and recommendations to be aware of when writing efficient T-SQL queries:

- Act on the whole set at once. This translates to querying the whole table at once, instead of cursor-based or iterative processing.
- Use declarative, set-based processing. Tell SQL Server what you want to retrieve by describing its attributes, not by navigating to its position.
- When possible, ensure that you are addressing elements via their unique identifiers, such as keys. For example, write JOIN clauses referencing unique keys on one side of the relationship.
- Provide your own sorting instructions, because result sets are not guaranteed to be returned in any order.

Additional Reading: For more information on set theory and logical query processing, and its application to SQL queries, see Chapter 1 of Itzik Ben-Gan's *T-SQL Querying* (Microsoft Press, 2015).

Check Your Knowledge

Categorize Activity

Place each employee into the appropriate set. Indicate your answer by writing the set number to the right of each item.

Employees in London

Carolos LamyWorks in: LondonSkills:JavaScriptXML

Patrick LorenzenWorks in: LondonSkills:SharePoint AdministrationSQL Server Administration

Employees who know T-SQL

Jeanie SheppardWorks in: Buenos AiresSkills:C#JavaScriptT-SQL

Naiyana KunakornWorks in: Washington DCSkills:JavaScriptSQL Server AdministrationT-SQL XML

Frederic TowleWorks in: TokyoSkills:Active Directory AdministrationT-SQL

Employees in Seattle who know SQL Server Administration

Nickolas McLaughlinWorks in: SeattleSkills:C#JavaScriptSQL Server Administration

Zachary ParsonsWorks in: SeattleSkills:Active Directory Administration SharePoint AdministrationSQL Server Administration

Check answer Show solution Reset

Correct

Lesson 3: Understanding Predicate Logic

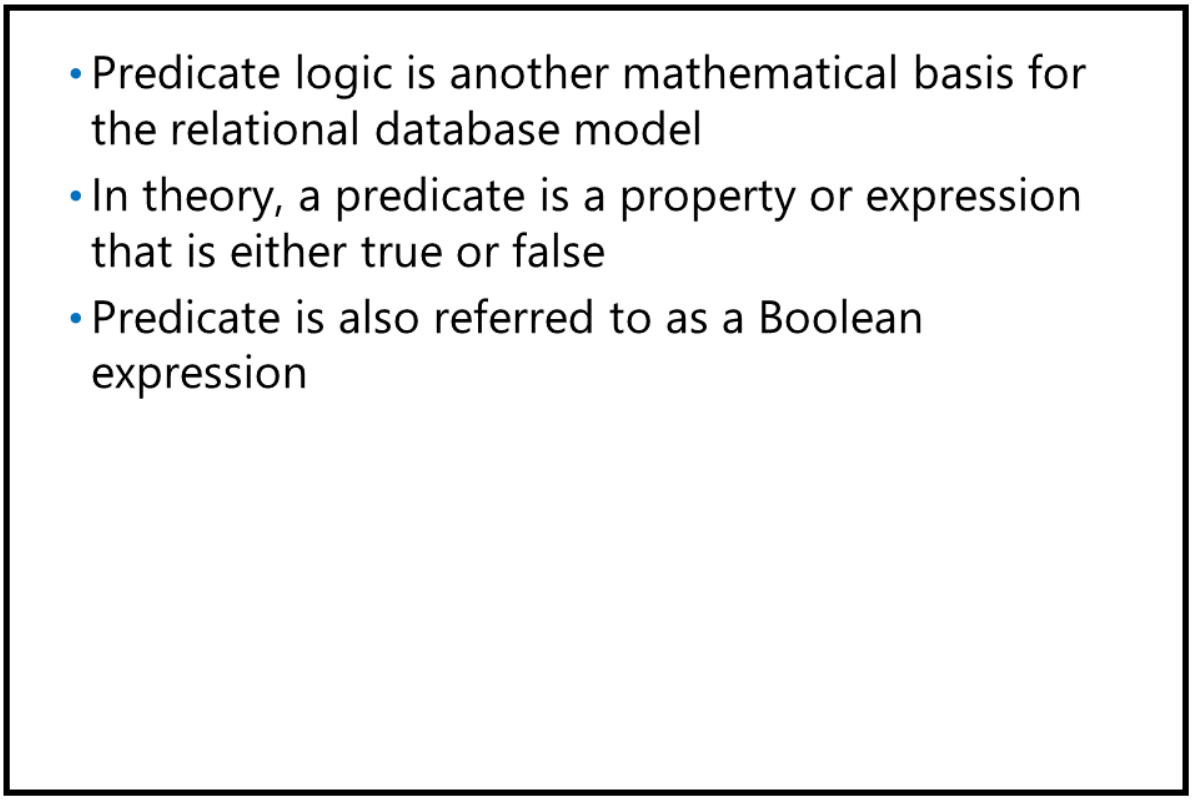
Along with set theory, predicate logic is a mathematical foundation for the relational database model, and with it, SQL Server. You probably have a fair amount of experience with predicate logic—rather than the set theory—even if you have never used the term to describe it. This lesson will introduce predicate logic and examine its application to querying SQL Server.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the role of predicate logic in a relational database.
- Understand the use of predicate logic on your T-SQL queries.

