Module 6: Working with SQL Server Data Types

Contents:

Module Overview

To write effective queries in T-SQL, you should understand how SQL Server® 2016 stores different types of data. This is especially important if your queries not only retrieve data from tables, but also perform comparisons, manipulate data, and implement other operations.

In this module, you will learn about the data types SQL Server uses to store data. In the first lesson, you will be introduced to many numeric and special-use data types. You will learn about conversions between data types and the importance of data type precedence. You will learn how to work with character-based data types, including functions that can be used to manipulate the data. You will also learn how to work with temporal data, or data and time data, including functions to retrieve and manipulate all or portions of a stored date.

Objectives

After completing this module, you will be able to:

- Describe SQL Server data types, type precedence, and type conversions.
- Write queries using numeric data types.
- Write queries using character data types.
- Write queries using date and time data types.

Lesson 1: Introducing SQL Server Data Types

In this lesson, you will explore many of the data types SQL Server uses to store data, and learn how data is converted between data types.

Note: Character, date, and time data types are excluded from this lesson but will be covered later in the module.

If your focus in taking this course is to write queries for reports, you might wish to note which data types are used in your environment. You can then plan your reports and client applications with sufficient capacity to display the range of values held by the SQL Server data types. You may also need to plan for data type conversions in your queries to display SQL Server data in other environments.

If your focus is to continue into database development and administration, you might wish to note the similarities and differences within categories of data types, and plan your storage accordingly, as you create types and design parameters for stored procedures.

Lesson Objectives

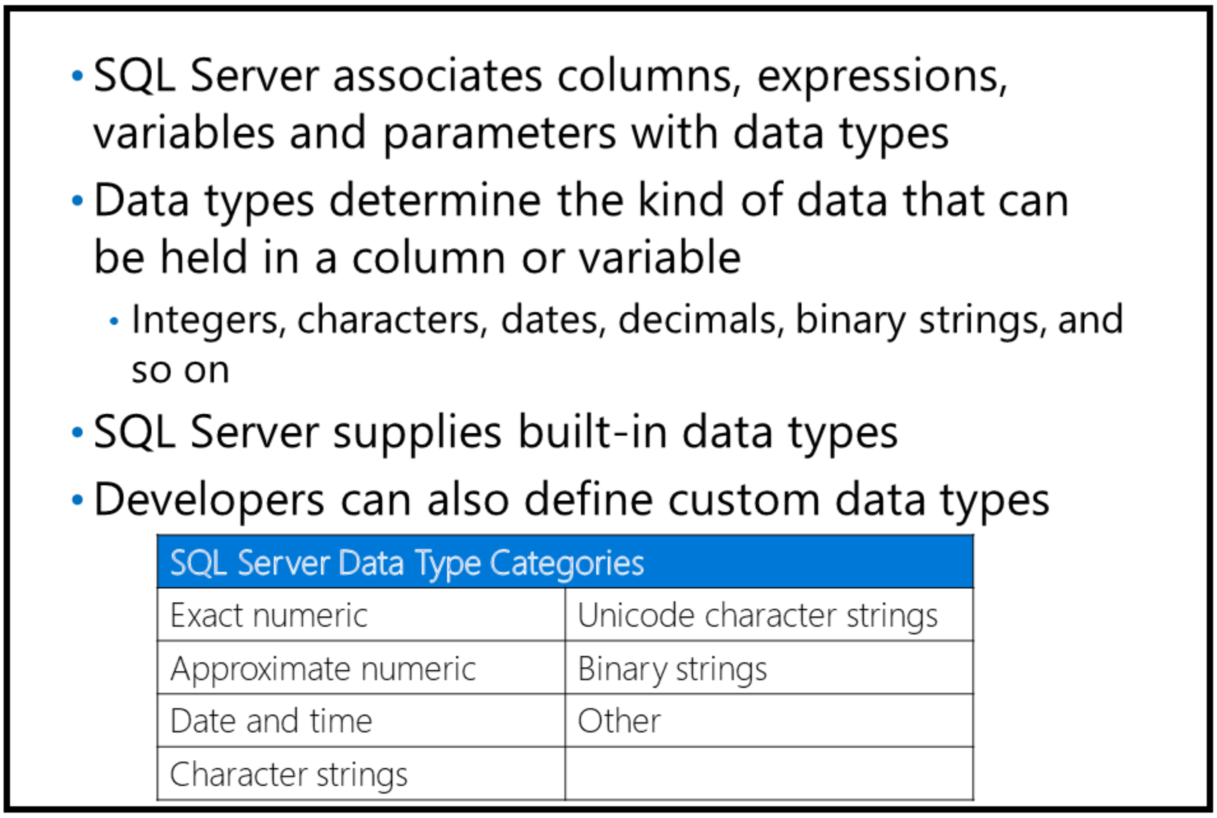
After completing this lesson, you will be able to:

- Describe how SQL Server uses data types.
- Describe the attributes of numeric data types, as well as binary strings and other specialized data types.

- Describe data type precedence and its use in converting data between different data types.

- Describe the difference between implicit and explicit data type conversion.

SQL Server Data Types



- SQL Server associates columns, expressions, variables and parameters with data types
- Data types determine the kind of data that can be held in a column or variable
  - Integers, characters, dates, decimals, binary strings, and so on
- SQL Server supplies built-in data types
- Developers can also define custom data types

SQL Server Data Type Categories	
Exact numeric	Unicode character strings
Approximate numeric	Binary strings
Date and time	Other
Character strings	

SQL Server 2016 defines a set of system data types for storing data in columns, holding values temporarily in variables, operating on data in expressions, and passing parameters to stored procedures.

Data types specify the type, length, precision, and scale of data. Understanding the basic types of data in SQL Server is fundamental to writing queries in T-SQL, along with designing tables and creating other objects.

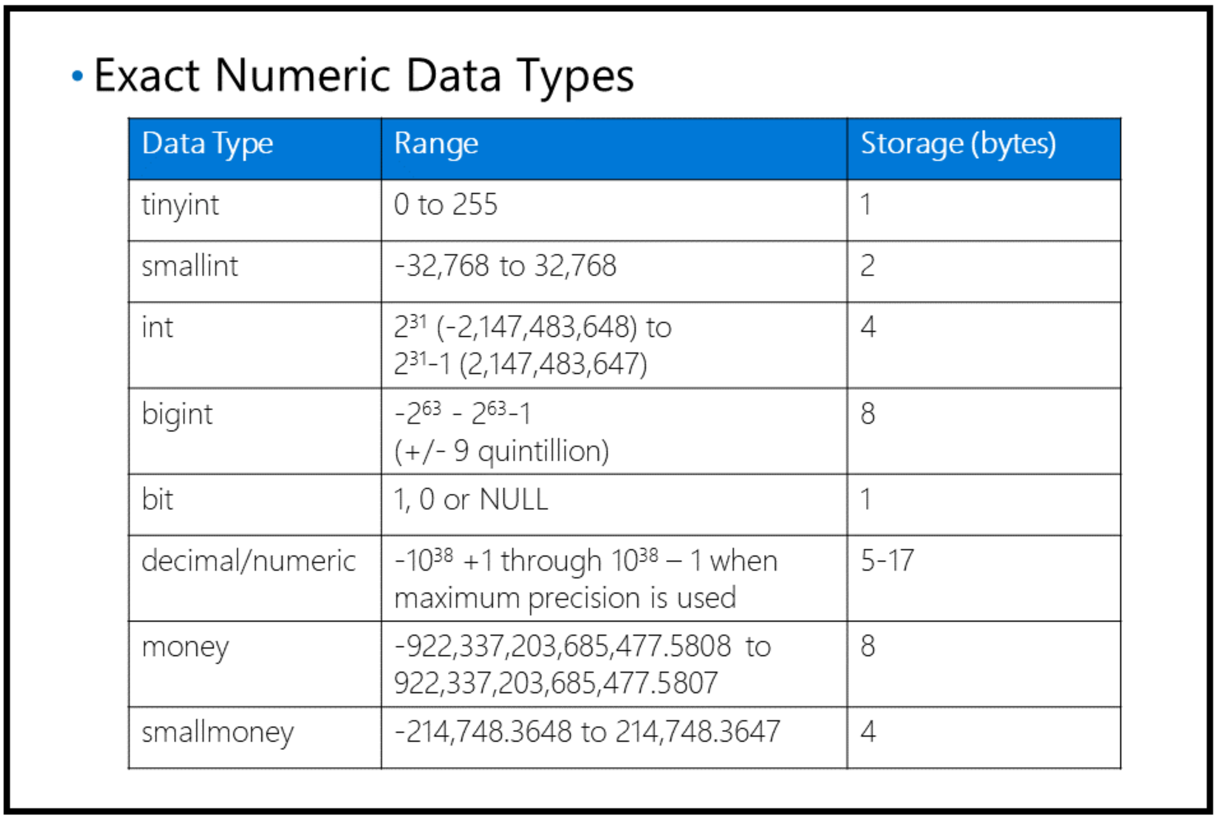
Developers might also extend the supplied set by creating aliases to built-in types and even by producing new user-defined types using the Microsoft® .NET Framework; however, this lesson will focus on the built-in system data types.

Other than character, date, and time types, which will be covered later in this module, SQL Server data types can be grouped into the following categories:

- **Exact numeric**. These data types store data with precision, either as:
  - o Integers—whole numbers with varying degrees of capacity.
  - o Decimals—decimal numbers with control over both the total number of digits stored and the number of digits to the right of the decimal place.
- **Approximate numeric**. These data types allow inexact values to be stored, typically for use in scientific calculations.
- **Binary strings**. These data types allow binary data to be stored, such as byte streams or hashes, to support custom applications.
- **Other data types**. This catch-all category includes several special types that fall outside the other categories. Some of these data types can be used as column data types (and are therefore accessible to queries). This category also includes data types not used for storage, but rather for special operations, such as cursor manipulation or creating table variables for further processing. If you are a report writer, you may only encounter the data types used for columns, such as the **uniqueidentifier** and **xml** data types.

As you learn about these types, take note of the relationship between capacity and storage requirements.

Numeric Data Types



- Exact Numeric Data Types

Data Type	Range	Storage (bytes)
tinyint	0 to 255	1
smallint	-32,768 to 32,768	2
int	2^{31} (-2,147,483,648) to 2^{31}-1 (2,147,483,647)	4
bigint	-2^{63} - 2^{63}-1 (+/- 9 quintillion)	8
bit	1, 0 or NULL	1
decimal/numeric	-10^{38} +1 through 10^{38} – 1 when maximum precision is used	5-17
money	-922,337,203,685,477.5808 to 922,337,203,685,477.5807	8
smallmoney	-214,748.3648 to 214,748.3647	4

- Numeric data types fall into one of two subcategories—exact numeric and approximate numeric.
- Exact numeric data types:
  - o Integer data types. The distinction between the integer data types (**tinyint**, **smallint**, **int**, **bigint**) relates to their capacity and storage requirements. The **tinyint** data type, for example, holds values from 0 to 255 with a storage cost of 1 byte. By contrast, the **bigint** data type holds values from -263 (-9,223,372,036,854,775,808) to 263-1 (9,223,372,036,854,775,807) with a storage cost of 8 bytes.
  - o Decimal data types. These data types are specified with the total number of digits to be stored (precision) and the number of digits to the right of the decimal place (scale). The larger the precision, the greater the storage cost. Note that there is no functional difference between the **decimal** data type and **numeric** data type—**decimal** is the ISO standards-compliant name for the data type; **numeric** is used for backward compatibility with earlier versions of SQL Server.
  - o Money data types, for storing monetary or currency values with a scale of up to four decimal places. As with the integer types, the distinction between the money data types **money** and **smallmoney** relates to their capacity and

storage requirements. The **smallmoney** data type holds values from -214,748.3648 to 214,748.3647 with a storage cost of 4 bytes. The **money** data type holds values from -922,337,203,685,477.5808 to 922,337,203,685,477.5807 with a storage cost of 8 bytes.

- Boolean data type. The **bit** data type is used to store Boolean values (true/false) which are treated by SQL Server as numeric values—1 for true and 0 for false.

