Module 8: Using Built-In Functions

Contents:

	Module Overview
Lesson 1:	Writing Queries with Built-In Functions
Lesson 2:	Using Conversion Functions
Lesson 3:	Using Logical Functions
Lesson 4:	Using Functions to Work with NULL
Lab:	Using Built-in Functions
	Module Review and Takeaways

Module Overview

In addition to retrieving data as it is stored in columns, you may have to compare or further manipulate values in your T-SQL queries.

In this module, you will:

- Learn about the many built-in functions in Microsoft® SQL Server® that provide data type conversion, comparison, and NULL handling.
- Learn about the various types of functions in SQL Server and how they are categorized.
- Work with scalar functions and see where they may be used in your queries.
- Learn conversion functions for changing data between different data types, and how to write logical tests.
- Learn how to work with NULLs, and use built-in functions to select non-NULL values, in addition to replacing certain values with NULL when applicable.

Objectives

After completing this module, you will be able to:

- Write queries with built-in scalar functions.
- Use conversion functions.
- Use logical functions.
- Use functions that work with NULL.

Lesson 1: Writing Queries with Built-In Functions

SQL Server provides many built-in functions, ranging from those that perform data type conversion, to those that aggregate and analyze groups of rows.

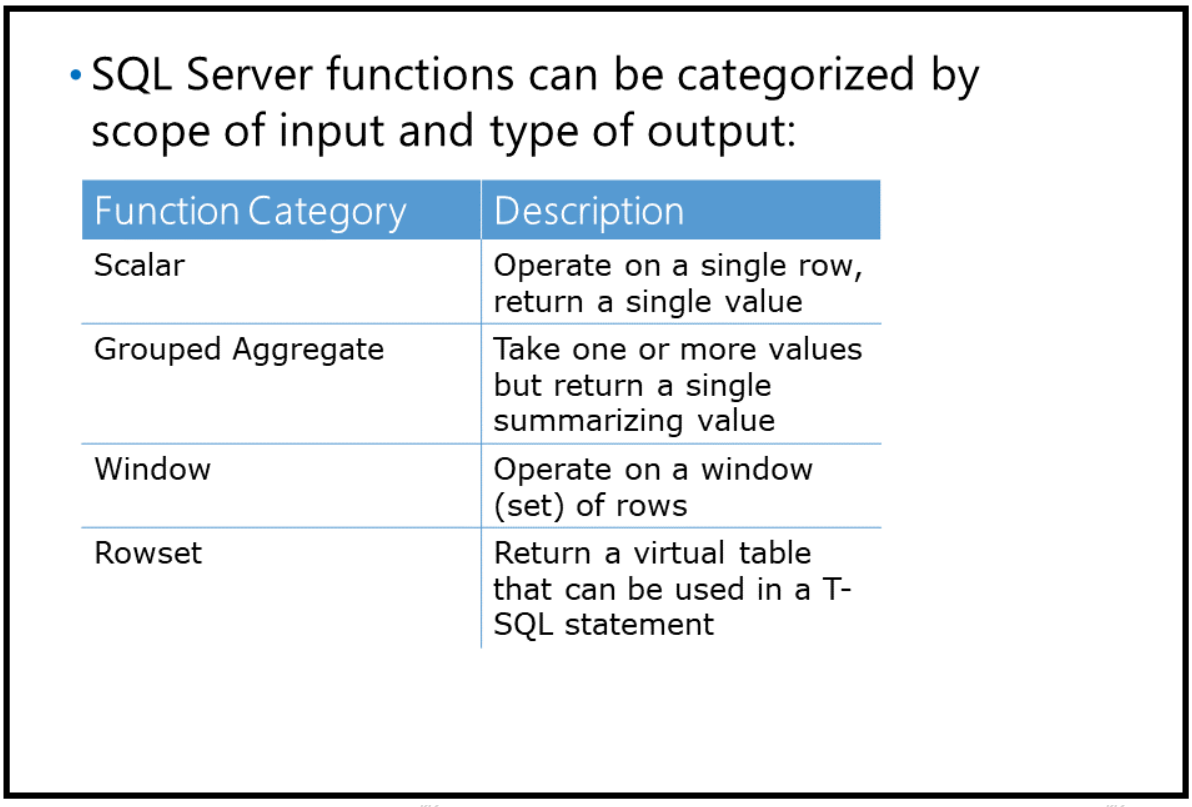
In this lesson, you will learn about SQL Server function types, and then work with scalar functions.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the types of built-in functions provided by SQL Server.
- Write queries using scalar functions.
- Describe aggregate, window and rowset functions.

SQL Server Built-in Function Types



Functions built into SQL Server can be categorized as follows:

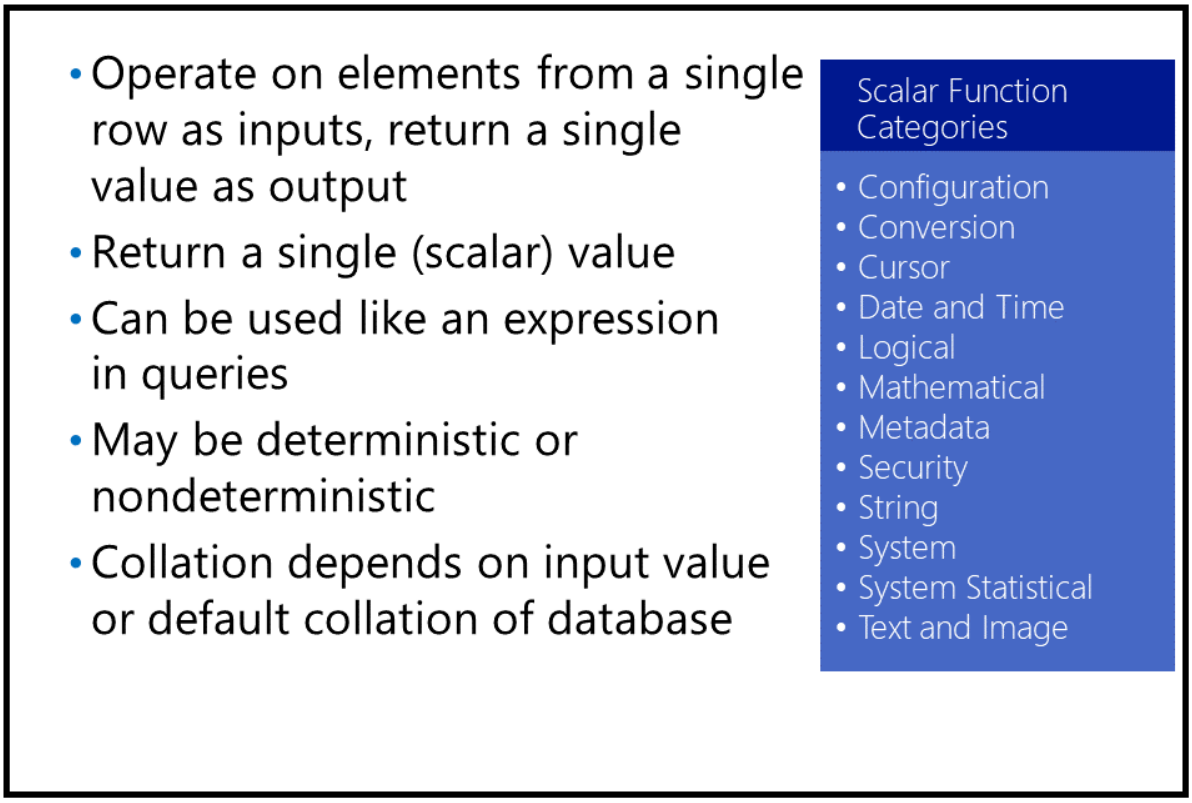
Function Category	Description
Scalar	Operate on a single row, return a single value
Grouped Aggregate	Take one or more input values, return a single summarizing value
Window	Operate on a window (set) of rows

Function Category	Description
Rowset	Return a virtual table that can be used in a T-SQL statement

Note:

- This course will cover aggregates and window functions in later modules.
- Rowset functions are beyond the scope of this course.
- The rest of this module will cover various scalar functions.

Scalar Functions



Scalar functions return a single value. The number of inputs they take may range from zero (such as GETDATE) to one (such as UPPER) to multiple (such as DATEADD). As scalar functions always return a single value, they can be used anywhere a single value (the result) could exist in its own right—from SELECT clauses to WHERE clause predicates.

Built-in scalar functions can be organized into many categories, such as string, conversion, logical, mathematical, and others. This lesson will look at a few common scalar functions.

Some considerations when using scalar functions include:

- **Determinism**: Will the function return the same value for the same input and database state each time? Many built-in functions are nondeterministic, and as such, their results cannot be indexed. This will have an impact on the query processor's ability to use an index when executing the query.
- **Collation**: When using functions that manipulate character data, which collation will be used? Some functions

use the collation of the input value; others use the collation of the database if no input collation is supplied.

At the time of writing, the SQL Server Technical Documentation listed more than 200 scalar functions. This course is not intended to provide a complete guide to all functions. The following list provides some representative examples:

- Date and time functions (covered previously in this course).
- Mathematical functions.
- Conversion functions (covered later in this module).
- System metadata functions.
- System functions.
- Text and image functions.