Predicate Logic and SQL Server



In theory, predicate logic is a framework for expressing logical tests that return true or false. A predicate is a property or expression that is true or false. You may have heard this referred to as a Boolean expression.

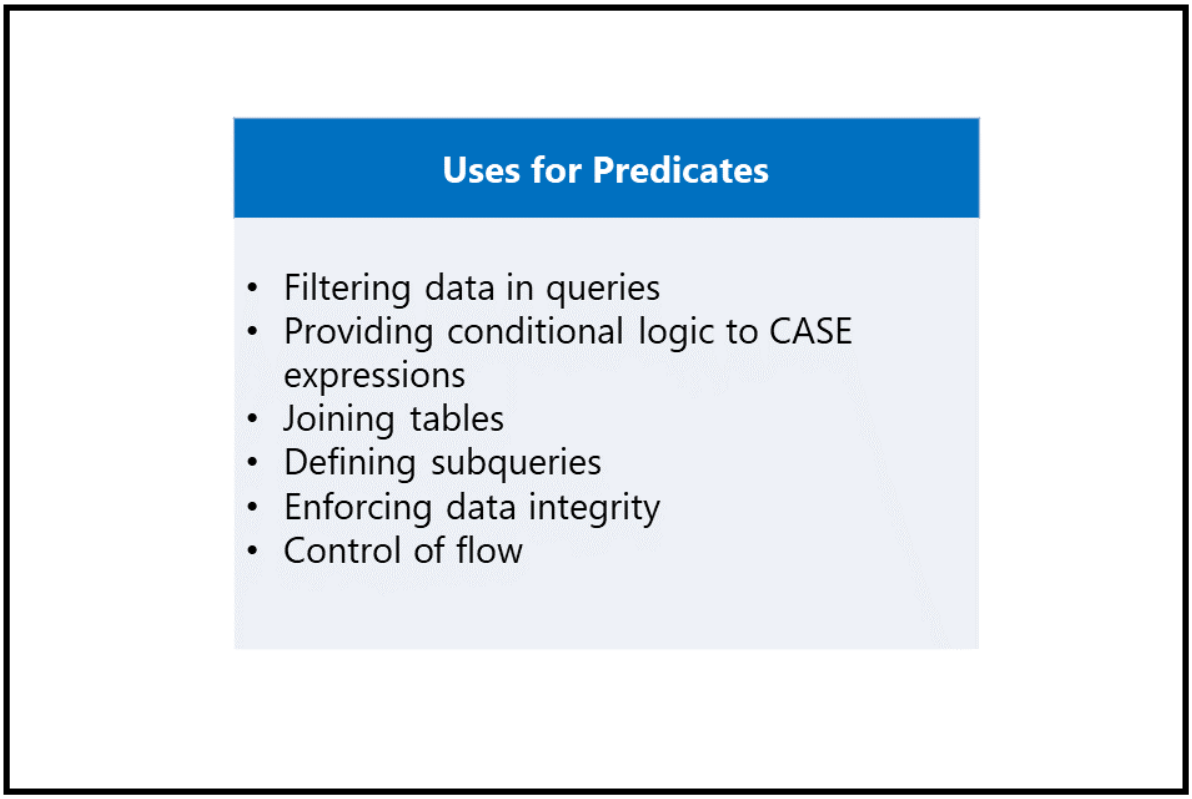
Taken by themselves, predicates make comparisons and express the results as true or false. However, in T-SQL, predicates don't stand alone. They are usually embedded in a statement that does something with the true or false