For more information, see the following topics in Microsoft Docs:

Data Types (Transact-SQL)

http://aka.ms/we8bzv

Precision, Scale, and Length (Transact-SQL)

http://aka.ms/t0hwx5

decimal and numeric (Transact-SQL)

http://aka.ms/sqkh78

- Approximate numeric data types. The approximate numeric data types are less accurate, but have more capacity than the exact numeric data types. The approximate numeric data types store values in scientific notation which, because of a lack of precision, loses accuracy.
  - The **float** data type takes an optional parameter of the number of bits used to store the mantissa of the **float** number in scientific notation. The size of the mantissa value determines the storage size of the float. If the mantissa is in the range 1 to 24, the float requires 4 bytes. If the mantissa is between 25 and 53, it requires 8 bytes.
  - The **real** data type is a synonym for a **float** data type with a mantissa value of 24 (that is, **float(24)**).

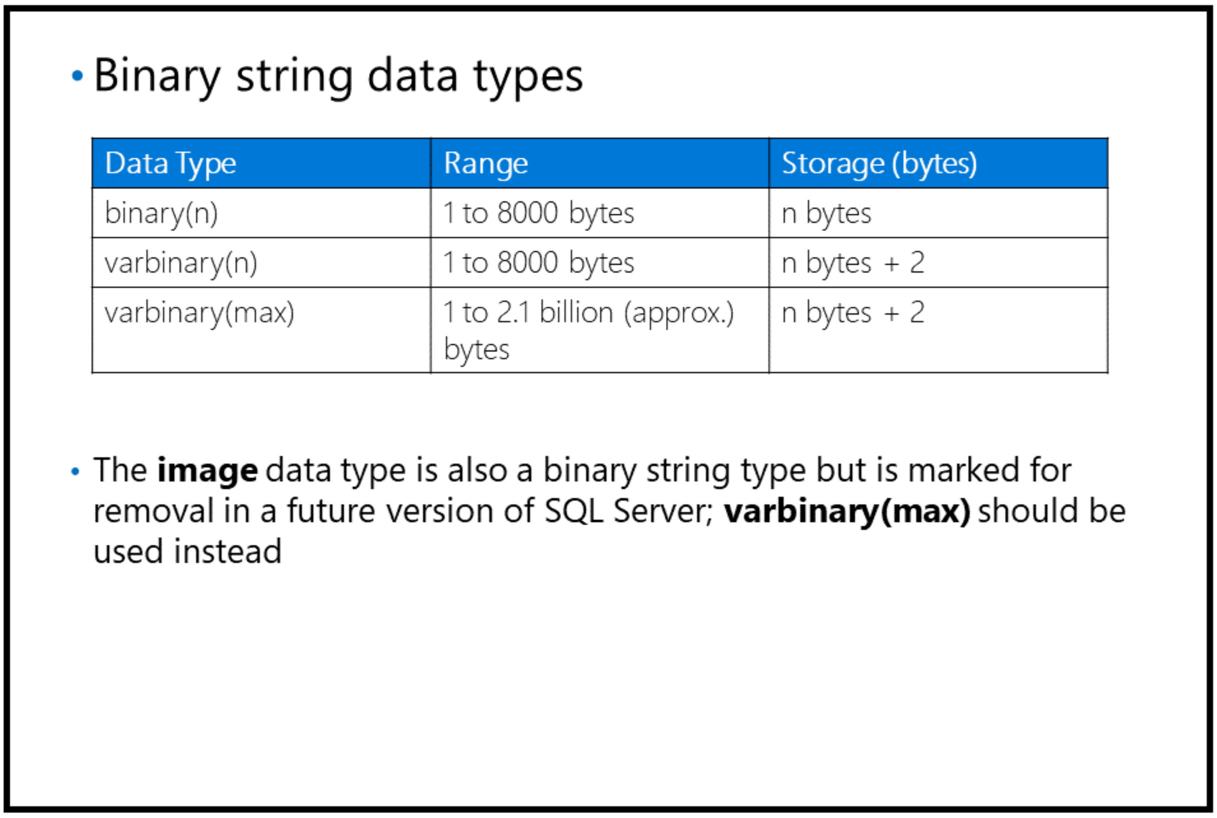
Note: Note that, in this context, the term *mantissa* is used to mean the significant digits of the floating point number. In mathematics, this portion of the number is more commonly referred to as the *significand;* however, in computer science, it is commonly referred to as the *mantissa*.

See the topic *float and real (Transact-SQL)* in Microsoft Docs:

float and real (Transact-SQL)

http://aka.ms/noqiea

Binary String Data Types



- Binary string data types

Data Type	Range	Storage (bytes)
binary(n)	1 to 8000 bytes	n bytes
varbinary(n)	1 to 8000 bytes	n bytes + 2
varbinary(max)	1 to 2.1 billion (approx.) bytes	n bytes + 2

- The **image** data type is also a binary string type but is marked for removal in a future version of SQL Server; **varbinary(max)** should be used instead

Binary string data types allow a developer to store binary information, such as serialized files, images, byte streams, and other specialized data. If you are considering using the binary data type, note the differences in range and storage requirements, compared with numeric and character string data. You can choose between fixed-width and variable-width binary strings; the differences between these will be explained in the character data type lesson later in the module.

Converting to Binary Data Type

```
SELECT CAST(12345 AS binary(4)) AS Result;
```

Returns the following:

```
Result
----------
0x00003039
```

The two leading characters in the output (0x) indicate that the output is a binary string.

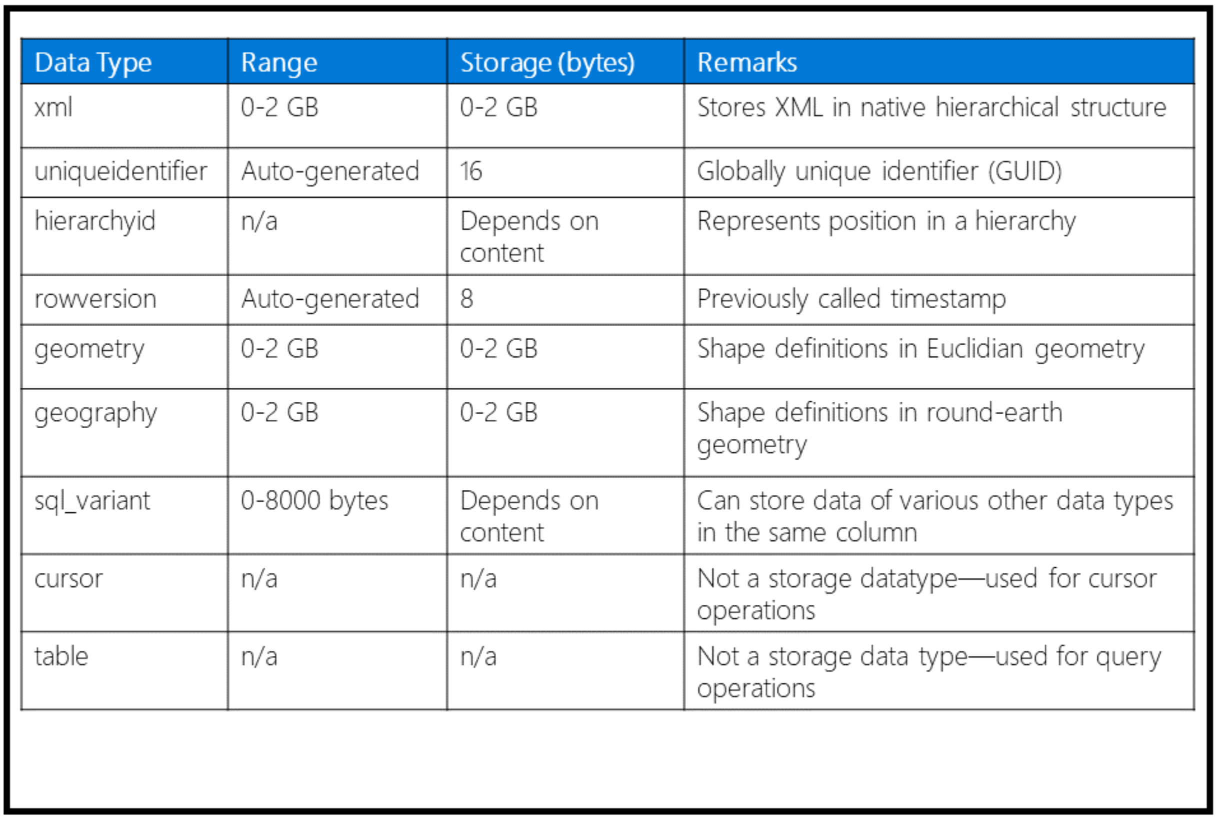
For more information, see the *binary and varbinary (Transact-SQL)* topic in the SQL Server Technical Documentation:

binary and varbinary (Transact-SQL)

http://aka.ms/o0ap4l

Note: The **image** data type is also a binary string type but is marked for removal in a future version of SQL Server. **varbinary(max)** should be used instead.

Other Data Types



Data Type	Range	Storage (bytes)	Remarks
xml	0-2 GB	0-2 GB	Stores XML in native hierarchical structure
uniqueidentifier	Auto-generated	16	Globally unique identifier (GUID)
hierarchyid	n/a	Depends on content	Represents position in a hierarchy
rowversion	Auto-generated	8	Previously called timestamp
geometry	0-2 GB	0-2 GB	Shape definitions in Euclidian geometry
geography	0-2 GB	0-2 GB	Shape definitions in round-earth geometry
sql_variant	0-8000 bytes	Depends on content	Can store data of various other data types in the same column
cursor	n/a	n/a	Not a storage datatype—used for cursor operations
table	n/a	n/a	Not a storage data type—used for query operations

In addition to numeric and binary types, SQL Server also supplies some other data types for specialized use cases, such as storage and processing of XML, generation and storage of globally unique identifiers (GUIDs), the representation of hierarchies, and more:

- The **xml** data type allows the storage and manipulation of Extensible Markup Language data (XML). The advantage of the **xml** data type over storing XML in a character data type is that the **xml** data type allows XML nodes and attributes to be queried within a T-SQL query using XQuery expressions. The **xml** data type also optionally allows an XML schema to be enforced. Each instance of an **xml** data type can store up to 2 GB of data.

Additional Reading: See course 20472-2: *Developing Microsoft SQL Server Databases* for additional information on the XML data type.

- The **uniqueidentifier** data type allows the generation and storage of globally unique identifiers (GUIDs), stored as a 16-byte value. Values for the **uniqueidentifier** data type can be generated within SQL Server by using the NEWID() system function; they can also be generated by external applications or converted from string values.

Creating GUIDs for the uniqueidentifier Data Type

```
SELECT NEWID() AS GUID_from_NEWID, CAST('1C0E3B5C-EA7A-41DC-8E1C-
D0A302B5E58B' AS uniqueidentifier) AS GUID_cast_from_string;
```

Returns:

```
GUID_from_NEWID                       GUID_cast_from_string
------------------------------------  ----------------------------
----
DB71DBAE-460B-41DD-8CF1-FBEE3000BE0D  1C0E3B5C-EA7A-41DC-8E1C-
                                      D0A302B5E58B
```

- The **hierarchyid** data type is used to simplify the recording and querying of hierarchical relationships between rows in the same table—for example, the levels in an organizational chart or a bill of materials. SQL Server stores **hierarchyid** as a variable-length binary data type; the hierarchy is exposed through built-in functions.

Additional Reading: See course 20472-2: *Developing Microsoft SQL Server Databases* for additional information on the **hierarchyid** data type.

- The **rowversion** data type stores an automatically generated 8-byte binary value in a table that increments each time a row is inserted or updated. **Rowversion** values do not store date or time information, but can be used to detect whether a row has been changed since it was last read by a client application (for instance, when implementing optimistic locking).
- The spatial data types are special complex data types for dealing with geometric and geographic data. A detailed discussion of these types is beyond the scope of this course:
    - The **geometry** data type is used to store data in a Euclidean (flat) coordinate system. Arrays of coordinates defining lines, polygons and other simple

geometric shapes can be stored in the **geometry** data type. Special built-in methods are available for carrying out operations on **geometry** data.

- The **geography** data type is used to store data in a round-earth coordinate system, such as GPS latitude and longitude coordinated. As with the **geography** data type, shape definitions can be stored in the **geography** type, then built-in methods used to operate on **geography** data.

- The **sql_variant** type is a special type that may be used to store data of any other built-in data type—for instance, enabling integer, decimal and character data to be stored in the same column. Use of the **sql_variant** data type is not a best practice for typical database designs, and its use may indicate design problems. The **sql_variant** data type is listed here for completeness.