Scalar Function in a Select Clause

```
SELECT orderid, orderdate, YEAR(orderdate) AS orderyear
FROM Sales.Orders;
```

The results:

```
orderid     orderdate               orderyear
----------- ----------------------- -----------
10248       2006-07-04 00:00:00.000 2006
10249       2006-07-05 00:00:00.000 2006
10250       2006-07-08 00:00:00.000 2006
```

Returning an Absolute Value

```
SELECT ABS(-1.0), ABS(0.0), ABS(1.0);
```

The results:

```
--- ---  ---
1.0 0.0  1.0
```

Metadata Function

```
Select DB_NAME() AS current_database
```

The results:

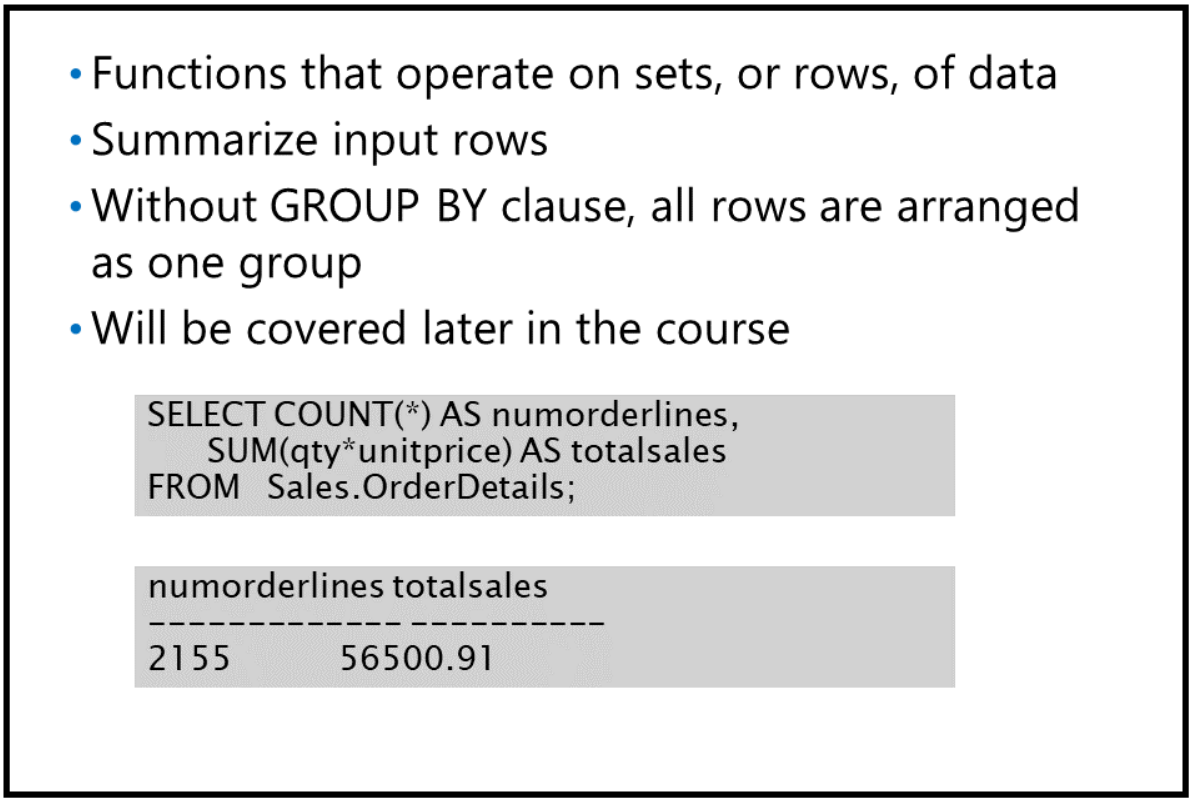
```
Current_database
-------------------
TSQL
```


For additional information about scalar functions and categories, see the SQL Server Technical Documentation:

Built-in Functions (Transact SQL)

http://aka.ms/oor5qi

Aggregate Functions



Grouped aggregate functions operate on sets of rows defined in a GROUP BY clause and return a summarized result. Examples include SUM, MIN, MAX COUNT, and AVG. In the absence of a GROUP BY clause, all rows are considered one set; aggregation is performed on all of them.

Aggregate Function

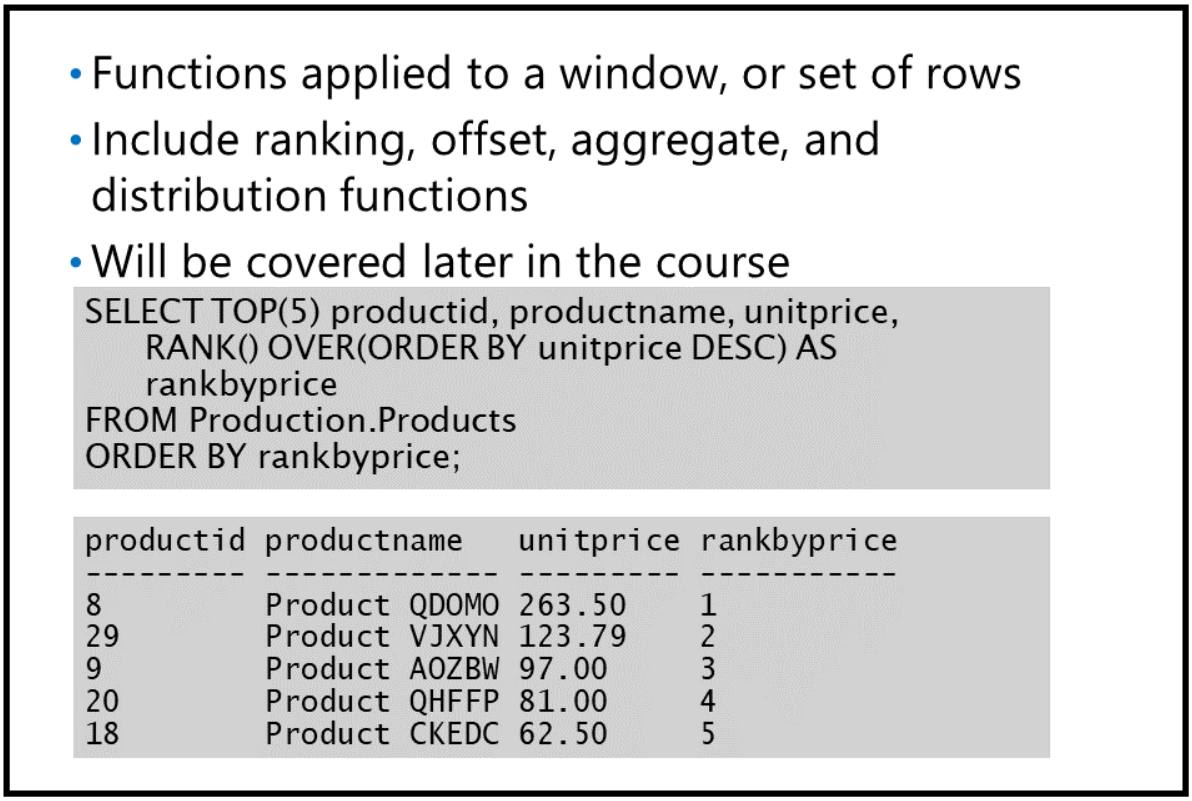
```
SELECT COUNT(*) AS numorders, SUM(unitprice) AS totalsales
FROMSales.OrderDetails;
```

The Results:

```
numorders   totalsales
----------- -----------
2155        56500.91
```

Note: Grouped aggregate functions and the GROUP BY clause will be covered in a later module.

Window Functions



Window functions allow you to perform calculations against a user-defined set, or window, of rows. They include ranking, offset, aggregate, and distribution functions. Windows are defined using the OVER clause, then window functions are applied to the sets defined.