result, such as a WHERE clause to filter rows; a CASE expression to match a value; or even a column constraint governing the range of acceptable values for that column in a table's definition.

There's one important omission in the formal definition of a predicate—how to handle unknown, or missing, values. If a database is set up so that missing values are not permitted (through constraints, or default value assignments), then perhaps this is not an important omission. In most real-world environments, however, you need to account for missing or unknown values, and extend your understanding of predicates from two possible outcomes (true or false) to three—true, false, or unknown.

The use of NULLs as a mark for missing data will be discussed further in the next topic, and later in this course.

Predicate Logic Applied to SQL Server Queries



As you have been learning, the ability to use predicates to express comparisons in terms of true, false, or unknown, is vital to writing effective queries in SQL Server. Although they have been discussing them separately, predicates do not stand alone, syntactically speaking. Typically, you will use predicates in any of the following roles within your queries:

- Filtering data (in WHERE and HAVING clauses).
- Providing conditional logic to CASE expressions.
- Joining tables (in the ON filter).
- Defining subqueries (in EXISTS tests, for example).

Additionally, predicates have uses outside SELECT statements, such as in CHECK constraints to limit values permitted in a column, and in control-of-flow elements, such as an IF statement.

In mathematics, you only need to consider values that are present, so predicates can result only in true or false values (known in predicate logic as “the law of the excluded middle”). In databases, however, you will likely have to account for missing values; the interaction of T-SQL predicates with missing values results in an unknown. When you are designing query logic, ensure that you have accounted for all three possible outcomes—true, false, or unknown. You will learn how to use three-valued logic in WHERE clauses later in this course.

Check Your Knowledge

Select the best answer

From the following T-SQL elements, select the one that can include a predicate:

- WHERE clauses
- JOIN conditions
- HAVING clauses
- WHILE statements
- All of the above

Check answer Show solution Reset

WHERE clauses use predicates to determine which rows to return. JOIN conditions use predicates to determine which rows from different tables to join into a single result row. HAVING clauses use predicates to determine which groups to return. WHILE statements use predicates to determine when to stop executing T-SQL statements in a batch.

Lesson 4: Understanding the Logical Order of Operations in SELECT Statements

T-SQL is unusual as a programming language in one key aspect. The order in which you write a statement is not necessarily that in which the database engine will evaluate and process it. Database engines may optimize their execution of a query, providing the accuracy of the result (as determined by the logical order) is retained. As a result, unless you learn the logical order of operations, you may find both conceptual and practical obstacles to writing your queries. This lesson will introduce the elements of a SELECT statement; delineate the order in which the elements are evaluated; and then apply this understanding for a practical approach to writing queries.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the elements of a SELECT statement.
- Understand the order in which clauses in a SELECT statement are evaluated.
- Apply your understanding of the logical order of operations to writing SELECT statements.

Elements of a SELECT Statement

Element	Expression	Role
SELECT	<select list>	Defines which columns to return
FROM	<table source>	Defines table(s) to query
WHERE	<search condition>	Filters returned data using a predicate
GROUP BY	<group by list>	Arranges rows by groups
HAVING	<search condition>	Filters groups by a predicate
ORDER BY	<order by list>	Sorts the results

To understand the logical order of operations, you need to look at a SELECT statement as a whole, including a number of optional elements. However, this lesson is not designed to provide detailed information about these elements—each part of a SELECT statement will be discussed in subsequent modules. Understanding the details of a WHERE clause, for example, is not required to recognize its place in the sequence of events.

