❖ Functions Aggregate

- **1. AVG** = this function returns the average of the values in a group. It ignores null values.

```
AVG ( [ ALL | DISTINCT ] expression )
  [ OVER ( [ partition_by_clause ] order_by_clause ) ]
```

This statement returns the average list price of products in the AdventureWorks2022 database. Through the use of DISTINCT, the calculation considers only unique values.

```
SELECT AVG(DISTINCT ListPrice)
FROM Production.Product;
```

Here is the result set.

```
------------------------------
437.4042
```

Without DISTINCT, the AVG function finds the average list price of all products in the Product table in the AdventureWorks2022 database, including any duplicate values.

```
SELECT AVG(ListPrice)
FROM Production.Product;
```

Here is the result set.

```
------------------------------
438.6662
```

- **2. COUNT** = This function returns the number of items found in a group. COUNT operates like the COUNT_BIG function. These functions differ only in the data types of their return values. COUNT always returns an **int** data type value. COUNT_BIG always returns a **bigint** data type value.

```
COUNT ( { [ [ ALL | DISTINCT ] expression ] | * } )
```

- ✓ COUNT(*) without GROUP BY returns the cardinality (number of rows) in the resultset. This includes rows comprised of all-NULL values and duplicates.
- ✓ COUNT(*) with GROUP BY returns the number of rows in each group. This includes NULL values and duplicates.
- ✓ COUNT(ALL <expression>) evaluates *expression* for each row in a group, and returns the number of nonnull values.
- ✓ COUNT(DISTINCT *expression*) evaluates *expression* for each row in a group, and returns the number of unique, nonnull values.

This example returns the number of different titles that an Adventure Works Cycles employee can hold.

SQLCopy

```
SELECT COUNT(DISTINCT Title)
FROM HumanResources.Employee;
GO
```

Here is the result set.

```
-----------
67
```

This example returns the total number of Adventure Works Cycles employees.

```
SELECT COUNT(*)
FROM HumanResources.Employee;
GO
```

Here is the result set.

```
-----------
290
```

- **3. MAX** = Returns the maximum value in the expression.

```
MAX( [ ALL | DISTINCT ] expression )
```

MAX ignores any null values.

MAX returns NULL when there is no row to select.

For character columns, MAX finds the highest value in the collating sequence.

The following example returns the highest (maximum) tax rate in the AdventureWorks2022 database.

```
SELECT MAX(TaxRate)
FROM Sales.SalesTaxRate;
GO
```

Here is the result set.

```
-------------------

19.60

Warning, null value eliminated from aggregate.
```

- **4. MIN** = Returns the minimum value in the expression. May be followed by the OVER clause.

```
MIN ( [ ALL | DISTINCT ] expression )
```

MIN ignores any null values.

With character data columns, MIN finds the value that is lowest in the sort sequence.

The following example returns the lowest (minimum) tax rate. The example uses the AdventureWorks2022 database

SQLCopy
```
SELECT MIN(TaxRate)
FROM Sales.SalesTaxRate;
GO
```

Here is the result set.

```
-------------------
5.00
```

- **5. SUM** = Returns the sum of all the values, or only the DISTINCT values, in the expression.

SUM can be used with numeric columns only. Null values are ignored.

```
SUM ( [ ALL | DISTINCT ] expression )
```

The following examples show using the SUM function to return summary data in the AdventureWorks2022 database.

```
SELECT Color, SUM(ListPrice), SUM(StandardCost)
FROM Production.Product
WHERE Color IS NOT NULL
   AND ListPrice != 0.00
   AND Name LIKE 'Mountain%'
GROUP BY Color
ORDER BY Color;
GO
```

Here is the result set.

```
Color
--------------- --------------------- ---------------------
Black          27404.84             5214.9616
Silver        26462.84             14665.6792
White          19.00              6.7926
```

❖ Date&Time

- **6. DATEDIFF** = This function returns the count (as a signed integer value) of the specified datepart boundaries crossed between the specified *startdate* and *enddate*.

See DATEDIFF_BIG (Transact-SQL) for a function that handles larger differences between the *startdate* and *enddate* values. See Date and Time Data Types and Functions (Transact-SQL) for an overview of all Transact-SQL date and time data types and functions.

```
DATEDIFF ( datepart , startdate , enddate )
```

datepart
The units in which DATEDIFF reports the difference between the *startdate* and *enddate*. Commonly used *datepart* units include month or second.

The *datepart* value cannot be specified in a variable, nor as a quoted string like 'month'.

The following table lists all the valid *datepart* values. **DATEDIFF** accepts either the full name of the *datepart*, or any listed abbreviation of the full name.

***datepart* name**	***datepart* abbreviation**
year	y, yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy
day	dd, d
week	wk, ww
weekday	dw, w
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns

startdate
An expression that can resolve to one of the following values:

- date
- datetime
- datetimeoffset
- datetime2
- smalldatetime
- time

Use four-digit years to avoid ambiguity. See Configure the two digit year cutoff Server Configuration Option for information about two-digit year values.

enddate
See *startdate*.

- **7. DATENAME** = This function returns a character string representing the specified *datepart* of the specified *date*.