Window Function

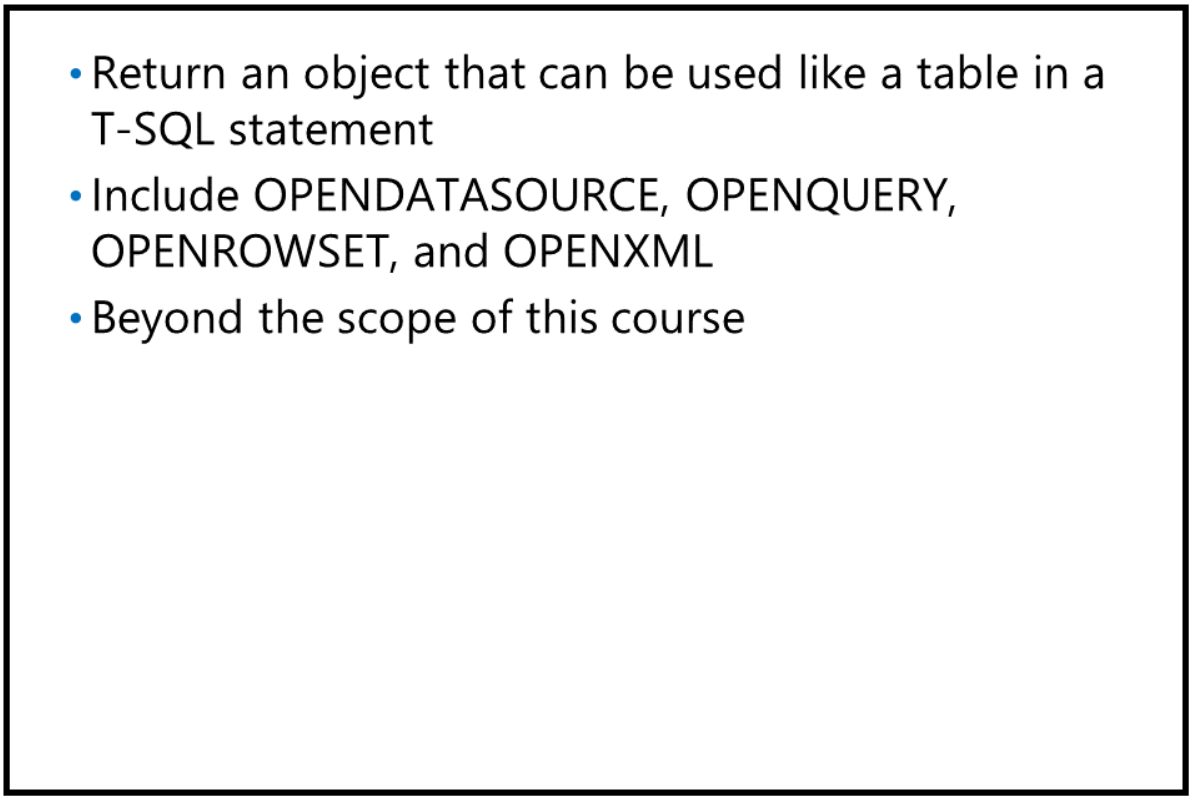
```
SELECT TOP(5) productid, productname, unitprice,
        RANK() OVER(ORDER BY unitprice DESC) AS rankbyprice
FROM Production.Products
ORDER BY rankbyprice;
```

The results:

```
productid   productname   unitprice  rankbyprice
----------- ------------- ---------- -------------
38          Product QDOMO 263.50     1
29          Product VJXYN 123.79     2
9           Product AOZBW 97.00      3
20          Product QHFFO 81.00      4
18          Product CKEDC 62.50      5
```

Note: Window functions will be covered later in this course. This example is provided for illustration only.

Rowset Functions



Rowset functions return a virtual table that can be used elsewhere in the query and take parameters specific to the rowset function itself. They include OPENDATASOURCE, OPENQUERY, OPENROWSET, and OPENXML.

For example, the OPENQUERY function enables you to pass a query to a linked server. It takes the system name of the linked server and the query expression as parameters. The results of the query are returned as a rowset, or virtual table, to the query containing the OPENQUERY function.

Further discussion of rowset functions is beyond the scope of this course. For more information, see Microsoft Docs:

Rowset Functions (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402746

Demonstration: Writing Queries Using Built-in Functions

In this demonstration, you will see how to use build-in scalar functions.

Demonstration Steps

Use Built-in Scalar Functions

1. Ensure that the **20761C-MIA-DC** and **20761C-MIA-SQL** virtual machines are both running, and then log on to **20761C-MIA-SQL** as **ADVENTUREWORKS\Student** with the password **Pa55w.rd**.
2. Run **D:\Demofiles\Mod08\Setup.cmd** as an administrator.
3. In the **User Account Control** dialog box, click **Yes**.
4. At the command prompt, type **y**, and then press Enter.
5. When the script has finished, press Enter.
6. Start SQL Server Management Studio and connect to the **MIA-SQL** database engine instance using Windows authentication.
7. Open the **Demo.ssmssln** solution in the **D:\Demofiles\Mod08\Demo** folder.
8. In Solution Explorer, expand **Queries**, and then double-click **11 - Demonstration A.sql**.
9. Select the code under the comment **Step 1**, and then click **Execute**.
10. Select the code under the comment **Step 2**, and then click **Execute**.
11. Select the code under the comment **Step 3**, and then click **Execute**.
12. Select the code under the comment **Step 4**, and then click **Execute**.
13. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Categorize Activity

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

Scalar Functions

- GETDATE()
- UPPER()
- DATEADD()

DB_NAME()

YEAR()

ABS()

Aggregate Functions

MAX()

COUNT()

MIN()

AVG()

SUM()

Rowset Functions

OPENXML()

OPENROWSET()

OPENQUERY()

OPENDATASOURCE()

Check answer Show solution Reset

Correct

Lesson 2: Using Conversion Functions

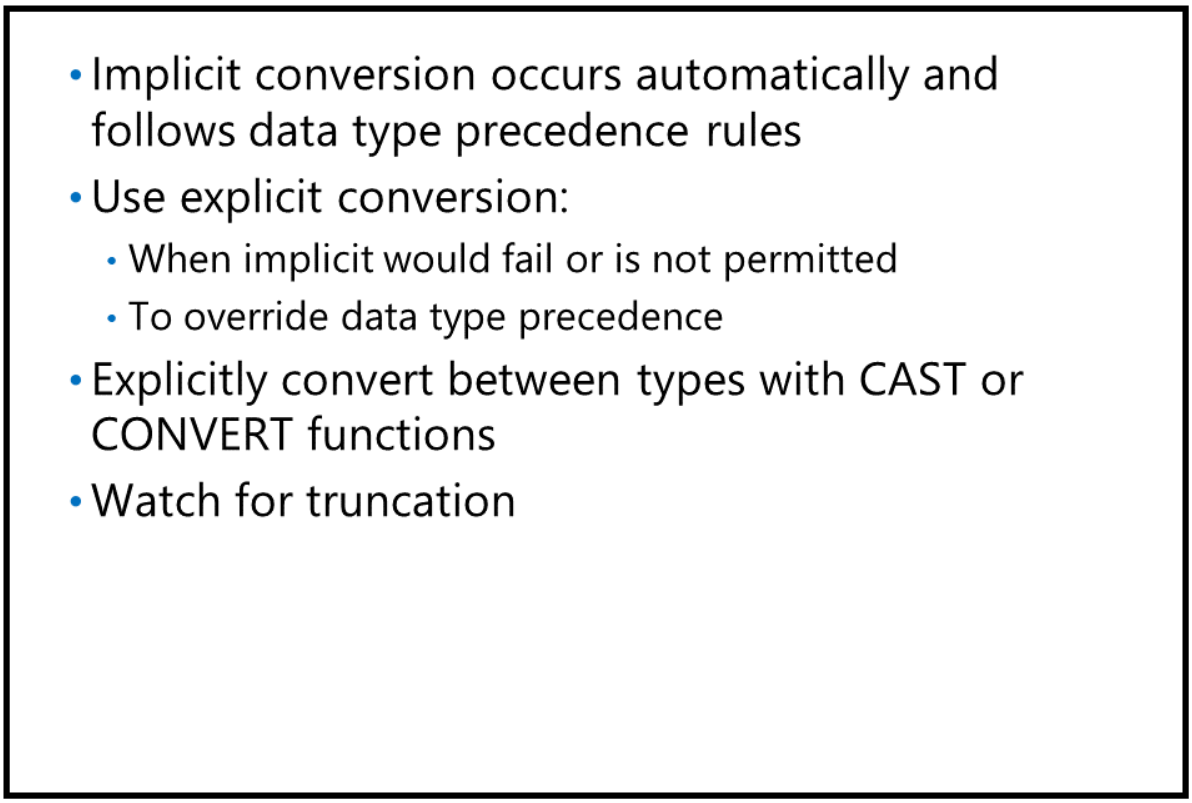
When writing T-SQL queries, it's very common to need to convert data between data types. Sometimes the conversion happens automatically; sometimes you need to control it. In this lesson, you will learn how to explicitly convert data between types using several SQL Server functions. You will also learn to work with functions in SQL Server that provide additional flexibility during conversion.

Lesson Objectives

After completing this lesson, you will be able to:

- Describe the difference between implicit and explicit conversions.
- Describe when you will need to use explicit conversions.
- Explicitly convert between data types using the CAST and CONVERT functions.
- Convert strings to date and numbers with the PARSE, TRY_PARSE, and TRY_CONVERT functions.

Implicit and Explicit Data Type Conversions



Earlier in this course, you learned that there are scenarios when data types may be converted during SQL Server operations. You learned that SQL Server may implicitly convert data types, following the precedence rules for type conversion. However, you might need to override the type precedence, or force a conversion where an implicit conversion might fail.

To accomplish this, you can use the CAST and CONVERT functions, in addition to the TRY_CONVERT function.