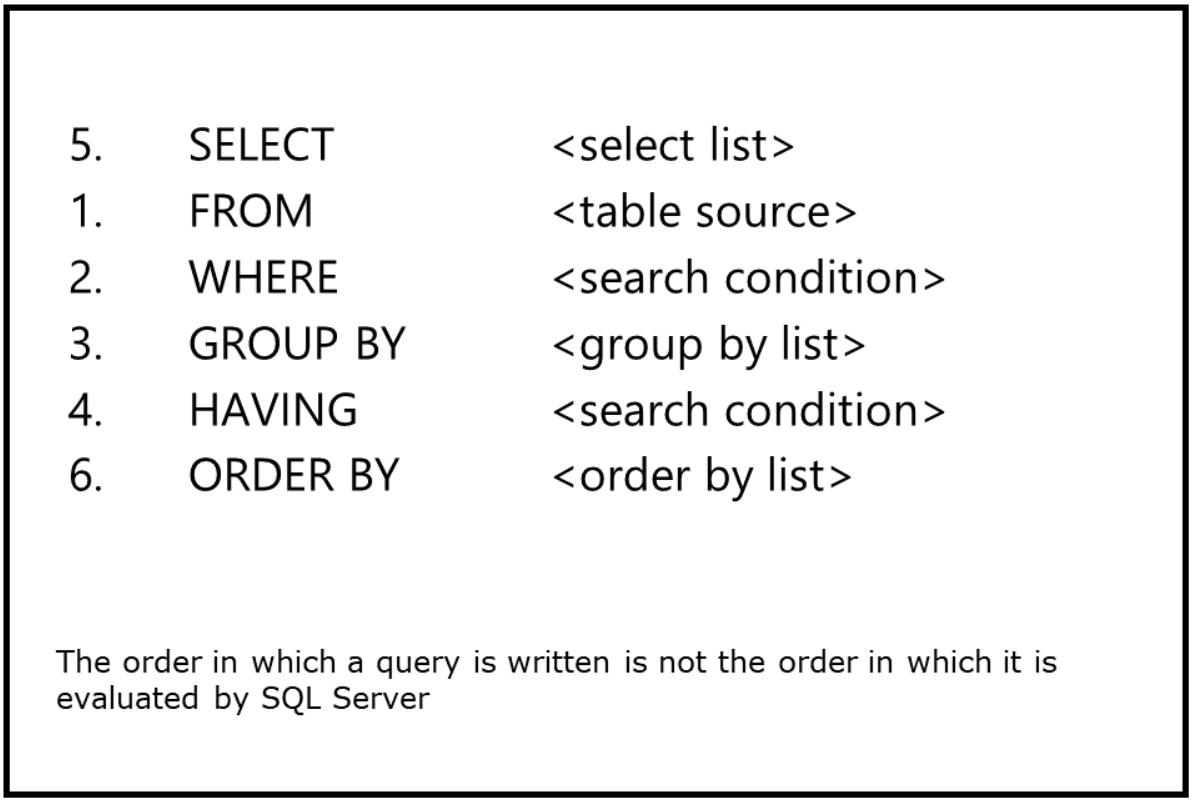
A SELECT statement is made up of mandatory and optional elements. Strictly speaking, SQL Server only requires a SELECT clause to execute without error. A SELECT clause without a FROM clause operates as if selecting from an imaginary table containing one row. You will see this behavior when you test variables later in this course. However, as a SELECT clause without a FROM clause cannot retrieve data from a table, you will treat stand-alone SELECT clauses as a special case not directly relevant to this lesson. Let's examine the elements, their high level role in a SELECT statement, and the order in which they are evaluated by SQL Server.

Not all elements will be present in every SELECT query. However, when an element is present, it will always be evaluated in the same order, with respect to the others present. For example, a WHERE clause will always be evaluated after the FROM clause and before a GROUP BY clause, if one exists.

You will discuss the order of these operations in the next topic.