The following data types may not be used as data types for columns in tables or views; they are used as variables or parameters for stored procedures:

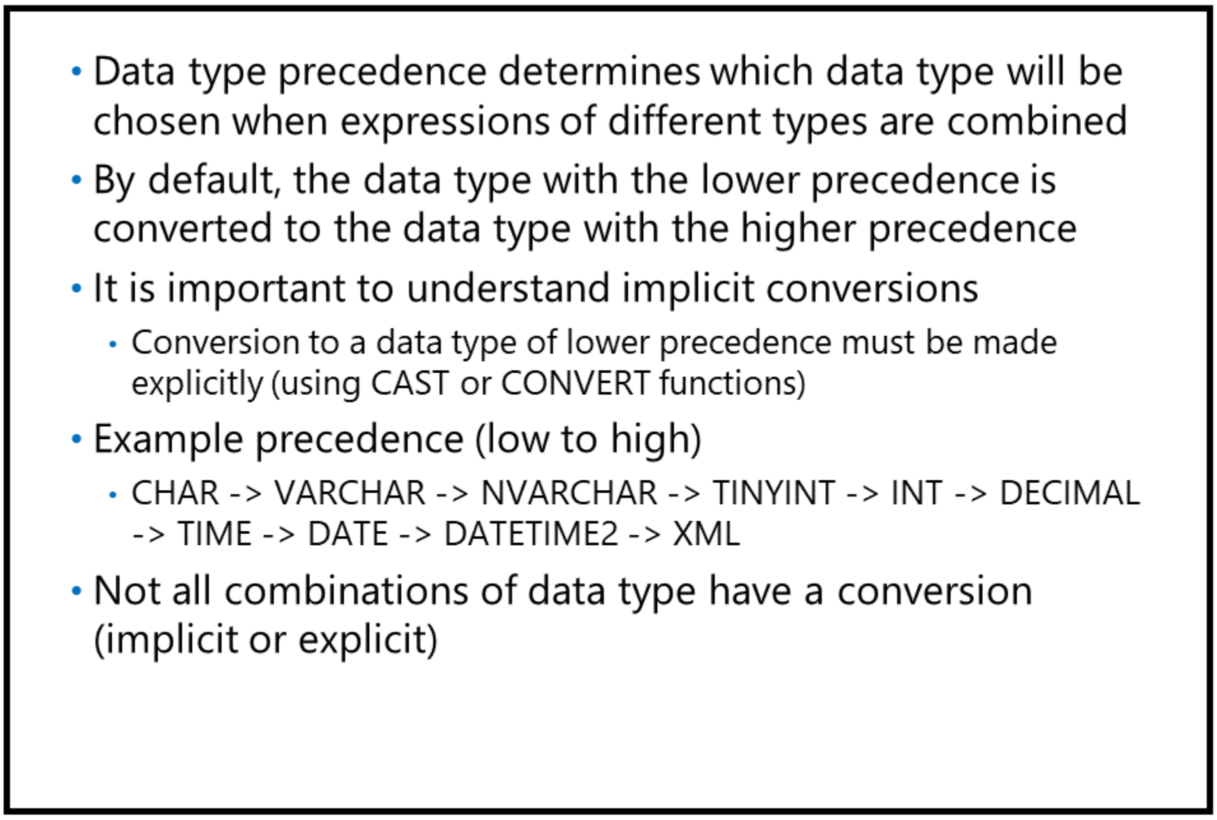
- The **cursor** data type is used to reference a cursor object, which allows row-by-row processing of a data set. A discussion of cursors is beyond the scope of this module.
- The **table** data type is used to define a table variable or stored procedure parameter, which has many of the properties of a standard database table but exists only in the context of the session for which it was created. Table data types are typically used to temporarily store the results of T-SQL statements for further processing later. You will learn about uses for the **table** data type later in this course.

For information on all of SQL Server's data types, see Microsoft Docs, starting from:

Data Types (Transact-SQL)

http://aka.ms/we8bzv

Data Type Precedence



- Data type precedence determines which data type will be chosen when expressions of different types are combined
- By default, the data type with the lower precedence is converted to the data type with the higher precedence
- It is important to understand implicit conversions
  - Conversion to a data type of lower precedence must be made explicitly (using CAST or CONVERT functions)
- Example precedence (low to high)
  - CHAR -> VARCHAR -> NVARCHAR -> TINYINT -> INT -> DECIMAL -> TIME -> DATE -> DATETIME2 -> XML
- Not all combinations of data type have a conversion (implicit or explicit)

When combining or comparing different data types in your queries, such as in a WHERE or JOIN clause, SQL Server will need to convert one value from its data type to that of the other value. Which data type is converted depends on the precedence between the two.

SQL Server defines a ranking of all its data types by precedence—between any two data types, one will have a lower precedence and the other a higher precedence. When converting, SQL Server will attempt to convert the lower data type to the higher one. Typically, this will happen implicitly, without the need for special code. However, it is important for you to have a basic understanding of this precedence arrangement so you know when you need to manually, or explicitly, convert data types to combine or convert them.

For example, here is a partial list of data types, ranked according to their precedence:

1. xml
2. datetime2
3. date
4. time

5. decimal
6. int
7. tinyint
8. nvarchar
9. char

When combining or comparing two expressions with different data types, the one lower on this list will be converted to the type that is higher. In this example, the variable of data type **tinyint** will be implicitly converted to **int** before being added to the **int** variable @myInt:

```
DECLARE @myTinyInt AS TINYINT = 25;
DECLARE @myInt as INT = 9999;
SELECT @myTinyInt + @myInt;
```

Note: Any implicit conversion is transparent to the user; therefore, if it fails (such as when your operation requires converting between data types for which no implicit conversion exists), you will need to explicitly convert the data type.
You will learn how to use the CAST and CONVERT functions for this purpose in the next module.
There are some combinations of data types for which no conversion, explicit or implicit, is possible.

For more information and a complete list of data types and a list of precedence, see Microsoft Docs:

Data Type Precedence (Transact-SQL)

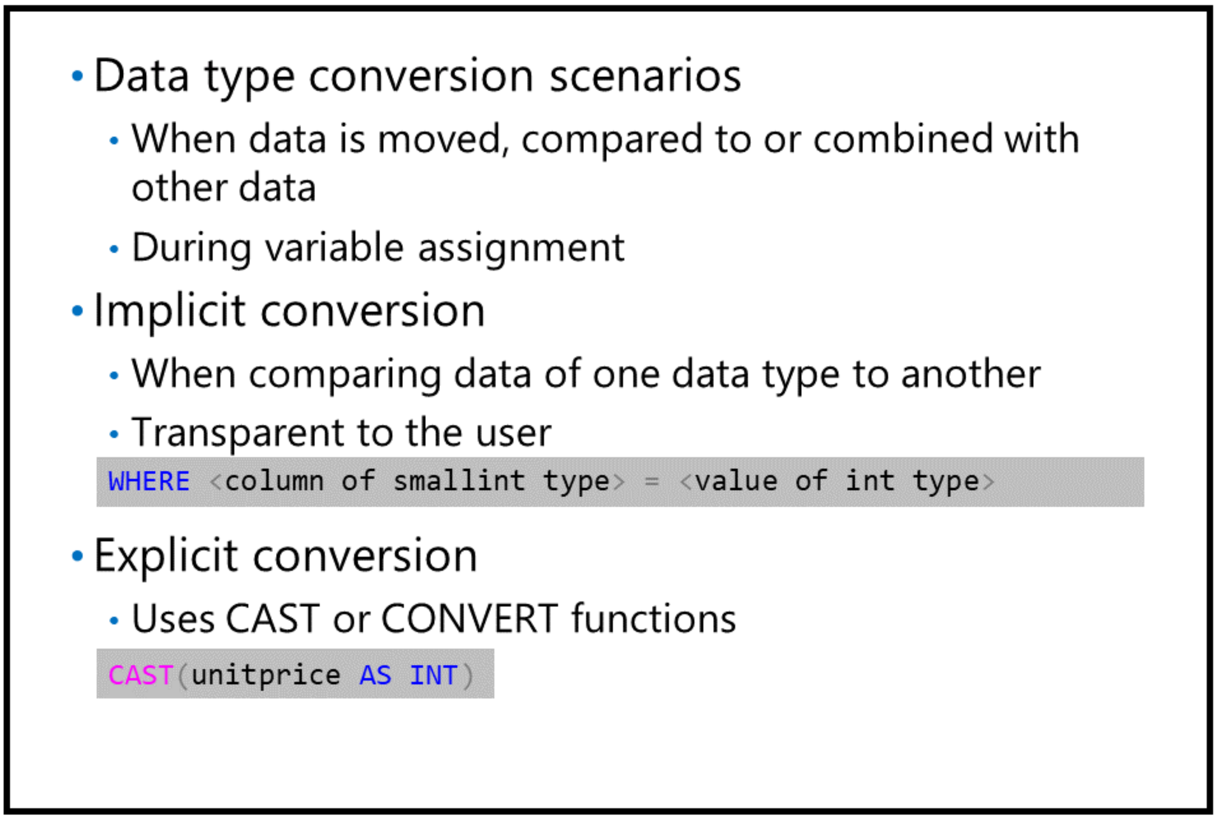
http://aka.ms/a8ihqi

For complete information on pairs of data types requiring implicit or explicit conversion, or for which no conversion is available, see the chart in the *Implicit Conversions* section of *CAST and CONVERT (Transact-SQL)*:

CAST and CONVERT (Transact-SQL) - Implicit Conversions

http://aka.ms/asaqq3

When are Data Types Converted?



- Data type conversion scenarios
  - When data is moved, compared to or combined with other data
  - During variable assignment
- Implicit conversion
  - When comparing data of one data type to another
  - Transparent to the user

```
WHERE <column of smallint type> = <value of int type>
```

- Explicit conversion
  - Uses CAST or CONVERT functions

```
CAST(unitprice AS INT)
```

When querying SQL Server, there are a number of scenarios in which data might be converted between data types:

- When data is moved, compared to, or combined with other data.
- During variable assignment.
- When using any operator that involves operands of different types.
- When T-SQL code explicitly converts one data type to another, using the CAST or CONVERT function.

Implicit Conversion Example - Integer Data Types

```
DECLARE @myTinyInt AS tinyint = 25;
DECLARE @myInt as int = 9999;
SELECT @myTinyInt + @myInt;
```

Implicit Conversion Example - Integer and Character Data Types

```
DECLARE @myChar AS char(5) = '6';
DECLARE @myInt AS int = 1;
SELECT @myChar + @myInt;
```

Question: In the example, which data type will be converted? To which data type will it be converted?

As you have learned, SQL Server will automatically attempt to perform an implicit conversion from a lower-precedence data type to a higher-precedence data type.

Failing Implicit Conversion Example

```
DECLARE @myChar AS char(5) = 'six';
DECLARE @myInt AS int = 1;
SELECT @myChar + @myInt;
```

Returns:

```
Msg 245, Level 16, State 1, Line 3
Conversion failed when converting the varchar value 'six' to data
type int.
```

Question: Why does SQL Server attempt to convert the character variable to an integer and not the other way around?

To force SQL Server to convert the **int** data type to a character data type for the purposes of this query, you need to explicitly convert it. You will learn how to do this in the next module.

To learn more about data type conversions, see Microsoft Docs:

Data Type Conversion (Database Engine)

http://aka.ms/t5db1i

Demonstration: SQL Server Data Types

In this demonstration, you will see how to convert data types.

Demonstration Steps

Convert Data Types

1. Ensure that the **MT17B-WS2016-NAT**, **20761C-MIA-DC**, and **20761C-MIA-SQL** virtual machines are running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Start SQL Server Management Studio and connect to your Azure instance of the **AdventureWorksLT** database engine instance using SQL Server authentication.
3. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod06\Demo** folder.
4. In Solution Explorer, expand **Queries**, and then double-click **11 - Demonstration A.sql**.
5. In the **Available Databases** list, click **AdventureWorks**.
6. Select the code under the comment **Step 2**, and then click **Execute**.
7. Select the code under the comment **Step 3**, and then click **Execute**. Note the error message.