Some considerations when converting between data types include:

- **Collation**. When CAST or CONVERT returns a character string from a character string input, the output uses the same collation. When converting from a noncharacter type to a character, the return value uses the collation of the database. The COLLATE option may be used with CAST or CONVERT to override this behavior.
- **Truncation**. When you convert data between character or binary types and different data types, data may be truncated, it might appear cut off, or an error could be thrown because the result is too short to display. The end

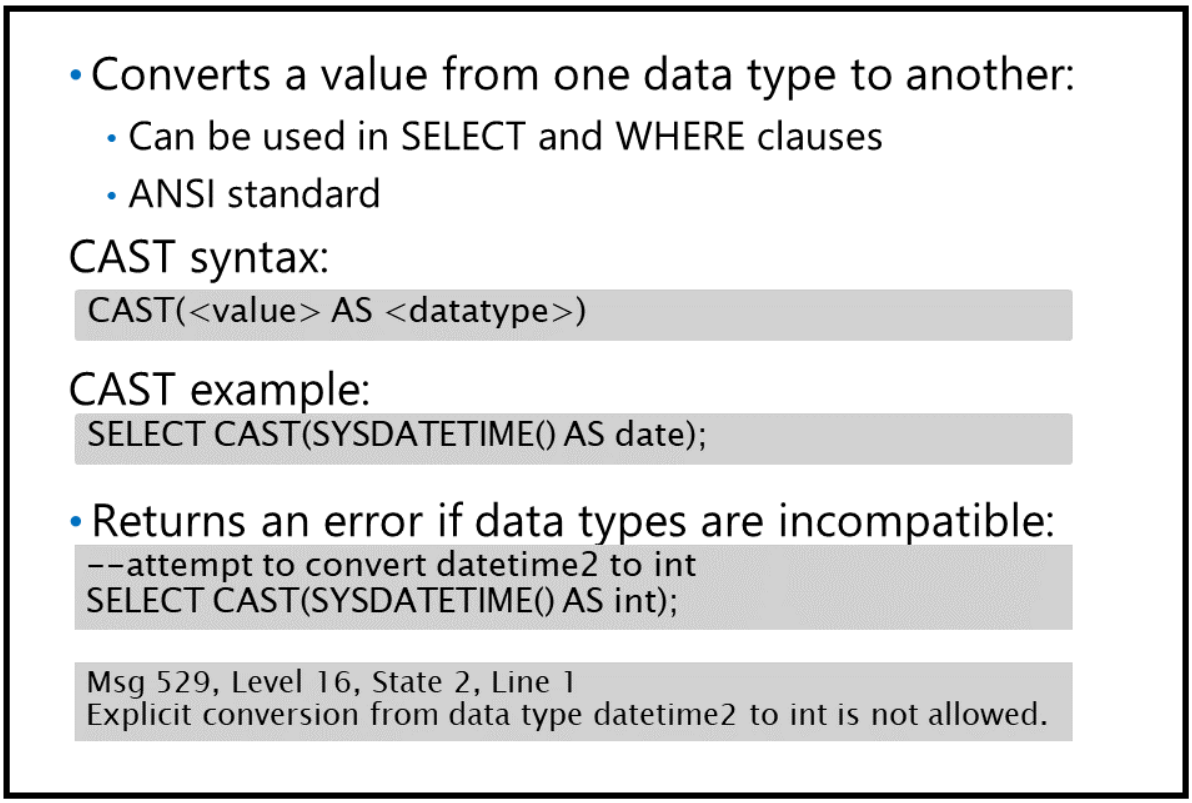
result depends on the data types involved. For example, conversion from an integer with a two-digit value to a char(1) will return an "*" which means the character type was too small to display the results.

For additional reading about truncation behavior, see Microsoft Docs:

CAST and CONVERT (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402747

Converting with CAST



To convert a value from one data type to another, SQL Server provides the CAST function. CAST is an ANSI-standard function and is therefore recommended over the SQL Server-specific CONVERT function, which you will learn about in the next topic.

As CAST is a scalar function, you may use it in SELECT and WHERE clauses.

Converting with CAST

```
CAST(<value> AS <datatype>)
```

CAST Example

```
SELECT orderid, orderdate AS order_datetime, CAST(orderdate AS DATE) AS order_date
FROM Sales.Orders;
```

The results:

```
orderid     order_datetime          order_date
----------- ----------------------- ----------
10248       2006-07-04 00:00:00.000 2006-07-04
10249       2006-07-05 00:00:00.000 2006-07-05
10250       2006-07-08 00:00:00.000 2006-07-08
```

CAST With Incompatible Data Types

```
SELECT CAST(SYSDATETIME() AS int);
```

The results:

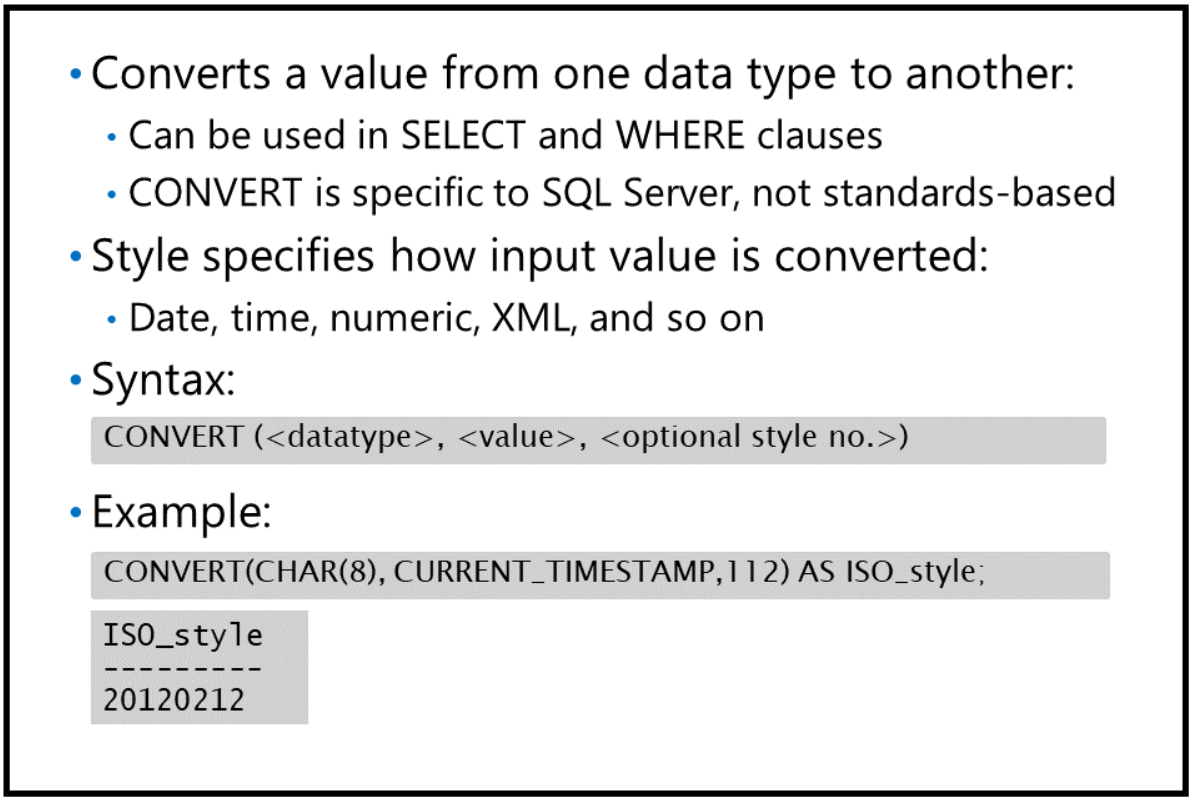
```
Msg 529, Level 16, State 2, Line 1
Explicit conversion from data type datetime2 to int is not allowed.
```

For more information about CATS, see Microsoft Docs:

CAST and CONVERT (Transact SQL)

http://go.microsoft.com/fwlink/?LinkID=402747

Converting with CONVERT



In addition to CAST, SQL Server provides the CONVERT function. Unlike the ANSI-standard CAST function, the CONVERT function is proprietary to SQL Server and is therefore not recommended. However, because of its additional capability to format the return value, you may occasionally still need to use CONVERT.

As with CAST, CONVERT is a scalar function. You may use CONVERT in SELECT and WHERE clauses.

