Module 8

Using Built-In Functions

Session 8

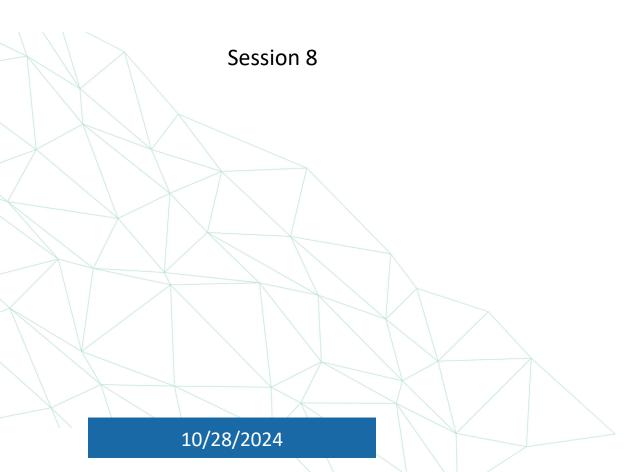# Module Overview

- Writing Queries with Built-In Functions
- Using Conversion Functions
- Using Logical Functions
- Using Functions to Work with NULL

Lesson 1: Writing Queries with Built-In Functions

- SQL Server Built-in Function Types

- Scalar Functions

- Aggregate Functions

- Window Functions

- Rowset Functions

- Demonstration: Writing Queries Using Built-in Functions

SQL Server Built-in Function Types

- SQL Server functions can be categorized by scope of input and type of output:

Function Category	Description
Scalar	Operate on a single row, return a single value
Grouped Aggregate	Take one or more values but return a single summarizing value
Window	Operate on a window (set) of rows
Rowset	Return a virtual table that can be used in a T-SQL statement

Scalar Functions

- Operate on elements from a single row as inputs, return a single value as output

- Return a single (scalar) value

- Can be used like an expression in queries

- May be deterministic or nondeterministic

- Collation depends on input value or default collation of database

Scalar Function Categories

- Configuration
- Conversion
- Cursor
- Date and Time
- Logical
- Mathematical
- Metadata
- Security
- String
- System
- System Statistical
- Text and Image

Aggregate Functions

- Functions that operate on sets, or rows, of data

- Summarize input rows

- Without GROUP BY clause, all rows are arranged as one group

- Will be covered later in the course

```
SELECT COUNT(*) AS numorderlines,
    SUM(qty*unitprice) AS totalsales
FROM   Sales.OrderDetails;
```

```
numorderlines totalsales
-------------- -----------
2155          56500.91
```

Window Functions

- Functions applied to a window, or set of rows
- Include ranking, offset, aggregate, and distribution functions
- Will be covered later in the course

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

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

Rowset Functions

- Return an object that can be used like a table in a T-SQL statement
- Include OPENDATASOURCE, OPENQUERY, OPENROWSET, and OPENXML
- Beyond the scope of this course

Demonstration: Writing Queries Using Built-in Functions

In this demonstration, you will see how to:
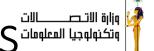
- Use built-in scalar functions

Lesson 2: Using Conversion Functions

- Implicit and Explicit Data Type Conversions
- Converting with CAST
- Converting with CONVERT
- Converting Strings with PARSE
- Converting with TRY_PARSE and TRY_CONVERT
- Demonstration: Using Conversion Functions

Implicit and Explicit Data Type Conversions

- Implicit conversion occurs automatically and follows data type precedence rules
- Use explicit conversion:
  - When implicit would fail or is not permitted
  - To override data type precedence
- Explicitly convert between types with CAST or CONVERT functions
- Watch for truncation

Converting with CAST

- Converts a value from one data type to another:
  - Can be used in SELECT and WHERE clauses
  - ANSI standard

CAST syntax:

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

CAST example:

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

- Returns an error if data types are incompatible:

```
--attempt to convert datetime2 to int
SELECT CAST(SYSDATETIME() AS int);
```