8. Select the code under the comment **Step 4**, and then click **Execute**.

9. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Categorize Activity

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

varbinary

real

bigint

decimal

tinyint

binary

float

money

int

bit

Exact Numeric Data Types

Please drag items here

Approximate Numeric Data Types

Please drag items here

Binary Data Types

Please drag items here

Check answer Show solution Reset

Lesson 2: Working with Character Data

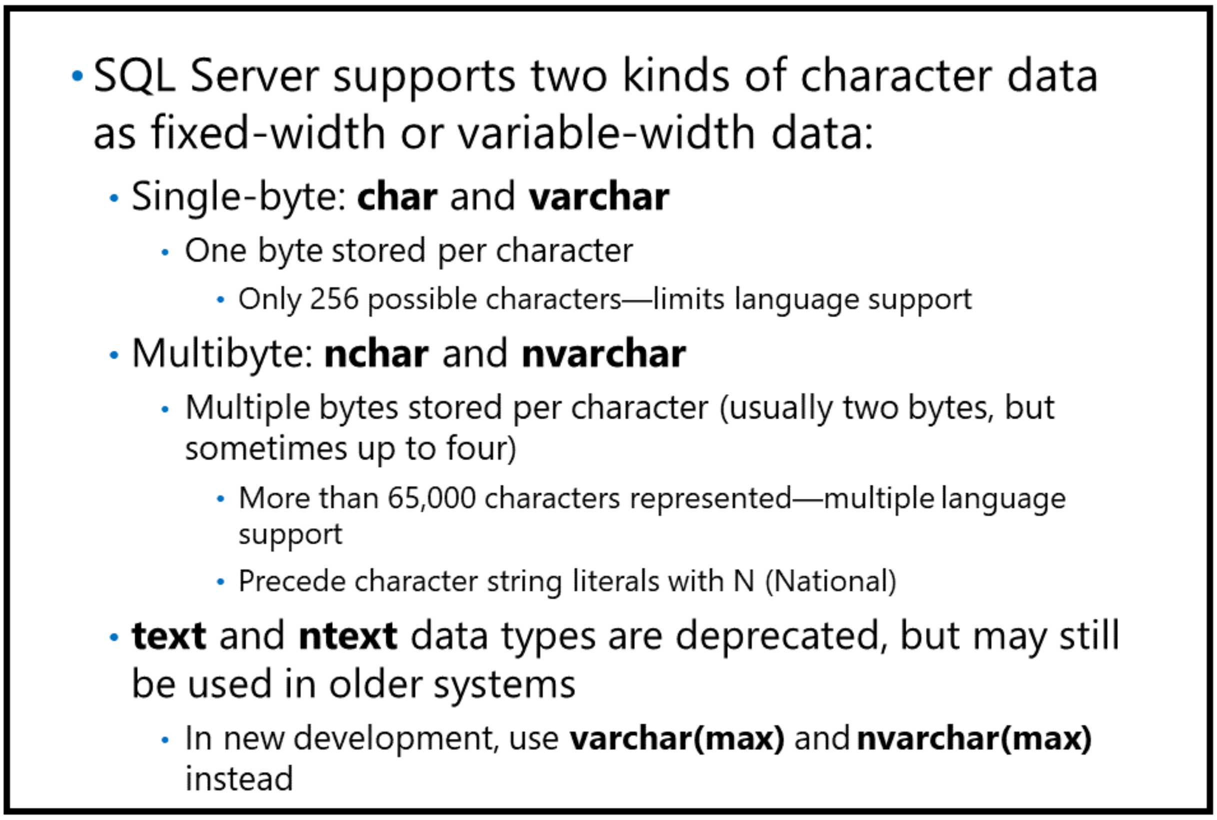
It is likely that the data you will work with in your T-SQL queries will include character data. As you will learn in this lesson, character data involves not only choices of capacity and storage, but also text-specific issues such as language, sort order, and collation. In this lesson, you will learn about the SQL Server character-based data types, how character comparisons work, and some common functions you might find useful in your queries.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the character data types supplied by SQL Server.
- Describe the impact of collation on character data.
- Concatenate strings.
- Extract and manipulate character data using built-in functions.
- Write queries using the LIKE predicate for matching patterns in character data.

Character Data Types



- SQL Server supports two kinds of character data as fixed-width or variable-width data:
  - Single-byte: **char** and **varchar**
    - One byte stored per character
      - Only 256 possible characters—limits language support
  - Multibyte: **nchar** and **nvarchar**
    - Multiple bytes stored per character (usually two bytes, but sometimes up to four)
      - More than 65,000 characters represented—multiple language support
      - Precede character string literals with N (National)
  - **text** and **ntext** data types are deprecated, but may still be used in older systems
    - In new development, use **varchar(max)** and **nvarchar(max)** instead

Even though SQL Server has many numeric data types, working with numeric data is relatively straightforward because numeric data follows a clearly defined set of mathematical rules.

By comparison, although there are fewer character data types available, working with character data in SQL Server can be more complicated. This is because you need to consider multiple languages, character sets, accented characters, sort rules and case sensitivity, and capacity and storage. Each of these factors might have an impact on which character data types you encounter when writing queries.

Character data types in SQL Server are categorized by two characteristics:

- Support for either fixed-width or variable-width data:
  - o Fixed-width data is always stored at a consistent size, regardless of the number of characters in the character data. Any unused space is filled with padding.
  - o Variable-width data is stored at the size of the character data, plus a small overhead.
- Support for either a single-byte character set or a multi-byte character set:
  - o A single-byte character set supports up to 256 different characters, stored as

one byte per character. By default, SQL Server uses the ASCII character set to interpret this data.

- A multi-byte character set supports more than 65,000 different characters by storing each character as multiple bytes—typically two bytes per character, but sometimes more. SQL Server uses the UNICODE UCS-2 character set to interpret this data.

The four available character data types support all possible combinations of these characteristics:

Data Type	Fixed Width?	Variable Width?	Single-Byte Characters?	Multi-Byte Characters?
char	Yes		Yes	
nchar	Yes			Yes
varchar		Yes	Yes	
nvarchar		Yes		Yes

Definitions for columns or variables take an optional value that defines the maximum length of the character data to be stored. You will almost always need to specify a value for the string length; if the maximum length value is not supplied, the default value is one character.

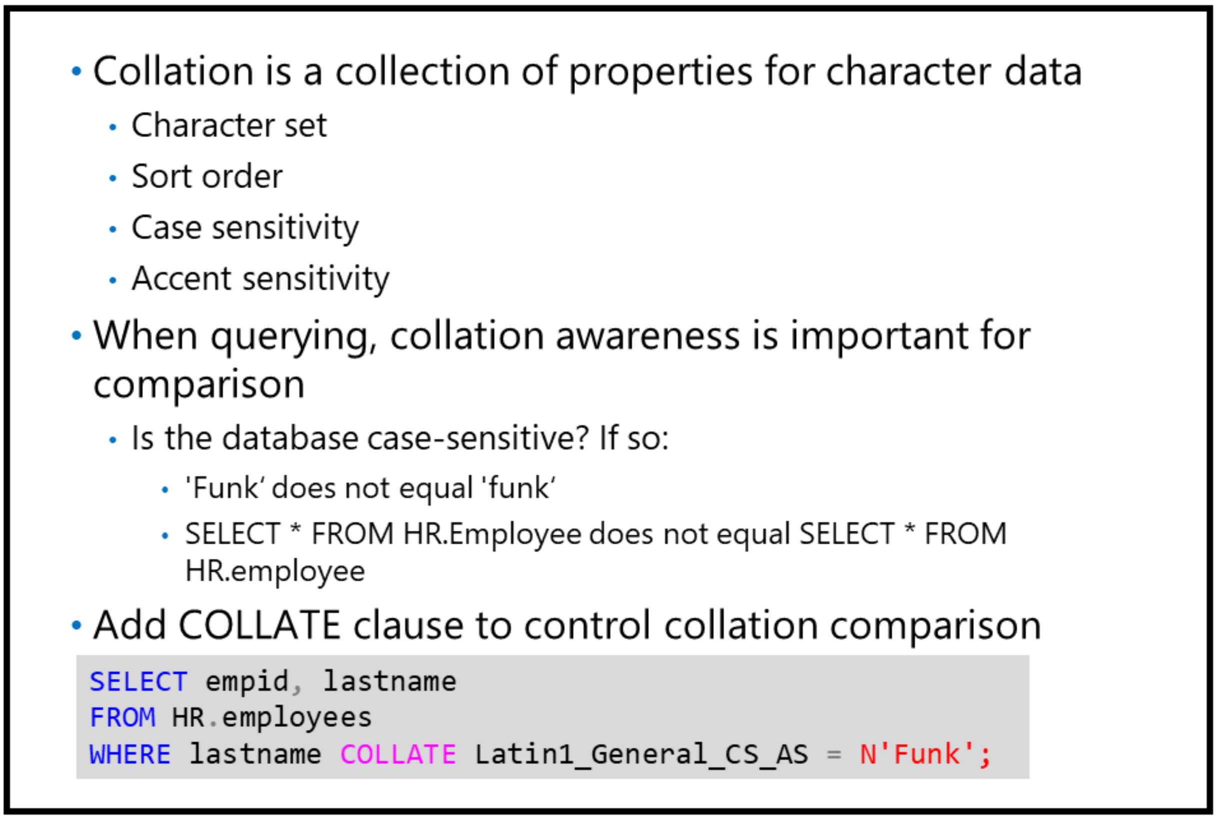
The **varchar** and **nvarchar** data types support the storage of very long strings of character data by using **max** for this value. Use of **varchar(max)** and **nvarchar(max)** replaces the use of the deprecated **text** and **ntext** types.

Data Type	Range	Storage
char(n) **nchar**(n)	1-8000 characters 1-4000 characters	n bytes, padded 2*n bytes, padded
varchar(n) **nvarchar**(n)	1-8000 characters 1-4000 characters	Actual length + 2 bytes
varchar(**max**) **nvarchar**(**max**)	Up to 2 GB	Actual length + 2 bytes

Note: All character data is delimited with single quotation marks.

- Single-byte character data is indicated with single quotation marks alone—for example 'SQL Server'.
- Multi-byte character data is indicated by single quotation marks with the prefix N (for National)— for example N'SQL Server'. The N prefix is always required, even when inserting the data into a column or variable with a multi-byte type.

Collation



- Collation is a collection of properties for character data
  - Character set
  - Sort order
  - Case sensitivity
  - Accent sensitivity
- When querying, collation awareness is important for comparison
  - Is the database case-sensitive? If so:
    - 'Funk' does not equal 'funk'
    - SELECT * FROM HR.Employee does not equal SELECT * FROM HR.employee
- Add COLLATE clause to control collation comparison

```
SELECT empid, lastname
FROM HR.employees
WHERE lastname COLLATE Latin1_General_CS_AS = N'Funk';
```

In addition to character byte count and length, SQL Server character data types are assigned a collation.

A collation is a collection of properties that determine several aspects of character data, including:

- Language or locale, from which is derived:
  - Character set

- o Sort order
- Case sensitivity
- Accent sensitivity

Note: A default collation is configured during the installation of SQL Server, but can be overridden on a per-database or per-column basis. As you will see, you might also override the current collation for some character data by explicitly setting a different collation in your query.

When querying, it is important to be aware of the collation settings for your character data—for example, whether it is case-sensitive.

The following query will return different results, depending on whether the column being tested in the WHERE clause is case-sensitive or not:

Case-Sensitivity Example (1)

```
SELECT empid, lastname
FROM HR.employees
WHERE lastname = N'Funk';
```

Case-Sensitivity Example (2)

```
SELECT empid, lastname
FROM HR.employees
WHERE lastname = N'funk';
```

The COLLATE clause can be used to override the collation of a column and force a different collation to be applied when the query is run.

Using COLLATE in the WHERE Clause

```
SELECT empid, lastname
FROM HR.employees
WHERE lastname COLLATE Latin1_General_CS_AS = N'Funk';
```

Note: Note that database-level collation settings apply to database object names (such as tables and views) as well as to character data.
For example, in a database with a case-sensitive default collation, the table names "HR.Employees" and "HR.employees" would refer to two different objects. In a database with a case-insensitive collation, the table names "HR.Employees" and "HR.employees" would refer to the same object.