Converting with CONVERT

```
CONVERT(<datatype>, <value>, <optional_style_number>);
```

The style number argument causes CONVERT to format the return data according to a specified set of options. These cover a wide range of date and time styles, in addition to styles for numeric, XML and binary data. Some date and time examples include:

Style Without Century	Style With Century	Standard Label	Value
1	101	U.S.	mm/dd/yyyy
2	102	ANSI	yy.mm.dd - no change for century
12	112	ISO	yymmdd or yyyymmdd

CONVERT Example

```
SELECT CONVERT(CHAR(8), CURRENT_TIMESTAMP, 12) AS ISO_short, CONVERT(CHAR(8),
CURRENT_TIMESTAMP, 112) AS ISO_long;
```

The results:

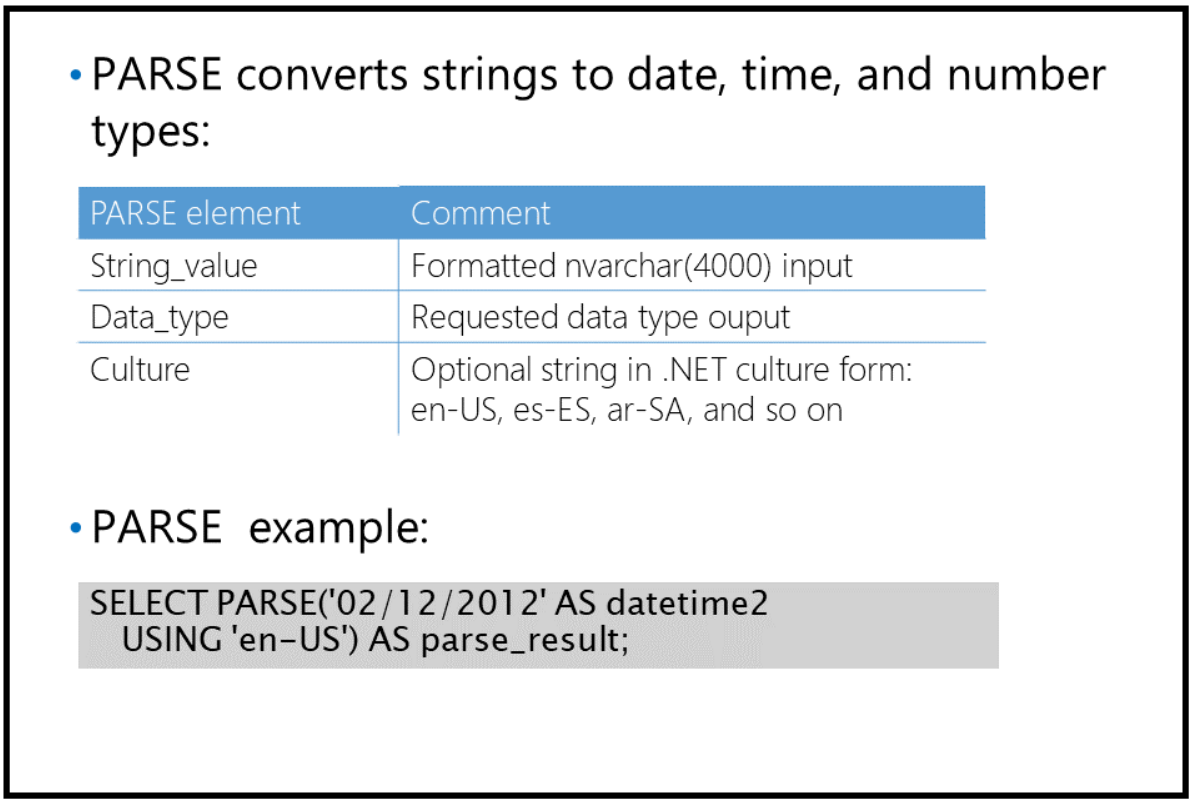
```
ISO_short ISO_long
--------- --------
120212    20120212
```

For more information about CONVERT and its style options, see Microsoft Docs:

CAST and CONVERT (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402747

Converting Strings with PARSE



A very common business problem is building a date, time, or numeric value from one or more strings, often concatenated. SQL Server makes this task easier with the PARSE function.

Converting Strings with PARSE

```
SELECT PARSE('<string_value>',<data_type> [USING <culture_code>]);
```

The culture parameter must be in the form of a valid .NET Framework culture code, such as “**en-US**” for US English, “**es-ES**” for Spanish, and so on. If the culture parameter is omitted, the settings for the current user session will be used.

PARSE Example with Culture Code

```
SELECT PARSE('02/12/2012' AS datetime2 USING 'en-US') AS us_result;
```

The results:

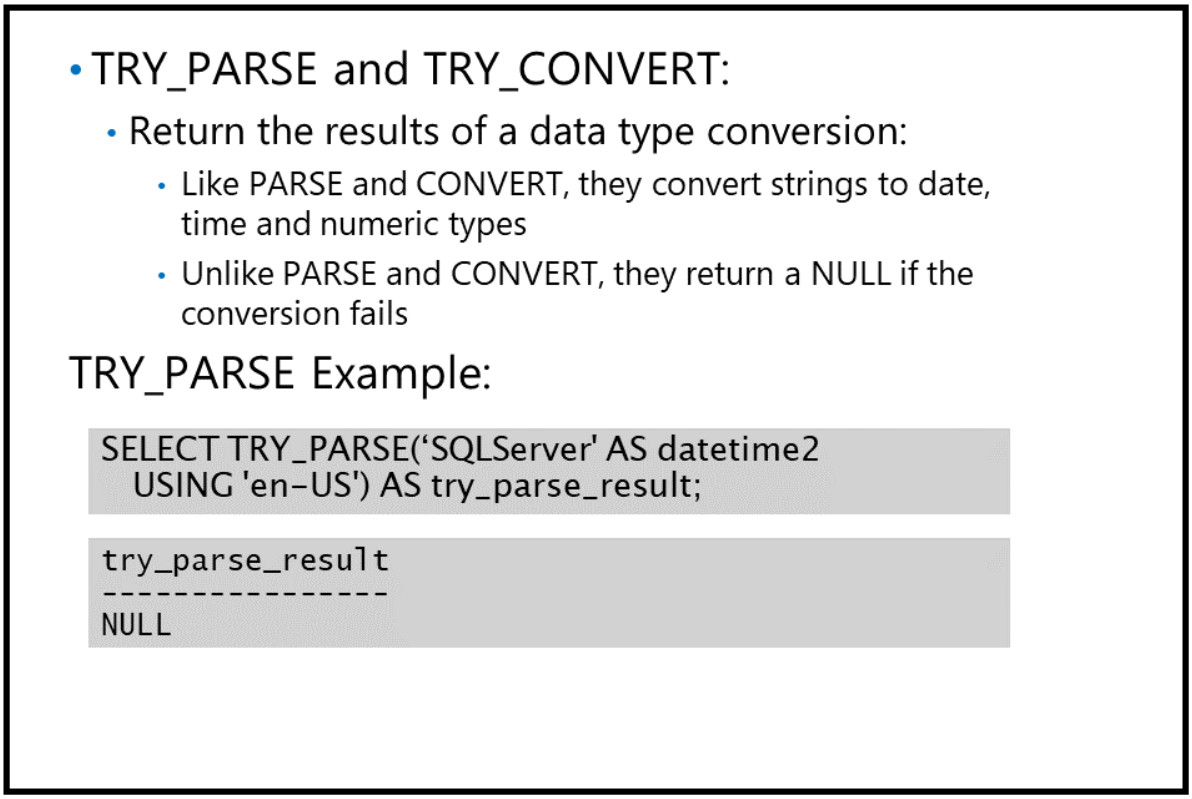
```
us_result
----------------------
2012-02-12 00:00:00.00
```

For more information about PARSE, including culture codes, see Microsoft Docs:

PARSE (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402732

Converting with TRY_PARSE and TRY_CONVERT



When using CONVERT or PARSE, an error may occur if the input value cannot be converted to the specified output type.

Convert Error

```
SELECT CONVERT(datetime2, '20120231');
```

The result:

```
--Msg 241, Level 16, State 1, Line 1
--Conversion failed when converting date and/or time from character string.
```

SQL Server provides conversion functions to address this. TRY_PARSE and TRY_CONVERT will attempt a conversion, just like PARSE and CONVERT, respectively. However, instead of raising a runtime error, failed conversions return NULL.

PARSE Error

```
SELECT PARSE('20120231' AS datetime2 USING 'en-US')
```

Returns:

```
------------------------
NULL
```

Demonstration: Using Conversion Functions

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

Demonstration Steps

Use Functions to Convert Data

1. In Solution Explorer, open the **21 - Demonstration B.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**. Note the error message.
5. Select the code under the comment **Step 4a**, and then click **Execute**.
6. Select the code under the comment **Step 4b**, and then click **Execute**. Note the error message.
7. Select the code under the comment **Step 5**, and then click **Execute**.
8. Select 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. Select the code under the comment **Step 8a**, and then click **Execute**. Note the error message.
11. Select the code under the comment **Step 8b**, and then click **Execute**.
12. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Discovery

You are writing a query against a Human Resources database. You want to ensure that the Employee.StartDate values are displayed in standard British form. What function should you use?

Show solution Reset

PARSE() or TRY_PARSE.