> **Note:** For the purposes of this lesson, additional optional elements, such as DISTINCT, OVER, and TOP, are omitted. They will be introduced, and their order discussed, in later modules.

Logical Query Processing



The order in which a SELECT statement is written is not that in which it is evaluated and processed by the SQL Server Database Engine.

Logical Query Processing

```
USE TSQL;
SELECT EmployeeId, YEAR(OrderDate) AS OrderYear
FROM Sales.Orders
WHERE CustomerId = 71
GROUP BY EmployeeId, YEAR(OrderDate)
HAVING COUNT(*) > 1
ORDER BY EmployeeId, OrderYear;
```

Before you examine the run-time order of operations, let's briefly examine what the query does, although details on many clauses will need to wait until the appropriate module. The first line ensures you're connected to the correct database for the query. This line is not being examined for its run-time order.

Change the Database Connection

```
USE TSQL; -- change connection context to a database named TSQL.
```

The next line is the start of the SELECT statement as you wrote it, but as you'll see, it will not be the first line evaluated.

Start of SELECT

```
SELECT EmployeeId, YEAR(OrderDate) AS OrderYear
```

FROM Clause

```
FROM Sales.Orders
```

WHERE Clause

```
WHERE CustomerId = 71
```

GROUP BY Clause

```
GROUP BY EmployeeId, YEAR(OrderDate)
```

HAVING Clause

```
HAVING COUNT(*) > 1
```

ORDER BY Clause

```
ORDER BY EmployeeId, OrderYear;
```

Now that you've established what each clause does, let's look at the order in which SQL Server must evaluate them:

1. The FROM clause is evaluated first, to provide the source rows for the rest of the statement. Later in the course, you'll see how to join multiple tables together in a FROM clause. A virtual table is created and passed to the next step.
2. The WHERE clause is next to be evaluated, filtering those rows from the source table that match a predicate. The filtered virtual table is passed to the next step.
3. GROUP BY is next, organizing the rows in the virtual table according to unique values found in the GROUP BY list. A new virtual table is created, containing the list of groups, and is passed to the next step.

> **Note:** From this point in the flow of operations, only columns in the GROUP BY list or aggregate functions may be referenced by other elements. This will have a significant impact on the SELECT

list.

4. The HAVING clause is evaluated next, filtering out entire groups based on its predicate. The virtual table created in step 3 is filtered and passed to the next step.

5. The SELECT clause finally executes, determining which columns will appear in the query results.

> **Note:** Because the SELECT clause is evaluated after the other steps, any column aliases (in our example, OrderYear) created there cannot be used in the GROUP BY or HAVING clause.

6. In our example, the ORDER BY clause is the last to execute, sorting the rows as determined in its column list.

Logical Order

```
FROM Sales.Orders
WHERE CustomerId = 71
GROUP BY EmployeeId, YEAR(OrderDate)
HAVING COUNT(*) > 1
SELECT EmployeeId, YEAR(OrderDate) AS OrderYear
ORDER BY EmployeeId, OrderYear;
```

As you have seen, you do not write T-SQL queries in the same order in which they are logically evaluated. Because the run-time order of evaluation determines what data is available to clauses downstream from one another, it's important to understand the true logical order when writing queries.

Applying the Logical Order of Operations to Writing SELECT Statements

```
USE TSQL;

SELECT EmployeeId, YEAR(OrderDate) AS OrderYear
FROM Sales.Orders
WHERE CustomerId = 71
GROUP BY EmployeeId, YEAR(OrderDate)
HAVING COUNT(*) > 1
ORDER BY EmployeeId, OrderYear;
```

Now that you have learned the logical order of operations when a SELECT query is evaluated and processed, remember the following considerations when writing a query. Note that some of these may refer to details you will learn in subsequent modules:

- Decide which tables to query first, in addition to any table aliases you will apply. This will determine the FROM clause.
- Decide which set or subset of rows will be retrieved from the table(s) in the FROM clause, and how you will express your predicate. This will determine your WHERE clause.
- If you intend to group rows, decide which columns will be grouped. Remember that only columns in the GROUP BY clause, in addition to aggregate functions such as COUNT, may ultimately be included in the SELECT clause.
- If you need to filter out groups, decide on your predicate and build a HAVING clause. The results of this phase become the input to the SELECT clause.
- If you are not using GROUP BY, determine which columns from the source table(s) you wish to display, and use any table aliases you created to refer to them. This will become the core of your SELECT clause. If you have used a GROUP BY clause, select from the columns in the GROUP BY clause, and add any additional aggregates to the SELECT list.
- Finally, remember that sets do not include any ordering—you will need to add an ORDER BY clause to guarantee a sort order if required.