For more information on this topic, see the SQL Server 2016 Technical Documentation:

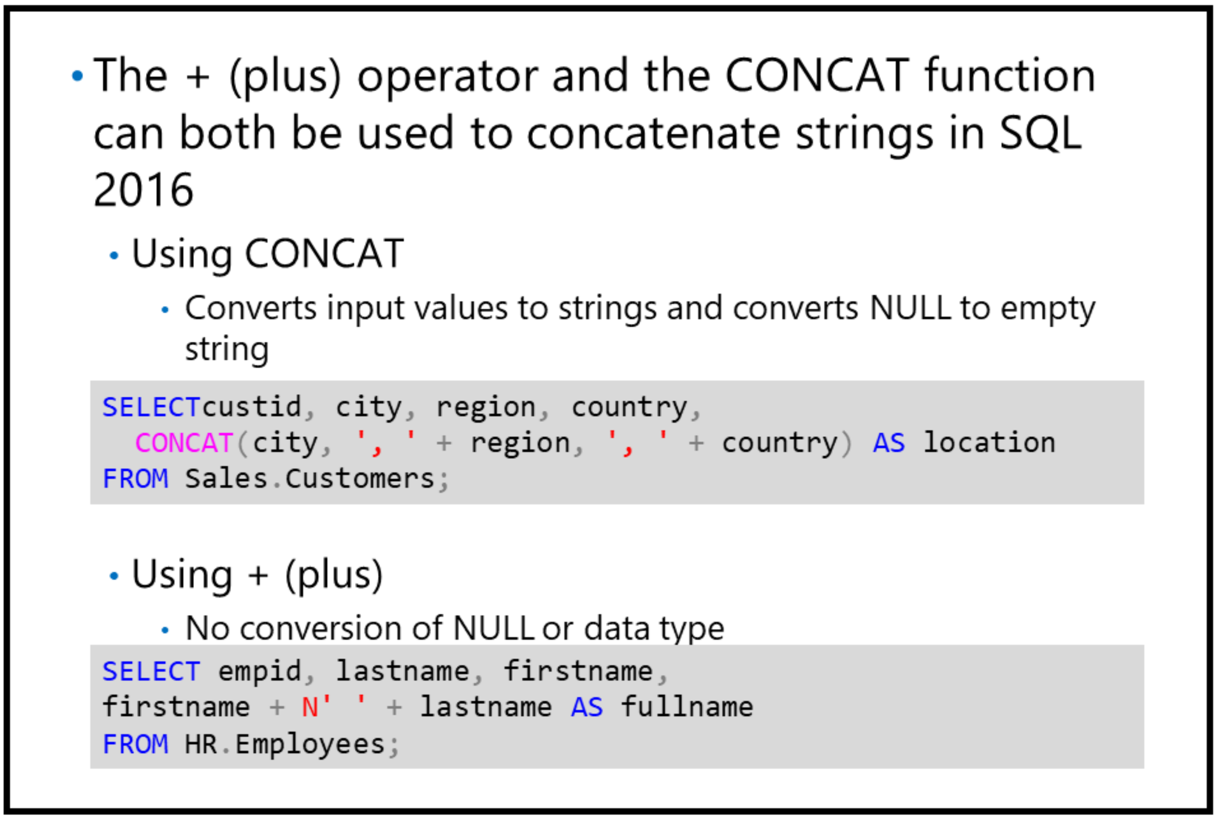
COLLATE (Transact-SQL)

http://aka.ms/ty97q8

Collation and Unicode Support

http://aka.ms/pm56d9

String Concatenation



- The + (plus) operator and the CONCAT function can both be used to concatenate strings in SQL 2016
  - Using CONCAT
    - Converts input values to strings and converts NULL to empty string

```
SELECTcustid, city, region, country,
  CONCAT(city, ', ' + region, ', ' + country) AS location
FROM Sales.Customers;
```

  - Using + (plus)
    - No conversion of NULL or data type

```
SELECT empid, lastname, firstname,
firstname + N' ' + lastname AS fullname
FROM HR.Employees;
```

There are multiple ways to concatenate, or join together, multiple character data, or string, values in SQL Server.

The CONCAT function takes at least two (or more) data values as arguments and returns a string value with the input values concatenated together.

If any of the input data values is not of a character data type, it will be implicitly converted to a character data type.

Any NULL values will be converted to an empty string.

CONCAT Function Syntax

```
CONCAT ( string_value1, string_value2 [, string_valueN ] )
```

Concatenating Strings Using CONCAT

```
SELECT custid, city, region, country,
CONCAT(city, ', ' + region, ', ' + country) AS location
```

```
FROM Sales.Customers;
```

Part of the result returned by this query is shown below:

```
custid city        region country  location
------ ----------- ------ -------- -------------------
1      Berlin      NULL   Germany  Berlin, Germany
2      México D.F. NULL   Mexico   México D.F., Mexico
3      México D.F. NULL   Mexico   México D.F., Mexico
4      London      NULL   UK       London, UK
5      Luleå       NULL   Sweden   Luleå, Sweden
```

See the topic *CONCAT (Transact-SQL)* in the SQL Server 2016 Technical Documentation:

CONCAT (Transact-SQL)

http://aka.ms/b9c34t

The CONCAT function was introduced in SQL Server 2012.

In earlier versions of SQL Server than 2012, the CONCAT function is not available; string concatenation is carried out using the + (plus) operator.

If any of the string values concatenated with the + operator is NULL, the output string will be NULL.

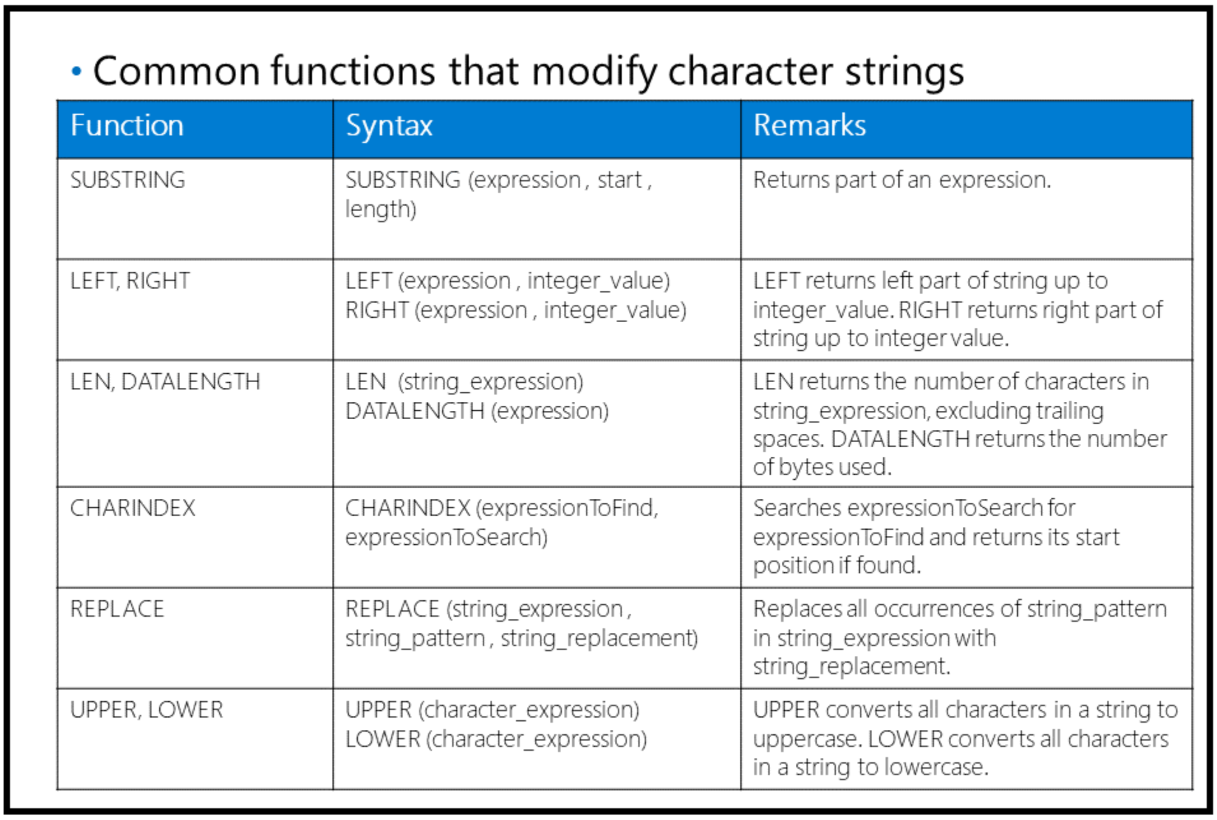
No conversion of data types is carried out (see note below).

Concatenating Strings Using +

```
SELECT
empid, lastname, firstname, firstname + N' ' + lastname AS fullname
FROM HR.Employees;
FROM HR.Employees;
```

Note: Since the plus sign is also used for arithmetic addition, consider whether any of your data is of a numeric data type when concatenating. Character data types have a lower precedence than numeric data type, and SQL Server will attempt to convert and add mixed data types rather than concatenating them.

Character String Functions



- Common functions that modify character strings

Function	Syntax	Remarks
SUBSTRING	SUBSTRING (expression , start , length)	Returns part of an expression.
LEFT, RIGHT	LEFT (expression , integer_value) RIGHT (expression , integer_value)	LEFT returns left part of string up to integer_value. RIGHT returns right part of string up to integer value.
LEN, DATALENGTH	LEN (string_expression) DATALENGTH (expression)	LEN returns the number of characters in string_expression, excluding trailing spaces. DATALENGTH returns the number of bytes used.
CHARINDEX	CHARINDEX (expressionToFind, expressionToSearch)	Searches expressionToSearch for expressionToFind and returns its start position if found.
REPLACE	REPLACE (string_expression , string_pattern , string_replacement)	Replaces all occurrences of string_pattern in string_expression with string_replacement.
UPPER, LOWER	UPPER (character_expression) LOWER (character_expression)	UPPER converts all characters in a string to uppercase. LOWER converts all characters in a string to lowercase.

In addition to retrieving character data as is from SQL Server, you may also need to extract portions of text or determine the location of characters within a larger string. SQL Server provides a number of built-in functions to accomplish these tasks. Some of these functions include:

- FORMAT—allows you to format an input value to a character string based on a .NET format string, with an optional culture parameter.

  FORMAT Function

```
DECLARE @m money = 120.595
```

```
SELECT @m AS unformatted_value,
FORMAT(@m,'C','zh-cn') AS zh_cn_currency,
FORMAT(@m,'C','en-us') AS en_us_currency,
FORMAT(@m,'C','de-de') AS de_de_currency;
```

Returns:

```
unformatted_value zh_cn_currency en_us_currency de_de_currency
----------------- -------------- -------------- --------------
120.595           ¥120.60        $120.60        120,60 €
```

- SUBSTRING—allows you to return part of a character string given a starting point and a number of characters to return.

  SUBSTRING Example

```
SELECT SUBSTRING('Microsoft SQL Server ',11,3) AS Result;
```

Returns:

```
Result
-------
SQL
```

- LEFT and RIGHT—allows you to return a number of characters from the left or right of a string.

  LEFT and RIGHT Example

```
SELECT LEFT('Microsoft SQL Server',9) AS left_example,
RIGHT('Microsoft SQL Server',6) AS right_example;
```

Returns:

```
left_example right_example
------------ -------------
Microsoft    Server
```

- LEN and DATALENGTH—allows you to query metadata about the number of characters or the number of bytes stored in a string.

LEN and DATALENGTH Example

```
SELECT LEN('Microsoft SQL Server     ') AS [LEN];
SELECT DATALENGTH('Microsoft SQL Server     ') AS [DATALENGTH];
```

Returns:

```
LEN
-----------
20
DATALEN
-----------
25
```

- CHARINDEX—allows you to query the start position of a string within another string. If the target string is not found, CHARINDEX returns 0.

CHARINDEX Example

```
SELECT CHARINDEX('SQL','Microsoft SQL Server') AS Result;
```

Returns:

```
Result
-----------
11
```

- REPLACE—allows you to substitute one string for another within a target string.

REPLACE Example

```
SELECT REPLACE('Learning about T-SQL string functions','T-
SQL','Transact-SQL') AS Result;
```

Returns:

```
Result
--------------------------------------
Learning about Transact-SQL string functions
```

- UPPER and LOWER—for performing character case conversions.