Lesson 3: Using Logical Functions

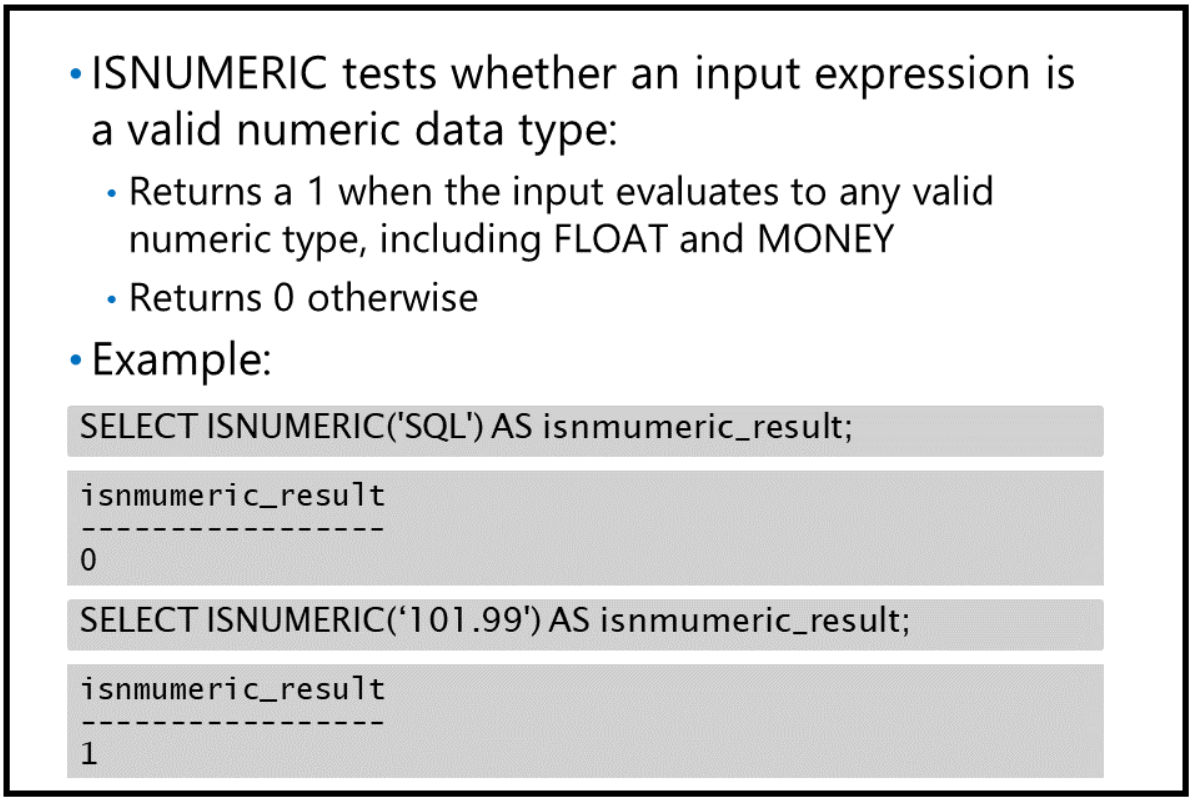
So far in this module, you have learned how to use built-in scalar functions to perform data conversions. In this lesson, you will learn how to use logical functions that evaluate an expression and return a scalar result.

Lesson Objectives

After completing this lesson, you will be able to:

- Use T-SQL functions to perform logical functions.
- Perform conditional tests with the IIF function.
- Select items from a list with CHOOSE.

Writing Logical Test with Functions



A useful function for validating the data type of an expression is ISNUMERIC. This tests an input expression and returns a 1 if the expression is convertible to any numeric type, including integers, decimals, money, floating point, and real. If the value is not convertible to a numeric type, ISNUMERIC returns a 0.

Writing Logical Tests with Functions

```
SELECT empid, lastname, postalcode
FROM HR.Employees
WHERE ISNUMERIC(postalcode)=1;
```

The results:

```
empid       lastname             postalcode
----------- -------------------- ----------
1           Davis                10003
2           Funk                 10001
3           Lew                  10007
4           Peled                10009
5           Buck                 10004
6           Suurs                10005
7           King                 10002
8           Cameron              10006
9           Dolgopyatova         10008
```

Question: How might you use ISNUMERIC when testing data quality?

Performing Conditional Tests with IIF

- IIF returns one of two values, depending on a logical test
- Shorthand for a two-outcome CASE expression:

IIF Element	Comments
Boolean_expression	Logical test evaluating to TRUE, FALSE, or UNKNOWN
True_value	Value returned if expression evaluates to TRUE
False_value	Value returned if expression evaluates to FALSE or UNKNOWN

- IIF example:

```
SELECT    productid, unitprice,
          IIF(unitprice > 50, 'high','low') AS pricepoint
FROM Production.Products;
```

IIF is a logical function in SQL Server. If you have used Visual Basic for Applications in Microsoft Excel®, used Microsoft Access®, or created expressions in SQL Server Reporting Services, you may have used IIF.

IIF Syntax

```
SELECT IIF(<Boolean expression>,<value_if_TRUE>,<value_if_FALSE_or_UNKNOWN);
```

You can think of IIF as a shorthand approach to writing a CASE statement with two possible return values. As with CASE, you may nest an IIF function within another IIF, down to a maximum level of 10.

IIF Example

```
SELECT productid, unitprice,
        IIF(unitprice > 50, 'high','low') AS pricepoint
FROM Production.Products;
```

Returns:

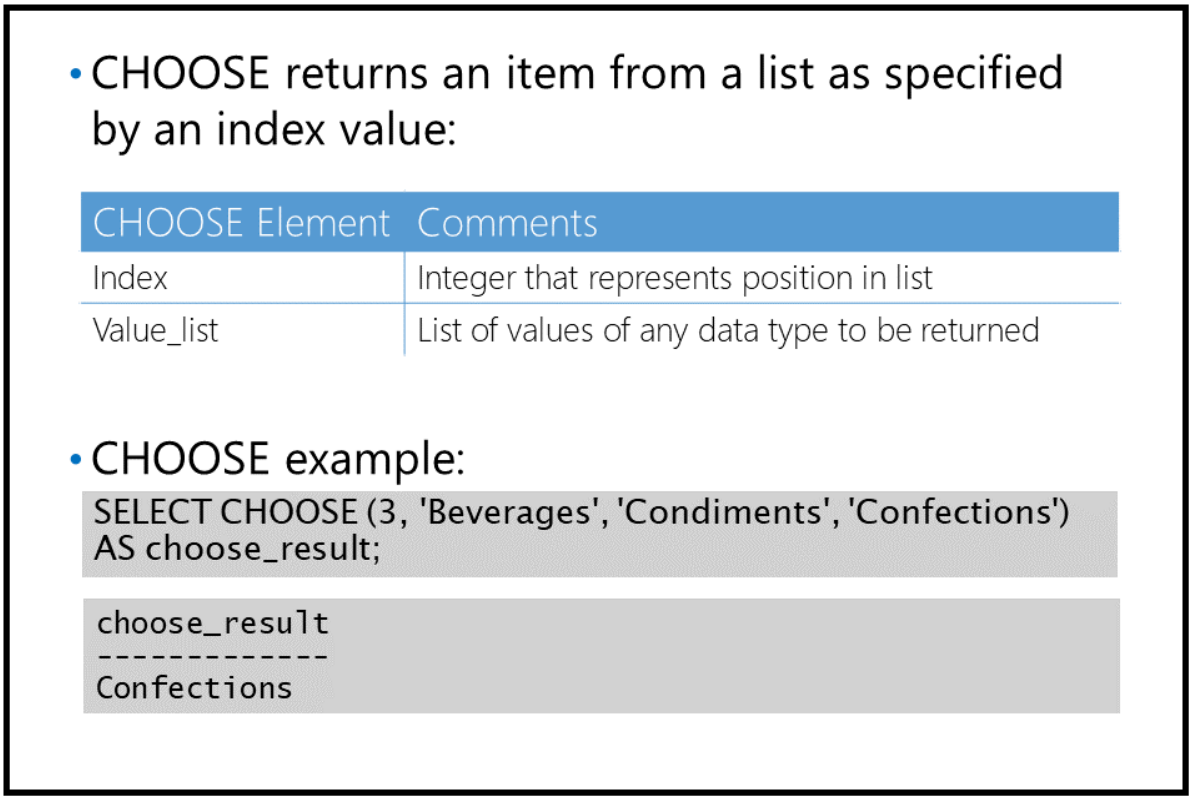
```
productid   unitprice             pricepoint
----------- --------------------- ----------
7           30.00                 low
8           40.00                 low
9           97.00                 high
17          39.00                 low
18          62.50                 high
```

To learn more about this logical function, see *IIF (Transact-SQL)* in Microsoft Docs:

IIF (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402748

Selecting Items from a List with CHOOSE



CHOOSE returns the value of an item at a specific index in a list.

CHOOSE Syntax

```
SELECT CHOOSE(<index_value>,<item1>, <item2>[,...]);
```

CHOOSE Example

```
SELECT CHOOSE (3, 'Beverages', 'Condiments', 'Confections') AS choose_result;
```

Returns:

```
choose_result
-------------
Confections
```

> **Note:** If the index value supplied to CHOOSE does not correspond to a value in the list, CHOOSE will return a NULL.

CHOOSE (Transact-SQL)

http://aka.ms/kt4v4m

Demonstration: Using Logical Functions

In this demonstration, you will see how to use logical functions.

Demonstration Steps

Using Logical Functions

1. In Solution Explorer, open the **31 - Demonstration C.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4**, and then click **Execute**.
6. Select the code under the comment **Step 5**, and then click **Execute**.
7. Keep SQL Server Management Studio open for the next demonstration.

Check Your Knowledge

Discovery

You have the following query:
SELECT e.FirstName, e.LastName, e.FirstAider
FROM Employees AS e
The FirstAider column contains ones and zeros. How can you change the query to make the results more readable?