Demonstration: Logical Query Processing

In this demonstration, you will see how to view query output that illustrates logical processing order.

Demonstration Steps

View Query Output That Illustrates Logical Processing Order

1. Start the **MT17B-WS2016-NAT**, **20761C-MIA-DC**, and **20761C-MIA-SQL** virtual machines, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Open SQL Server Management Studio.
3. In the **Connect to Server** dialog box, in the **Server name** box, enter the server you created during preparation. For example, **20761Ca-azure.database.windows.net**.
4. In the **Authentication** list, click **SQL Server Authentication**.
5. In the **Login** box, type **Student**.
6. In the **Password** box, type **Pa55w.rd**, and then click **Connect**.
7. On the **File** menu, point to **Open**, and then click **Project/Solution**.
8. In the **Open Project** dialog box, browse to the **D:\Demofiles\Mod02\Demo** folder, and then double-click **Demo.ssmssln**.
9. In Solution Explorer, double-click the **21 - Demonstration B.sql** script file.
10. On the **Query** menu, point to **Connection**, and then click **Change Connection**.
11. In the **Connect to Database Engine** dialog box, in the **Server name** box, enter the server you created during preparation. For example, **20761Ca-azure.database.windows.net**.
12. In the **Authentication** list, click **SQL Server Authentication**.
13. In the **Login** box, type **Student**.
14. In the **Password** box, type **Pa55w.rd**, and then click **Connect**.
15. In the **Available Databases** list, click **AdventureWorksLT**.
16. Select the code under the comment **Step 1**, and then click **Execute**.
17. Select the code under **Step 2**, and then click **Execute**.
18. Select the code under **Step 3**, and then click **Execute**.
19. Select the code under **Step 4**, and then click **Execute**.
20. Select the code under **Step 5**, and then click **Execute**. Note the error message.
21. Select the code under **Step 6**, and then click **Execute**.
22. Select the code under **Step 7**, and then click **Execute**.
23. Select the code under **Step 8**, and then click **Execute**.
24. Close SQL Server Management Studio, without saving any changes.

Check Your Knowledge

Sequencing Activity

Put the following T-SQL elements in order by numbering each to indicate the order that SQL Server will process them in when they appear in a single SELECT statement.

FROM

WHERE

GROUP BY

HAVING

SELECT

ORDER BY

Check answer Show solution Reset

Correct

Lab: Introduction to T-SQL Querying

Scenario

You are an Adventure Works business analyst, who will be writing reports against corporate databases stored in SQL Server. To help you become more comfortable with SQL Server querying, the Adventure Works IT department has provided some common queries to run against their databases. You will review and execute these queries.

Objectives

After completing this lab, you will be able to:

- Execute basic SELECT statements.
- Execute queries that filter data.
- Execute queries that sort data.

Lab Setup

Estimated Time: 30 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Executing Basic SELECT Statements

Scenario

The T-SQL script provided by the IT department includes a SELECT statement that retrieves all rows from the HR.Employees table—this includes the firstname, lastname, city, and country columns. You will execute the T-SQL script against the TSQL database.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Execute the T-SQL Script
3. Execute a Part of the T-SQL Script

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\Lab02\Starter** folder as Administrator.

Detailed Steps ▲

Task 2: Execute the T-SQL Script

1. Open the project file **D:\Labfiles\Lab02\Starter\Project\Project.ssmssln**.
2. Connect to the **MIA-SQL** database using Windows authentication.
3. Open the T-SQL script **51 - Lab Exercise 1.sql**.
4. Execute the whole script.
5. Observe the result and the database context.
6. Which database is selected in the Available Databases box?