UPPER and LOWER Example

```
SELECT UPPER('Microsoft SQL Server') AS [UP],LOWER('Microsoft SQL
Server') AS [LOW];
```

Returns:

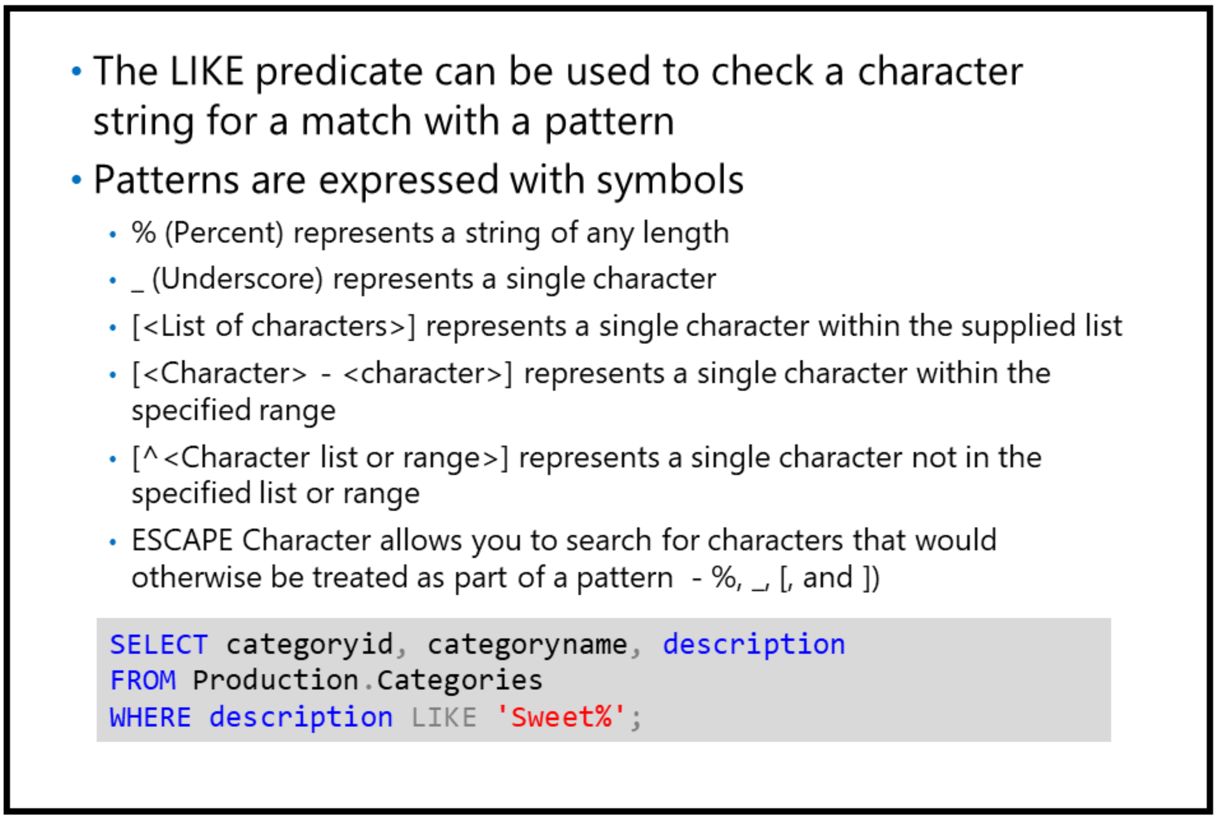
```
UP                   LOW
-------------------- --------------------
MICROSOFT SQL SERVER microsoft sql server
```

For references on these and other string functions, see the SQL Server 2016 Technical Documentation:

String Functions (Transact-SQL)

http://aka.ms/lt6hg9

The LIKE Predicate



- The LIKE predicate can be used to check a character string for a match with a pattern
- Patterns are expressed with symbols
  - % (Percent) represents a string of any length
  - _ (Underscore) represents a single character
  - [<List of characters>] represents a single character within the supplied list
  - [<Character> - <character>] represents a single character within the specified range
  - [^<Character list or range>] represents a single character not in the specified list or range
  - ESCAPE Character allows you to search for characters that would otherwise be treated as part of a pattern - %, _, [, and])

```
SELECT categoryid, categoryname, description
FROM Production.Categories
WHERE description LIKE 'Sweet%';
```

Character-based data in SQL Server provides for more than exact matches in your queries. Through the use of the LIKE predicate, you can also perform pattern matching in your WHERE clause.

The LIKE predicate allows you to check a character string against a pattern. Patterns are expressed with symbols, which can be used alone or in combinations to search within your strings:

- **% (Percent)** represents a string of any length. For example, LIKE N'Sand%' will match 'Sand', 'Sandwich', 'Sandwiches', and so on.

- **_ (Underscore)** represents a single character. For example, LIKE N'_a%' will match any string whose second character is an 'a'.

- **[<List of characters>]** represents a single character within the supplied list. For example, LIKE N'[DEF]%' will find any string that starts with a 'D', an 'E', or an 'F'.
- **[<Character> - <character>]** represents a single character within the specified range. For example, LIKE N'[N-Z]%' will match any string that starts with a letter of the alphabet between N and Z, inclusive.
- **[^<Character list or range>]** represents a single character not in the specified list or range. For example, LIKE N'^[A]% ' will match a string beginning with any other character than an 'A'.
- **ESCAPE** is used to set an escape character, meaning you can search for a character that is a wildcard character but to treat it as a literal, rather than a wildcard. Each instance of the special character to be treated as a literal must be preceded by the specified escape character. For example, LIKE N'10!% off%' ESCAPE '!' will match any string that starts with ‘10% off’, but would not match the string ‘100 special offers’ (which would be matched if the ESCAPE character was not used).

For further information on LIKE, see the SQL Server 2016 Technical Documentation:

LIKE (Transact-SQL)

http://aka.ms/rm8ihw

Demonstration: Working with Character Data

In this demonstration, you will see how to manipulate character data.

Demonstration Steps

Manipulate Character Data

1. In Solution Explorer, open the **21 - Demonstration B.sql** script file.
2. In the **Available Databases** list, click **AdventureWorks**.
3. Select the code under the comment **Step 2**, and then click **Execute**.

4. Select the code under the comment **Step 3a**, and then click **Execute**.

5. Select the code under the comment **Step 3b**, and then click **Execute**.

6. Select the code under the comment **Step 4**, and then click **Execute**.

7. Select the code under the comment **Step 5**, and then click **Execute**.

8. Select all the code under the comment **Step 6**, and then click **Execute**.

9. Select the code under the comment **Step 7**, and then click **Execute**.

10. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Discovery

You have the following query:
SELECT FirstName
FROM HumanResources.Employees
WHERE FirstName LIKE N'[^MA]%'
Will the query return an employee with the first name 'Matthew'?

Show solution Reset

Lesson 3: Working with Date and Time Data

Date and time data is very common when you are working with SQL Server data types. In this lesson, you will learn which data types are used to store date and time data; how to enter dates and times so they will be properly parsed by SQL Server; and how to manipulate dates and times with built-in functions.

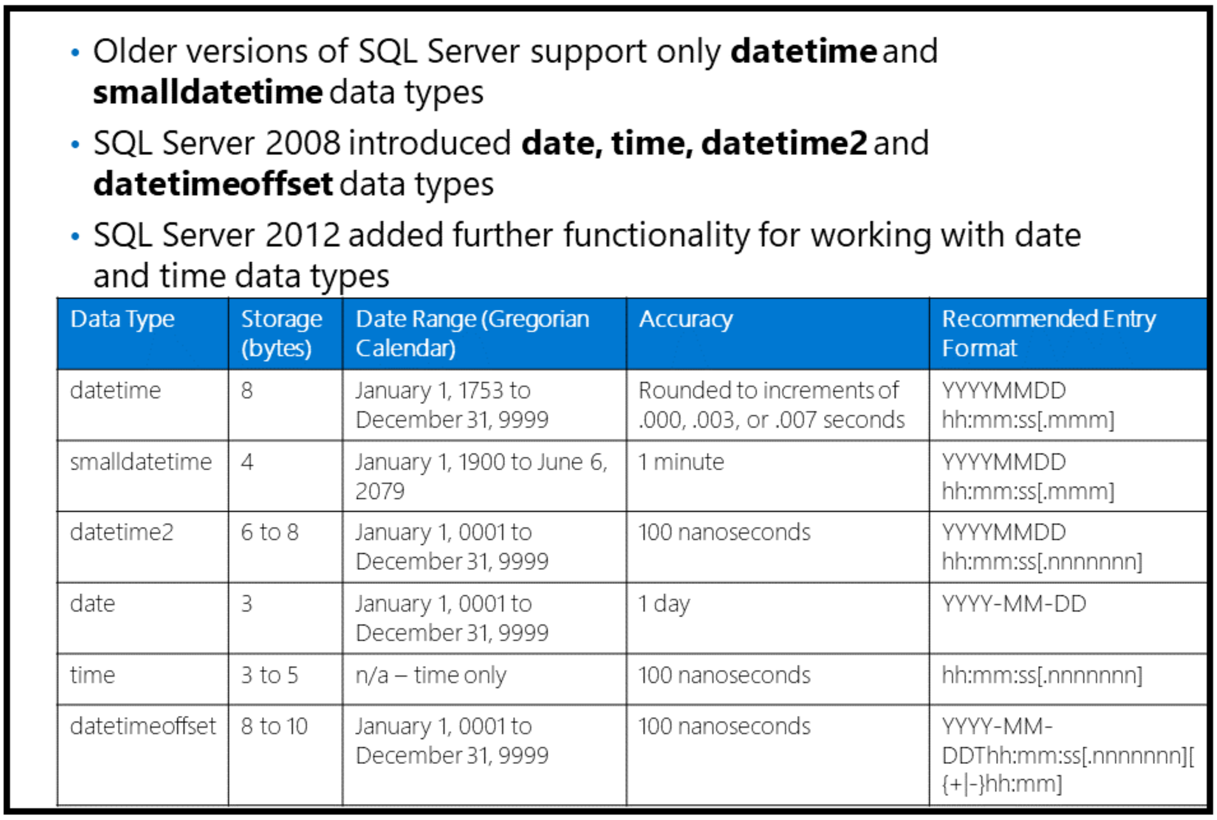
Lesson Objectives

After completing this lesson, you will be able to:

- Describe the data types used to store date and time information.

- Enter dates and times as literal values for SQL Server to convert to date and time types.
- Write queries comparing dates and times.
- Write queries using built-in functions to manipulate dates and extract date parts.

Date and Time Data Types



- Older versions of SQL Server support only **datetime** and **smalldatetime** data types
- SQL Server 2008 introduced **date, time, datetime2** and **datetimeoffset** data types
- SQL Server 2012 added further functionality for working with date and time data types

Data Type	Storage (bytes)	Date Range (Gregorian Calendar)	Accuracy	Recommended Entry Format
datetime	8	January 1, 1753 to December 31, 9999	Rounded to increments of .000, .003, or .007 seconds	YYYYMMDD hh:mm:ss[.mmm]
smalldatetime	4	January 1, 1900 to June 6, 2079	1 minute	YYYYMMDD hh:mm:ss[.mmm]
datetime2	6 to 8	January 1, 0001 to December 31, 9999	100 nanoseconds	YYYYMMDD hh:mm:ss[.nnnnnnn]
date	3	January 1, 0001 to December 31, 9999	1 day	YYYY-MM-DD
time	3 to 5	n/a – time only	100 nanoseconds	hh:mm:ss[.nnnnnnn]
datetimeoffset	8 to 10	January 1, 0001 to December 31, 9999	100 nanoseconds	YYYY-MM-DDThh:mm:ss[.nnnnnnn][{+|-}hh:mm]

There has been a progression in SQL Server's handling of temporal data as newer versions are released. As you may need to work with data created for older versions of SQL Server, even though you're writing queries for SQL Server 2016, it will be useful to review past support for date and time data:

- Before SQL Server 2008, there were only two data types for date and time data: **datetime** and **smalldatetime**. Each of these stored both the date and the time in a single value. For example, a **datetime** could store '20140212 08:30:00' to represent February 12, 2014 at 08:30.
- In SQL Server 2008, Microsoft introduced four new data types: **datetime2**, **date**, **time**, and **datetimeoffset**. These addressed issues of precision, capacity, time

zone tracking, and separating dates from times. For new work, Microsoft recommends these types over the older **datetime** and **smalldatetime**.