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

Converting with CONVERT

- Converts a value from one data type to another:
  - Can be used in SELECT and WHERE clauses
  - CONVERT is specific to SQL Server, not standards-based
- Style specifies how input value is converted:
  - Date, time, numeric, XML, and so on
- Syntax:

```
CONVERT (<datatype>, <value>, <optional style no.>)
```

- Example:

```
CONVERT(CHAR(8), CURRENT_TIMESTAMP,112) AS ISO_style;
```

```
ISO_style
---------
20120212
```

Converting Strings with PARSE

- PARSE converts strings to date, time, and number types:

PARSE element	Comment
String_value	Formatted nvarchar(4000) input
Data_type	Requested data type ouput
Culture	Optional string in .NET culture form: en-US, es-ES, ar-SA, and so on

- PARSE example:

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

Converting with TRY_PARSE and TRY_CONVERT

- TRY_PARSE and TRY_CONVERT:
  - Return the results of a data type conversion:
    - Like PARSE and CONVERT, they convert strings to date, time and numeric types
    - Unlike PARSE and CONVERT, they return a NULL if the conversion fails

TRY_PARSE Example:

```
SELECT TRY_PARSE('SQLServer' AS datetime2
    USING 'en-US') AS try_parse_result;
```

```
try_parse_result
-----------------
NULL
```

Demonstration: Using Conversion Functions

In this demonstration, you will see how to:

- Use functions to convert data

Lesson 3: Using Logical Functions

- Writing Logical Test with Functions
- Performing Conditional Tests with IIF
- Selecting Items from a List with CHOOSE
- Demonstration: Using Logical Functions

Writing Logical Test with Functions

- ISNUMERIC tests whether an input expression is a valid numeric data type:
  - Returns a 1 when the input evaluates to any valid numeric type, including FLOAT and MONEY
  - Returns 0 otherwise
- Example:

```
SELECT ISNUMERIC('SQL') AS isnmumeric_result;
```

```
isnmumeric_result
------------------
0
```

```
SELECT ISNUMERIC('101.99') AS isnmumeric_result;
```

```
isnmumeric_result
------------------
1
```

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;
```

Selecting Items from a List with CHOOSE

- CHOOSE returns an item from a list as specified by an index value:

CHOOSE Element	Comments
Index	Integer that represents position in list
Value_list	List of values of any data type to be returned

- CHOOSE example:

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

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

Demonstration: Using Logical Functions

In this demonstration, you will see how to:
- Use logical functions

Lesson 4: Using Functions to Work with NULL

- Converting NULL with ISNULL

- Using COALESCE to Return Non-NULL Values

- Using NULLIF to Return NULL If Values Match

- Demonstration: Using Functions to Work with NULL

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
```

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
```

Using NULLIF to Return NULL If Values Match

- NULLIF compares two expressions:
  - Returns NULL if both arguments are equal
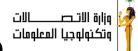 - Returns the first argument if the two arguments are not equal

emp_id	goal	actual
1	100	110
2	90	90
3	100	90
4	100	80

```
SELECT emp_id, NULLIF(actual,goal) AS actual_if_different
FROM dbo.employee_goals;
```

```
emp_id       actual_if_different
-----------  --------------------
1            110
2            NULL
3            90
4            80
```

Demonstration: Using Functions to Work with NULL

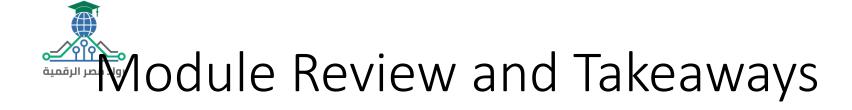In this demonstration, you will see how to:

• Use functions to work with NULL

Lab: Using Built-in Functions

- Exercise 1: Writing Queries That Use Conversion Functions
- Exercise 2: Writing Queries That Use Logical Functions
- Exercise 3: Writing Queries That Test for Nullability

Lab 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.

Module Review and Takeaways

- Review Question(s)
- Best Practice