Detailed Steps ▲

Task 3: Execute a Part of the T-SQL Script

1. Highlight the SELECT statement in the T-SQL script under the **Task 2** description and click **Execute**.
2. Observe the result. You should get the same result as in **Task 2**.

> **Note:** One way to highlight a portion of code is to hold down the Alt key while drawing a rectangle around it with your mouse. The code inside the drawn rectangle will be selected. Try it.

3. Close all open windows.

Result: After this exercise, you should know how to open the T-SQL script and execute the whole script or just a specific statement inside it.

Exercise 2: Executing Queries That Filter Data Using Predicates

Scenario

The next T-SQL script is very similar to the first one. The SELECT statement retrieves the same columns from the HR.Employees table, but uses a predicate in the WHERE clause to retrieve only rows with the value “USA” in the country column.

The main tasks for this exercise are as follows:

1. Execute the T-SQL Script
2. Change the Database Context with the GUI
3. Change the Database Context with T-SQL

Detailed Steps ▲

Task 1: Execute the T-SQL Script

1. Open the project file **D:\Labfiles\Lab02\Starter\Project\Project.ssmssln** and the T-SQL script **61 - Lab Exercise 2.sql**. Execute the whole script.
2. There is an error. What is the error message? Why do you think this happened?

Detailed Steps ▲

Task 2: Change the Database Context with the GUI

1. Apply the needed changes to the script so that it will run without an error. (Hint: you do not need to change any T-SQL information to fix the error.) Test the changes by executing the whole script.
2. Observe the result. Notice that the result has fewer rows than the result in exercise 1, task 2.

Detailed Steps ▲

Task 3: Change the Database Context with T-SQL

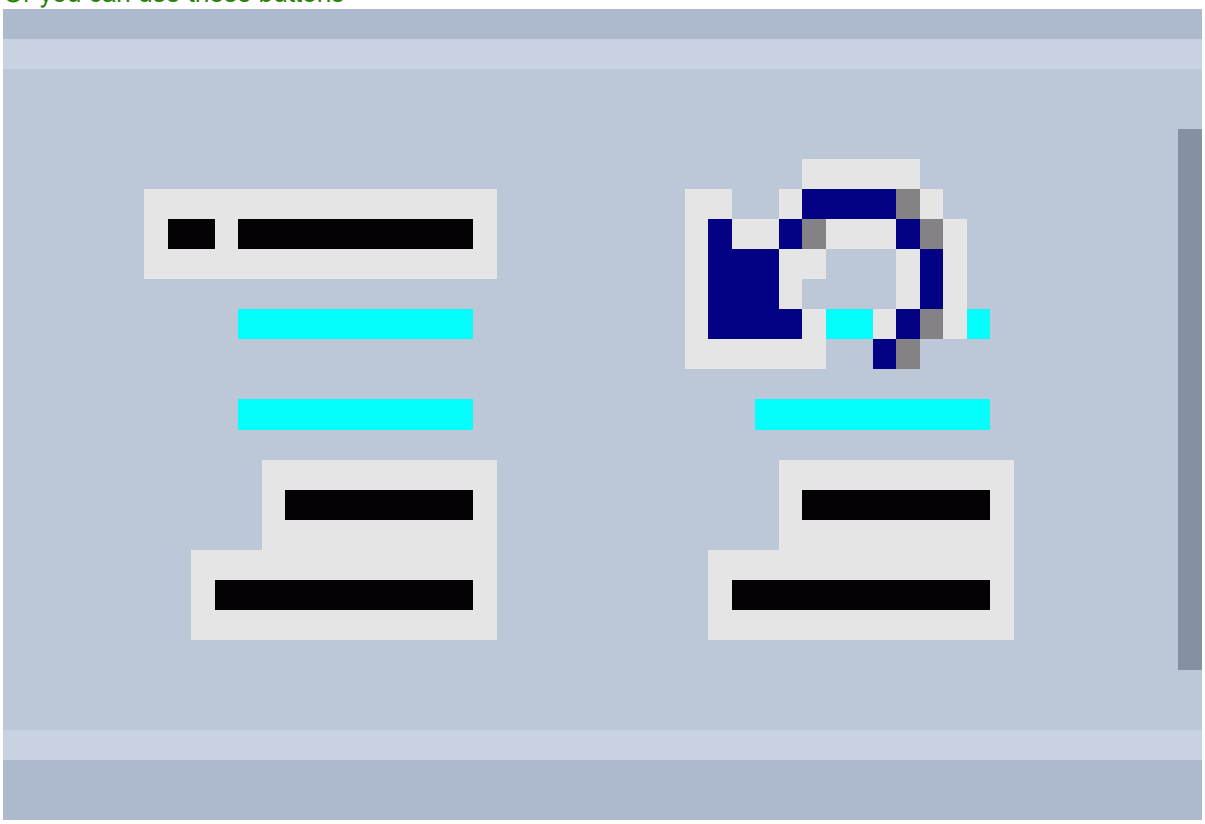
1. Comments in T-SQL scripts can be written inside the line by specifying --. The text after the two hyphens will be ignored by SQL Server. You can also specify a comment as a block starting with **/*** and ending with ***/**. The text in between is treated as a block comment and is ignored by SQL Server.
2. Uncomment the following statements:

```
USE TSQL;
GO
```