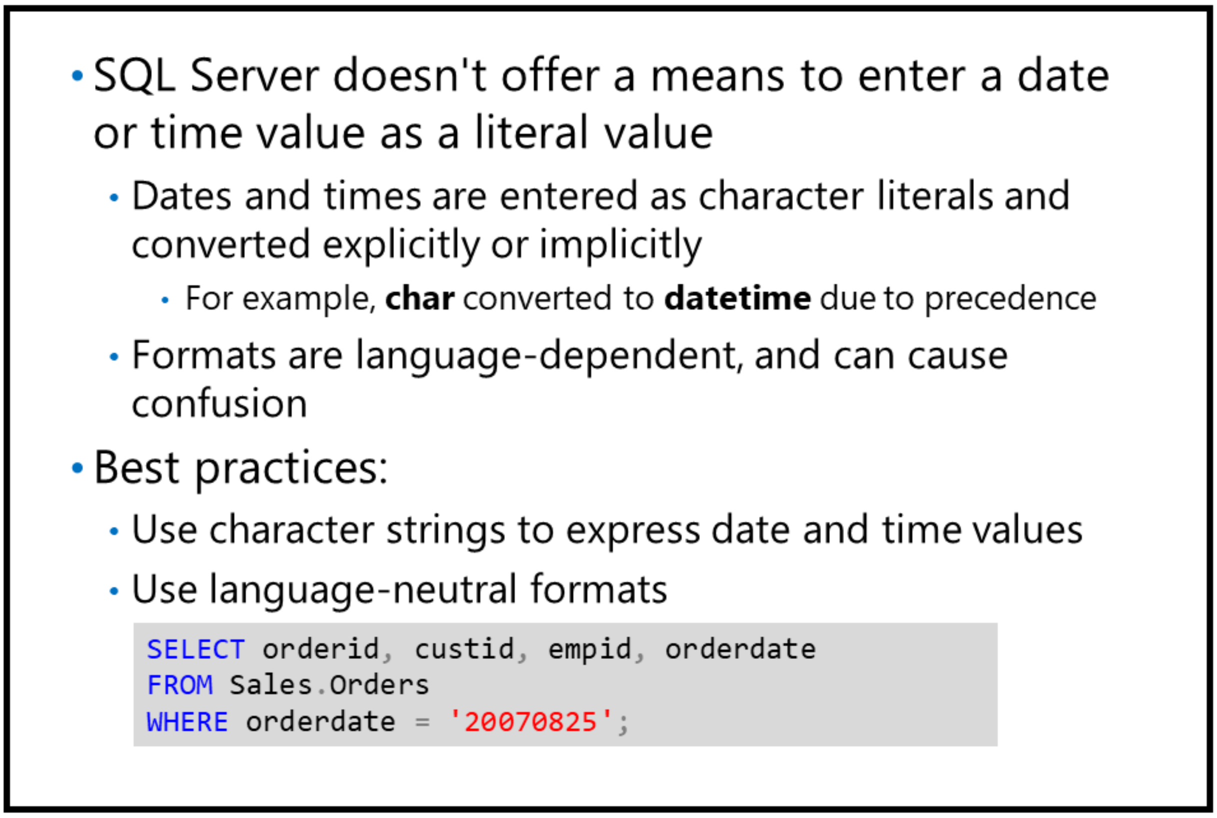
- In SQL Server 2012, Microsoft introduced new functions for working with partial data from date and time data types (such as DATEFROMPARTS) and for performing calculations on dates (such as EOMONTH).

For more information on all the date and time data types, see the SQL Server 2016 Technical Documentation:

Date and Time Types

http://aka.ms/aekgy8

Entering Date and Time Data Types Using Strings



- SQL Server doesn't offer a means to enter a date or time value as a literal value
  - Dates and times are entered as character literals and converted explicitly or implicitly
    - For example, **char** converted to **datetime** due to precedence
  - Formats are language-dependent, and can cause confusion
- Best practices:
  - Use character strings to express date and time values
  - Use language-neutral formats

```sql
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
WHERE orderdate = '20070825';
```

To use date and time data in your queries, you will need to be able to represent date and time data in T-SQL. SQL Server doesn't offer the means to enter dates and times as literal values, so you will use character strings (often referred to as string literals) which are delimited, like all other strings in SQL Server, with single quotes. SQL Server will implicitly convert the string literals to date and time values. (You might also explicitly convert string literals with the T-SQL CAST and CONVERT functions, which you will learn about in the next module.)

SQL Server can interpret a wide variety of string literal formats as dates but, for consistency and to avoid issues with language or nationality interpretation, it is recommended that you use a neutral format, such as 'YYYYMMDD'. To represent February 12, 2014, you would use the literal '20140212'.

String Literals Example

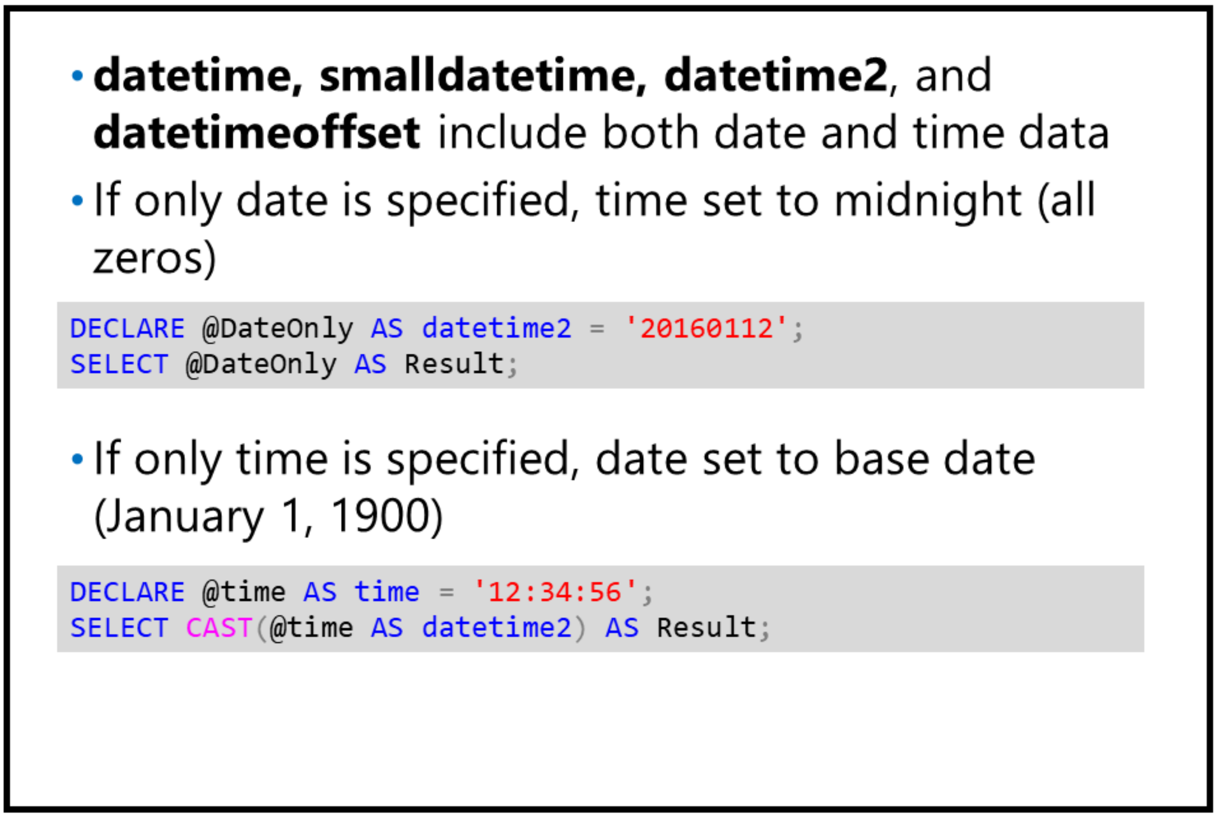
```
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
WHERE orderdate = '20070825';
```

Various other language-neutral formats for date and time literals are available to you:

Data Type	Language-Neutral Formats	Examples
datetime	'YYYYMMDD hh:mm:ss.nnn' 'YYYY-MM-DDThh:mm:ss.nnn' 'YYYYMMDD'	'20140212 12:30:15.123' '2014-02-12T12:30:15.123' '20140212'
smalldatetime	'YYYYMMDD hh:mm' 'YYYY-MM-DDThh:mm' 'YYYYMMDD'	'20140212 12:30' '2014-02-12T12:30' '20140212'
datetime2	'YYYY-MM-DD' 'YYYYMMDD hh:mm:ss.nnnnnnn' 'YYYY-MM-DD hh:mm:ss.nnnnnnn' 'YYYY-MM-DDThh:mm:ss.nnnnnnn' 'YYYYMMDD' 'YYYY-MM-DD'	'2014-02-12' '20140212 12:30:15.1234567' '2014-02-12 12:30:15.1234567' '2014-02-12T12:30:15.1234567' '20140212' '2014-02-12'
date	'YYYYMMDD' 'YYYY-MM-DD'	'20140212' '2014-02-12'
time	'hh:mm:ss.nnnnnnn'	'12:30:15.1234567'

Data Type	Language-Neutral Formats	Examples
datetimeoffset	'YYYYMMDD hh:mm:ss.nnnnnnn [+|-]hh:mm' 'YYYY-MM-DD hh:mm:ss.nnnnnnn [+|-]hh:mm' 'YYYYMMDD' 'YYYY-MM-DD'	'20140212 12:30:15.1234567 +02:00' '2014-02-12 12:30:15.1234567 +02:00' '20140212' '2014-02-12'

Working Separately with Date and Time



- **datetime, smalldatetime, datetime2**, and **datetimeoffset** include both date and time data
- If only date is specified, time set to midnight (all zeros)

```
DECLARE @DateOnly AS datetime2 = '20160112';
SELECT @DateOnly AS Result;
```

- If only time is specified, date set to base date (January 1, 1900)

```
DECLARE @time AS time = '12:34:56';
SELECT CAST(@time AS datetime2) AS Result;
```

As you have learned, some SQL Server temporal data types store both date and time together in one value. **datetime** and **datetime2** combine year, month, day, hour, minute, seconds, and more. The **datetimeoffset** data type also adds time zone information to the date and time. The time and date components are optional in combination data types such as **datetime2**. So, when using these data types, you should be aware of how they behave when provided with only partial data:

- If only the date is provided, the time portion of the data type is filled with zeros and the time is considered to be set at midnight.

datetime2 with No Time

```
DECLARE @DateOnly AS datetime2 = '20160112';
SELECT @DateOnly AS Result;
```

Returns:

```
Result
----------------------------
2016-01-12 00:00:00.0000000
```

- If a data type that holds both date and time—such as **datetime** or **datetime2**—data is populated only with time data, the date portion of the value will be set to a default value of January 1, 1900. If you need to store time data alone, use the **time** data type.