Show solution Reset

Use an IIF function:

```
SELECT e.FirstName, e.LastName,
  IIF(e.FirstAider = 1, 'Can administer First Aid','No First Aid Skills')
FROM Employees AS e
```

This will output descriptive text instead of ones and zeros.

Lesson 4: Using Functions to Work with NULL

You will often need to take special steps to deal with NULL. Earlier in this module, you learned how to test for NULL with ISNULL. In this module, you will learn additional functions for working with NULL.

Lesson Objectives

After completing this lesson, you will be able to:

- Use ISNULL to replace NULLs.
- Use the COALESCE function to return non-NULL values.
- Use the NULLIF function to return NULL if values match.

Converting NULL with ISNULL

- ISNULL replaces NULL with a specified value
- Not standard; use COALESCE instead
- Syntax:

ISNULL Element	Comment
expression_to_check	Return expression itself if not NULL
replacement_value	Returned if expression evaluates to NULL

- ISNULL example:

```
SELECT custid, city, ISNULL(region, 'N/A') AS region, country
FROM Sales.Customers;
```

```
custid      city            region          country
----------- --------------- --------------- ---------------
7           Strasbourg      N/A             France
9           Marseille       N/A             France
32          Eugene          OR              USA
43          Walla Walla     WA              USA
45          San Francisco   CA              USA
```

In addition to data type conversions, SQL Server provides functions for conversion or replacement of NULL. Both COALESCE and ISNULL can replace NULL input with another value.

To use ISNULL, supply an expression to check for NULL and a replacement value, as in the following example, using the TSQL sample database:

Converting NULL with ISNULL

```
SELECT custid, city, ISNULL(region, 'N/A') AS region, country
FROM Sales.Customers;
```

The result:

```
custid      city            region          country
----------- --------------- --------------- ---------------
40          Versailles      N/A             France
41          Toulouse        N/A             France
43          Walla Walla     WA              USA
45          San Francisco   CA              USA
```

Note: ISNULL is not standard; use COALESCE instead. COALESCE will be covered later in this module.

ISNULL (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402750

Using COALESCE to Return Non-NULL Values

- COALESCE returns the first non-NULL value in a list:
  - With only two arguments, COALESCE behaves like ISNULL
  - If all arguments are NULL, COALESCE returns NULL
- COALESCE is standards-based
- COALESCE example:

```
SELECT custid, country, region, city,
    country + ',' + COALESCE(region, ' ') + ', ' + city as location
FROM Sales.Customers;
```

```
custid country region city        location
------ ------- ------ ----------- --------------------
-
17     Germany NULL   Aachen      Germany, , Aachen
65     USA     NM     Albuquerque USA,NM, Albuquerque
55     USA     AK     Anchorage   USA,AK, Anchorage
83     Denmark NULL   Århus       Denmark, , Århus
```

Earlier in this module, you learned how to use the ISNULL function to test for NULL. Since ISNULL is not ANSI standard, you may wish to use the COALESCE function instead. COALESCE takes as its input one or more expressions, and returns the first non-NULL argument it finds.

With only two arguments, COALESCE behaves like ISNULL. However, with more than two arguments, COALESCE can be used as an alternative to a multipart CASE expression using ISNULL.

If all arguments are NULL, COALESCE returns NULL.

COALESCE Syntax

```
SELECT COALESCE(<expression_1>[, ...<expression_n>];
```

COALESCE Example

```
Code Example Content
SELECT          custid, country, region, city,
                country + ',' + COALESCE(region, ' ') + ', ' + city as location
FROM Sales.Customers;
```

Returns:

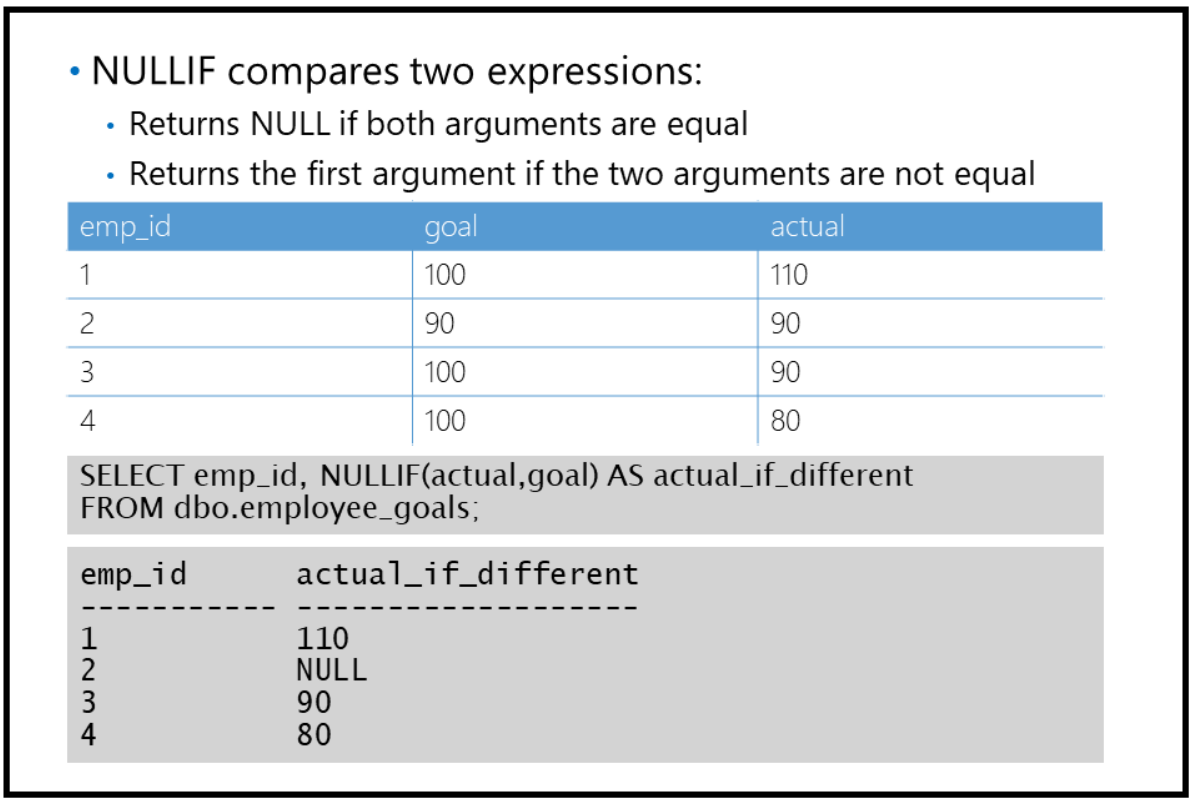
```
custid country region city        location
------ ------- ------ ----------- ----------------------
17     Germany  NULL   Aachen      Germany, , Aachen
65     USA      NM     Albuquerque USA,NM, Albuquerque
55     USA      AK     Anchorage   USA,AK, Anchorage
83     Denmark  NULL   Århus       Denmark, , Århus
```

For more information on COALESCE and comparisons to ISNULL, see Microsoft Docs:

COALESCE (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402751

Using NULLIF to Return NULL If Values Match



In this module, the NULLIF function is the first you will learn that is designed to return NULL, if its condition is met. NULLIF returns NULL when two arguments match. This has useful applications in areas such as data cleansing, when you wish to replace blank or placeholder characters with NULL.

NULLIF takes two arguments and returns NULL if they both match. If they are not equal, NULLIF returns the first argument.

NULLIF Example

```
SELECT empid, lastname, firstname, NULLIF(middleinitial,' ') AS middle_initial
FROM HR.Employees;
```

Returns:

```
empid       lastname            firstname  middle_initial
----------- ------------------- ---------- --------------
1           Davis               Sara       NULL
2           Funk                Don        D
3           Lew                 Judy       NULL
4           Peled               Yael       Y
```

> **Note:** This example is provided for illustration only and will not run against the sample database supplied with this course.

For more information, see *NULLIF (Transact-SQL)* in the SQL Server Technical Documentation:

NULLIF (Transact-SQL)

http://go.microsoft.com/fwlink/?LinkID=402752

Demonstration: Using Functions to Work with NULL

In this demonstration, you will see how to use functions to work with NULL.

Demonstration Steps

Use Functions to Work with NULL

1. In Solution Explorer, open the **41 - Demonstration D.sql** script file.
2. Select the code under the comment **Step 1**, and then click **Execute**.
3. Select the code under the comment **Step 2**, and then click **Execute**.
4. Select the code under the comment **Step 3**, and then click **Execute**.
5. Select the code under the comment **Step 4a**, and then click **Execute**.
6. Select the code under the comment **Step 4b**, and then click **Execute**.
7. Select the code under the comment **Step 4c**, and then click **Execute**.
8. Select the code under the comment **Step 4d**, and then click **Execute**.
9. Select the code under the comment **Step 5**, and then click **Execute**.
10. Close SQL Server Management Studio without saving any files.

Check Your Knowledge

Select the best answer

You are writing a query against the Employees table in the Human Resources database. The CurrentStatus column can contain the string values “New”, “Retired”, and “Under Caution”. Many employees have this column set to NULL when those statuses do not apply to them. For confidentiality, you want to ensure that the employees currently under caution are displayed like those employees with no applicable status. What function should you use?