3. Save and close the T-SQL script. Re-open the T-SQL script **61 - Lab Exercise 2.sql**. Execute the whole script.
4. Why did the script execute with no errors?
5. Observe the result and notice the database context in the Available Databases box.

Note: SSMS supplies keyboard shortcuts and two buttons so you can quickly comment and uncomment code.
The keyboard shortcuts are CTRL+K then CTRL+C to comment, and CTRL+K then CTRL+U to uncomment.

Or you can use these buttons



on the toolbar.

Result: After this exercise, you should have a basic understanding of database context and how to change it.

Exercise 3: Executing Queries That Sort Data Using ORDER BY

Scenario

The last T-SQL script provided by the IT department has a comment: “This SELECT statement returns first name, last name, city, and country/region information for all employees from the USA, ordered by last name.”

The main tasks for this exercise are as follows:

1. Execute the Initial T-SQL Script
2. Uncomment the Needed T-SQL Statements and Execute Them

Detailed Steps ▲

Task 1: Execute the Initial T-SQL Script

1. Open the T-SQL script **71 - Lab Exercise 3.sql**, and execute the whole script.

2. Observe the results. Why is the result window empty?

Detailed Steps ▲

Task 2: Uncomment the Needed T-SQL Statements and Execute Them

1. Observe that, before the USE statement, there are the characters -- which means that the USE statement is treated as a comment. There is also a block comment around the whole T-SQL SELECT statement. Uncomment both statements.
2. First, execute the USE statement, and then execute the SELECT clause.
3. Observe the results. Notice that the results have the same rows as in exercise 1, task 2, but they are sorted by the lastname column.

Note: What changes would you make to change the sort order to descending?

Result: After this exercise, you should have an understanding of how comments can be specified inside T-SQL scripts. You will also have an appreciation of how to order the results of a query.

Module Review and Takeaways

In this module, you have learned how to describe:

- The elements of T-SQL and their role in writing queries.
- The use of sets in SQL Server.
- The use of predicate logic in SQL Server.
- The logical order of operations in SELECT statements.

Review Question(s)

Check Your Knowledge

Discovery

Which category of T-SQL statements concerns querying and modifying data?

Show solution Reset

Data Manipulation Language (DML).

Check Your Knowledge

Discovery

What are some examples of aggregate functions supported by T-SQL?

Show solution Reset

SUM, MIN, COUNT, COUNTBIG, MAX, AVG.
Aggregate functions are used in conjunction with a GROUP BY clause.

Check Your Knowledge

Discovery

Which SELECT statement element will be processed before a WHERE clause?

Show solution Reset

FROM. The FROM clause is evaluated first to provide the source rows for the rest of the statement.