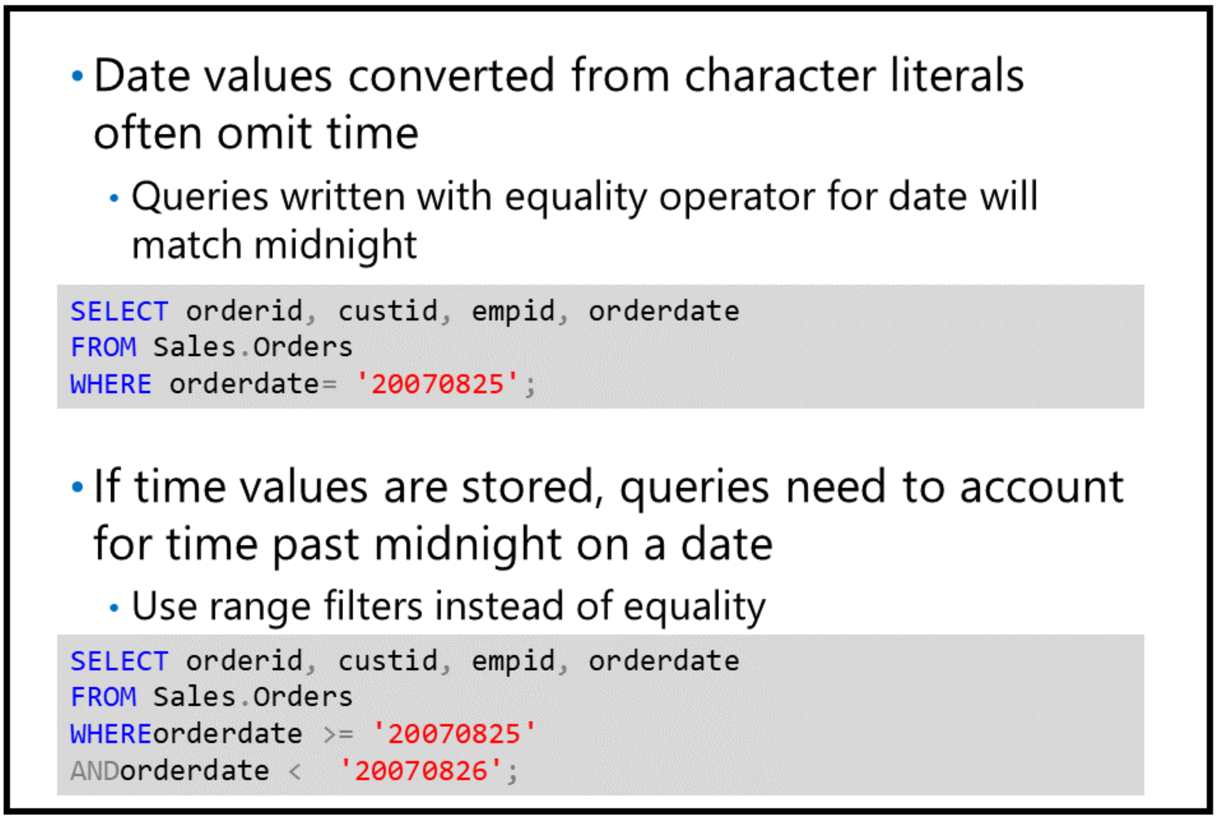
  Default Date Example

```
DECLARE @time AS time = '12:34:56';
SELECT CAST(@time AS datetime2) AS Result;
```

Returns:

```
Result
---------------------------
1900-01-01 12:34:56.0000000
```

Querying Date and Time Values



- Date values converted from character literals often omit time
  - Queries written with equality operator for date will match midnight

```
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
WHERE orderdate= '20070825';
```

- If time values are stored, queries need to account for time past midnight on a date
  - Use range filters instead of equality

```
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
WHEREorderdate >= '20070825'
ANDorderdate <  '20070826';
```

When querying date and time data types, you might need to consider both the date and time portions of the data to return the results you expect.

Midnight Time Values Example

```
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
WHERE orderdate= '20070825';
```

This query might satisfy the user’s requirement—note that only sales orders with an order date of midnight on August 25, 2007 are returned:

```
orderid     custid      empid       orderdate
----------- ----------- ----------- -----------------------
10643       1           6           2007-08-25 00:00:00.000
10644       88          3           2007-08-25 00:00:00.000
```

This is because SQL Server implicitly converts the string literal '20070825' used in the query to the same data type as the Sales.Orders.orderdate column—**datetime**—and in doing so applies the default value of midnight for the time portion of the value.

Midnight Time Values Example (2)

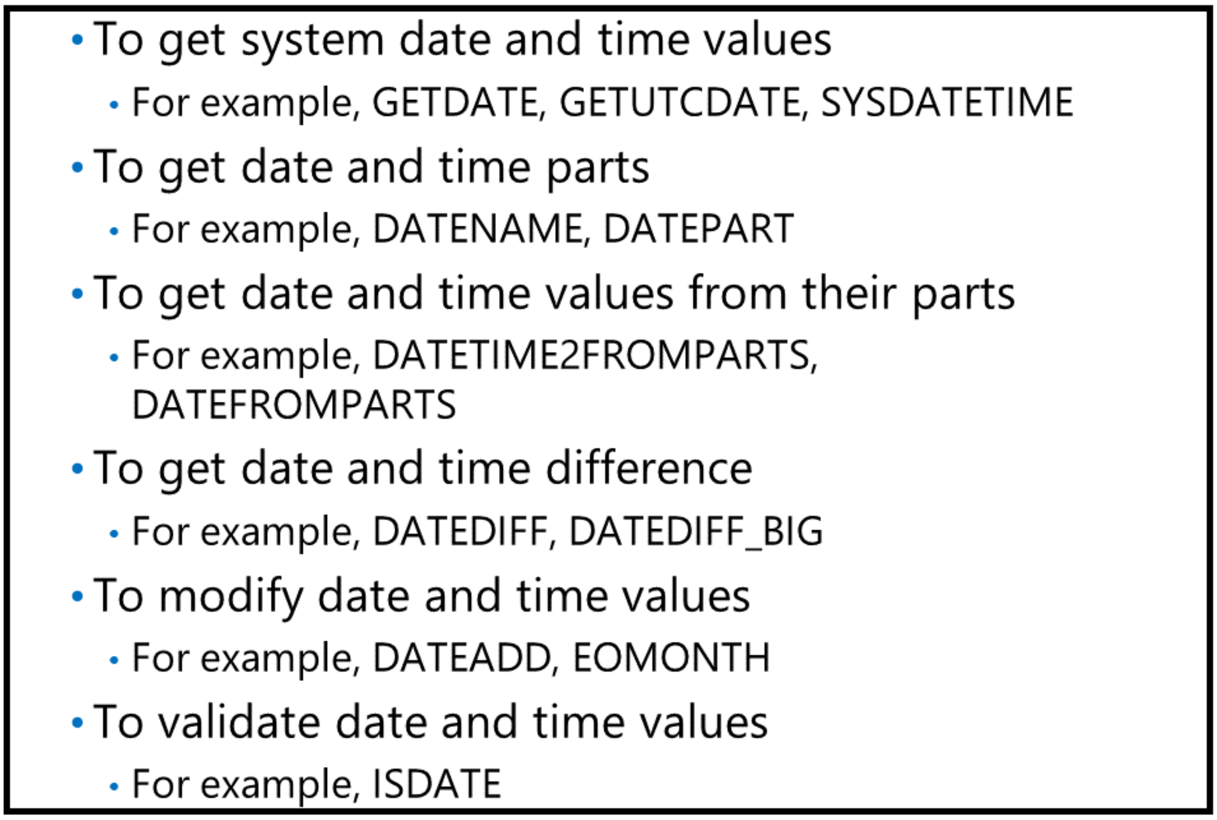
```
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
WHERE orderdate= '20070825 00:00:00.000';
```

This means that only values that exactly match midnight are returned. If there are rows in the table with an order date of August 25, 2007 but with a time after midnight, they would not be returned by this query.

Querying a Date Range Example

```
SELECT orderid, custid, empid, orderdate
FROM Sales.Orders
WHEREorderdate >= '20070825'
ANDorderdate <  '20070826';
```

Date and Time Functions



- To get system date and time values
  - For example, GETDATE, GETUTCDATE, SYSDATETIME
- To get date and time parts
  - For example, DATENAME, DATEPART
- To get date and time values from their parts
  - For example, DATETIME2FROMPARTS, DATEFROMPARTS
- To get date and time difference
  - For example, DATEDIFF, DATEDIFF_BIG
- To modify date and time values
  - For example, DATEADD, EOMONTH
- To validate date and time values
  - For example, ISDATE

SQL Server provides a number of functions designed to manipulate date and time data:

- Functions that return current date and time, offering you choices between various return types, in addition to whether to include or exclude time zone information.

- Functions that return parts of date and time values, enabling you to extract only the portion of a date or time that your query requires. Note that DATENAME and DATEPART offer functionality similar to one another. The difference between them is the return type.

- Functions that return date and time typed data from components such as separately supplied year, month, and day. This offers an alternative to providing dates as string literals, as already covered in this lesson. Note that these functions require all parts of the target date/time data to be provided.

- Functions that modify date and time values, including to increment dates, to calculate the last day of a month, and to alter time zone offset information.

- Functions that examine date and time values, returning metadata or calculations about intervals between input dates.

For details of all date and time functions, see the SQL Server 2016 Technical Documentation at:

Date and Time Data Types and Functions (Transact-SQL)

http://aka.ms/ifob87

Demonstration: Working with Date and Time Data

In this demonstration, you will see how to query date and time values.

Demonstration Steps

Query Data and Time Values

1. In Solution Explorer, open the **31 - Demonstration C.sql** script file.
2. In the **Available Databases** list, click **AdventureWorks**.
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**.
8. Select the code under the comment **Step 7**, and then click **Execute**.
9. Close SQL Server Management Studio without saving any files.

Check Your Knowledge

Categorize Activity

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

date

datetimeoffset

DATEFROMPARTS

datetime

EOMONTH

datetime2

time

smalldatetime

Present in all versions of SQL Server

Please drag items here

Only present in SQL Server 2008 and later

Please drag items here

Only present in SQL Server 2012 and later

Please drag items here

Check answer Show solution Reset

Lab: Working with SQL Server 2016 Data Types

Scenario

You are an Adventure Works business analyst who will be writing reports using corporate databases stored in SQL Server 2016. You have been given a set of business requirements for data and you will write T-SQL queries to retrieve the specified data from the databases. You will need to retrieve and convert character, and date and time data into various formats.

Objectives

After completing this lab, you will be able to:

- Write queries that return date and time data.
- Write queries that use date and time functions.
- Write queries that return character data.
- Write queries that use character functions.

Lab Setup

Estimated Time: 90 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Writing Queries That Return Date and Time Data

Scenario

Before you start using different date and time functions in business scenarios, you should practice on sample data.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Write a SELECT Statement to Retrieve Information About the Current Date
3. Write a SELECT Statement to Return the Date Data Type
4. Write a SELECT Statement That Uses Different Date and Time Functions
5. Write a SELECT Statement to Show Whether a Table of Strings Can Be Used as Dates

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Result: After this exercise, you should be able to retrieve date and time data using T-SQL.

Exercise 2: Writing Queries That Use Date and Time Functions

Scenario

The sales department wants to have different reports that focus on data during specific time frames. The sales staff would like to analyze distinct customers, distinct products, and orders placed near the end of the month. You should write the SELECT statements using the different date and time functions.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement to Retrieve Customers with Orders in a Given Month
2. Write a SELECT Statement to Calculate the First and Last Day of the Month

3. Write a SELECT Statement to Retrieve the Orders Placed in the Last Five Days of the Ordered Month

4. Write a SELECT Statement to Retrieve All Distinct Products Sold in the First 10 Weeks of the Year 2007

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Result: After this exercise, you should know how to use the date and time functions.

Exercise 3: Writing Queries That Return Character Data

Scenario

Members of the marketing department would like to have a more condensed version of a report for when they talk with customers. They want the information that currently exists in two columns displayed in a single column.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement to Concatenate Two Columns

2. Add an Additional Column to the Concatenated String Which Might Contain NULL

3. Write a SELECT Statement to Retrieve Customer Contacts Based on the First Character in the Contact Name

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Result: After this exercise, you should have an understanding of how to concatenate character data.

Exercise 4: Writing Queries That Use Character Functions

Scenario

The marketing department want to address customers by their first and last names. In the Sales.Customers table, there is only one column named contactname—it has both elements separated by a comma. You will have to prepare a report to show the first and last names separately.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement That Uses the SUBSTRING Function
2. Write a Query to Retrieve the Contact's First Name Using SUBSTRING
3. Write a SELECT Statement to Format the Customer ID
4. Challenge: Write a SELECT Statement to Return the Number of Character Occurrences

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Detailed Steps ▼

Result: After this exercise, you should have an understanding of how to use the character functions.

Module Review and Takeaways

In this module, you have learned how to:

- Describe SQL Server data types, type precedence, and type conversions.
- Write queries using numeric data types.
- Write queries using character data types.
- Write queries using date and time data types.

Review Question(s)

Check Your Knowledge

Discovery

Will SQL Server be able to successfully and implicitly convert an int data type to a varchar?

Show solution Reset

Check Your Knowledge

Discovery

What data type is suitable for storing Boolean flag information, such as TRUE or FALSE?

Show solution Reset

Check Your Knowledge

Discovery

What logical operators are useful for retrieving ranges of date and time values?

Show solution Reset