ISNULL()

COALESCE()

NULLIF()

TRY_PARSE()

PARSE()

Check answer Show solution Reset

You can use NULLIF(Employees.CurrentStatus, 'Under Caution') to display the employees currently under caution; for example, employees with no applicable status.

Lab: Using Built-in Functions

Scenario

You are an Adventure Works business analyst, who will be writing reports using corporate databases stored in SQL Server. You have been provided with 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 the data, convert it, and then check for missing values.

Objectives

After completing this lab, you will be able to:

- Write queries that include conversion functions.
- Write queries that use logical functions.
- Write queries that test for nullability.

Lab Setup

Estimated Time: 40 minutes

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

User name: **ADVENTUREWORKS\Student**

Password: **Pa55w.rd**

Exercise 1: Writing Queries That Use Conversion Functions

Scenario

You have been asked to write the following reports for these departments:

1. **Sales**. The product name and unit price for each product within an easy to read string.
2. **Marketing**. The order id, order date, shipping date, and shipping region for each order after 4/1/2007.
3. **IT**. Convert all Sales phone number information into integers.

The main tasks for this exercise are as follows:

1. Prepare the Lab Environment
2. Write a SELECT Statement that Uses the CAST or CONVERT Function
3. Write a SELECT Statement to Filter Rows Based on Specific Date Information
4. Write a SELECT Statement to Convert the Phone Number Information to an Integer Value

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\Lab08\Starter** folder as Administrator.

Detailed Steps ▲

Task 2: Write a SELECT Statement that Uses the CAST or CONVERT Function

1. Open the project file **D:\Labfiles\Lab08\Starter\Project\Project.ssmssln** and the T-SQL script **51 - Lab Exercise 1.sql**. Ensure that you are connected to the **TSQL** database.
2. Write a SELECT statement against the **Production.Products** table to retrieve a calculated column named **productdesc**. The calculated column should be based on the **productname** and **unitprice** columns and look like this:

```
Results: The unit price for the Product HHYDP is 18.00 $.
```

3. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab08\Solution\52 - Lab Exercise 1 - Task 1 Result.txt.
4. Did you use the CAST or the CONVERT function? Which one do you think is more appropriate to use?

Detailed Steps ▲

Task 3: Write a SELECT Statement to Filter Rows Based on Specific Date Information

1. The US marketing department has supplied you with a start date of "**4/1/2007**" (using US English form, read as "**April 1, 2007**") and an end date of "**11/30/2007**" (using US English form, read as "**November 30, 2007**").
2. Write a SELECT statement against the **Sales.Orders** table to retrieve the **orderid**, **orderdate**, **shippeddate**, and **shipregion** columns. Filter the result to include only rows with the order date between the specified start date and end date, and have more than 30 days between the shipped date and order date. Also check the **shipregion** column for missing values. If there is a missing value, then return the value "**No region**".
3. In this SELECT statement, you can use the CONVERT function with a style parameter or the PARSE function.
4. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab08\Solution\53 - Lab Exercise 1 - Task 2 Result.txt.

Detailed Steps ▲

Task 4: Write a SELECT Statement to Convert the Phone Number Information to an Integer Value

1. The IT department would like to convert all the information about phone numbers in the **Sales.Customers** table to integer values. The IT staff indicated that all hyphens, parentheses, and spaces have to be removed before the conversion to an integer data type.
2. Write a SELECT statement to implement the requirement of the IT department. Replace all the specified characters in the phone column of the **Sales.Customers** table, and then convert the column from the nvarchar datatype to the int datatype. The T-SQL statement must not fail if there is a conversion error—it should return a NULL. (Hint: first try writing a T-SQL statement using the CONVERT function, and then compare it with the TRY_CONVERT function.) Use the alias **phoneasint** for this calculated column.
3. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab08\Solution\54 - Lab Exercise 3 - Task 3 Result.txt.

Result: After this exercise, you should be able to use conversion functions.

Exercise 2: Writing Queries That Use Logical Functions

Scenario

The sales department would like to have different reports regarding the segmentation of customers and specific order lines. You will add a new calculated column to show the target group for the segmentation.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement to Mark Specific Customers Based on Their Country and Contact Title
2. Modify the T-SQL Statement to Mark Different Customers
3. Create Four Groups of Customers

Detailed Steps ▲

Task 1: Write a SELECT Statement to Mark Specific Customers Based on Their Country and Contact Title

1. Open the T-SQL script 61 - Lab Exercise 2.sql. Ensure that you are connected to the **TSQL** database.
2. Write a SELECT statement against the **Sales.Customers** table and retrieve the **custid** and **contactname** columns. Add a calculated column named **segmentgroup**, using a logical function IIF with the value "**Target group**" for customers that are from Mexico and have the value "**Owner**" in the contact title. Use the value "**Other**" for the rest of the customers.
3. Execute the written statement and compare the results that you achieved with the desired results shown in the file D:\Labfiles\Lab08\Solution\62 - Lab Exercise 2 - Task 1 Result.txt.

Detailed Steps ▲

Task 2: Modify the T-SQL Statement to Mark Different Customers

1. Modify the T-SQL statement from task 1 to change the calculated column to show the value "**Target group**" for all customers without a missing value in the region attribute or with the value "**Owner**" in the contact title attribute.
2. Execute the written statement and compare the results that you achieved with the recommended result shown in the file D:\Labfiles\Lab08\Solution\63 - Lab Exercise 2 - Task 2 Result.txt.

Detailed Steps ▲

Task 3: Create Four Groups of Customers

1. Write a SELECT statement against the **Sales.Customers** table and retrieve the **custid** and **contactname** columns. Add a calculated column named **segmentgroup** using the logical function CHOOSE with four possible descriptions ("**Group One**", "**Group Two**", "**Group Three**", "**Group Four**"). Use the modulo operator on the column **custid**. (Use the expression **custid % 4 + 1** to determine the target group.)
2. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab08\Solution\64 - Lab Exercise 2 - Task 3 Result.txt.

Result: After this exercise, you should know how to use the logical functions.

Exercise 3: Writing Queries That Test for Nullability

Scenario

The sales department would like to have additional segmentation of customers. Some columns that you should retrieve contain missing values, and you will have to change the NULL to some more meaningful information for the business users.

The main tasks for this exercise are as follows:

1. Write a SELECT Statement to Retrieve the Customer Fax Information
2. Write a Filter for a Variable That Could Be a Null
3. Write a SELECT Statement to Return All the Customers That Do Not Have a Two-Character Abbreviation for the Region

Detailed Steps ▲

Task 1: Write a SELECT Statement to Retrieve the Customer Fax Information

1. Open the T-SQL script 71 - Lab Exercise 3.sql. Ensure that you are connected to the **TSQL** database.
2. Write a SELECT statement to retrieve the **contactname** and fax columns from the **Sales.Customers** table. If there is a missing value in the fax column, return the value "**No information**".
3. Write two solutions, one using the COALESCE function and the other using the ISNULL function.
4. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab08\Solution\72 - Lab Exercise 3 - Task 1 Result.txt.

5. What is the difference between the ISNULL and COALESCE functions?

Detailed Steps ▲

Task 2: Write a Filter for a Variable That Could Be a Null

- Update the provided T-SQL statement with a WHERE clause to filter the region column using the provided variable **@region**, which can have a value or a NULL. Test the solution using both provided variable declaration cases:

```
DECLARE @region AS NVARCHAR(30) = NULL;
SELECT
custid, region
FROM Sales.Customers;

GO

DECLARE @region AS NVARCHAR(30) = N'WA';
SELECT
custid, region
FROM Sales.Customers;
```

Detailed Steps ▲

Task 3: Write a SELECT Statement to Return All the Customers That Do Not Have a Two-Character Abbreviation for the Region

1. Write a SELECT statement to retrieve the **contactname**, **city**, and **region** columns from the **Sales.Customers** table. Return only rows that do not have two characters in the region column, including those with an inapplicable region (where the region is NULL).
2. Execute the written statement and compare the results that you achieved with the recommended results shown in the file D:\Labfiles\Lab08\Solution\73 - Lab Exercise 3 - Task 3 Result.txt. Notice the number of rows returned.

Result: After this exercise, you should have an understanding of how to test for nullability.

Module Review and Takeaways

In this module, you have learned how to:

- Write queries with built-in scalar functions.
- Use conversion functions.
- Use logical functions.
- Use functions that work with NULL.

Best Practice:

- When possible, use standards-based functions, such as CAST or COALESCE, rather than SQL Server-specific functions like NULLIF or CONVERT.
- Consider the impact of functions in a WHERE clause on query performance.

Review Question(s)

Check Your Knowledge

Discovery

Which function should you use to convert from an int to a nchar(8)?

Show solution Reset

CAST(). It is better to use CAST() whenever possible because it is an ANSI-standard function.

Check Your Knowledge

Discovery

Which function will return a NULL, rather than an error message, if it cannot convert a string to a date?

Show solution Reset

TRY_CONVERT().

Check Your Knowledge

Discovery

What is the name for a function that returns a single value?

Show solution Reset

A scalar function.