See Date and Time Data Types and Functions (Transact-SQL) for an overview of all Transact-SQL date and time data types and functions.

```
DATENAME ( datepart , date )
```

Use DATENAME in the following clauses:

- GROUP BY
- HAVING
- ORDER BY
- SELECT <list>
- WHERE

In SQL Server, DATENAME implicitly casts string literals as a **datetime2** type. In other words, DATENAME does not support the format YDM when the date is passed as a string. You must explicitly cast the string to a **datetime** or **smalldatetime** type to use the YDM format.

```
SELECT DATENAME(datepart,'1985-11-19 12:15:00');
```

- **8. DATEPART** =This function returns an integer representing the specified *datepart* of the specified *date*.

DATEPART (datepart , date)

datepart
The specific part of the *date* argument for which DATEPART will return an **integer**. This table lists all valid *datepart* arguments.

datepart	**Abbreviations**
year	yy, yyyy
quarter	qq, q
month	mm, m
dayofyear	dy, y
day	dd, d
week	wk, ww
weekday	dw
hour	hh
minute	mi, n
second	ss, s
millisecond	ms
microsecond	mcs
nanosecond	ns
tzoffset	tz
iso_week	isowk, isoww

date
An expression that resolves to one of the following data types:

- date
- datetime
- datetimeoffset
- datetime2
- smalldatetime
- time

For *date*, DATEPART will accept a column expression, expression, string literal, or user-defined variable. Use four-digit years to avoid ambiguity issues. See Configure the two digit year cutoff Server Configuration Option for information about two-digit years.

Week and weekday datepart arguments

For a **week** (**wk**, **ww**) or **weekday** (**dw**) *datepart*, the DATEPART return value depends on the value set by SET DATEFIRST:

```
SET DATEFIRST { number | @number_var }
```

number | **@***number_var*
Is an integer that indicates the first day of the week. It can be one of the following values.

Value	**First day of the week is**
1	Monday
2	Tuesday
3	Wednesday
4	Thursday
5	Friday
6	Saturday
7 (default, U.S. English)	Sunday

```
SET DATEFIRST 3;

GO

SELECT @@DATEFIRST; -- 3 (Wednesday)

GO
```

January 1 of any year defines the starting number for the **week** *datepart*. For example:

DATEPART (**wk**, 'Jan 1, *xxxx*') = 1

where *xxxx* is any year.

This table shows the return value for the **week** and **weekday** *datepart* for '2007-04-21 ' for each SET DATEFIRST argument. January 1, 2007 falls on a Monday. April 21, 2007 falls on a Saturday. For U.S. English,

SET DATEFIRST 7 -- (Sunday)

serves as the default. After setting DATEFIRST, use this suggested SQL statement for the datepart table values.

year, month, and day datepart Arguments

The values that are returned for DATEPART (**year**, *date*), DATEPART (**month**, *date*), and DATEPART (**day**, *date*) are the same as those returned by the functions YEAR, MONTH, and DAY, respectively.

iso_week datepart

ISO 8601 includes the ISO week-date system, a numbering system for weeks. Each week is associated with the year in which Thursday occurs. For example, week 1 of 2004 (2004W01) covered Monday, 29 December 2003 to Sunday, 4 January 2004. European countries/regions typically use this style of numbering. Non-European countries/regions typically don't use it.

Note: the highest week number in a year could be either 52 or 53.

The numbering systems of different countries/regions might not comply with the ISO standard. This table shows six possibilities:

First day of week	**First week of year contains**	**Weeks assigned two times**	**Used by/in**
Sunday	1 January, First Saturday, 1-7 days of year	Yes	United States
Monday	1 January, First Sunday, 1-7 days of year	Yes	Most of Europe and the United Kingdom
Monday	4 January, First Thursday, 4-7 days of year	No	ISO 8601, Norway, and Sweden
Monday	7 January, First Monday, Seven days of year	No	
Wednesday	1 January, First Tuesday, 1-7 days of year	Yes	
Saturday	1 January, First Friday, 1-7 days of year	Yes	

- **9. EOMONTH** = This function returns the last day of the month containing a specified date, with an optional offset.

Tip: You can use **DATETRUNC** to calculate the start of the month.

```
EOMONTH ( start_date [ , month_to_add ] )
```

The values shown in these result sets reflect an execution date between and including 12/01/2022 and 12/31/2022.

```
DECLARE @date DATETIME = GETDATE();
SELECT EOMONTH ( @date ) AS 'This Month';
SELECT EOMONTH ( @date, 1 ) AS 'Next Month';
SELECT EOMONTH ( @date, -1 ) AS 'Last Month';
GO
```

Here is the result set.

```
This Month
-----------------------
2022-12-31

Next Month
-----------------------
2012-01-31

Last Month
-----------------------
2022-11-30
```

❖ Mathematical

- **10. CEILING** = This function returns the smallest integer greater than, or equal to, the specified numeric expression.

```
CEILING ( numeric_expression )
```

numeric_expression
An expression of the exact numeric or approximate numeric data type category.

The return type depends on the input type of *numeric_expression*:

Input type	**Return type**
float, real	float
decimal(*p*, *s*)	decimal(38, *s*)
int, smallint, tinyint	int
bigint	bigint
money, smallmoney	money
bit	float

If the result does not fit in the return type, an arithmetic overflow error occurs.

This example shows positive numeric, negative numeric, and zero value inputs for the CEILING function.

```
SELECT CEILING($123.45), CEILING($-123.45), CEILING($0.0);
GO
```

Here is the result set.

```
--------- --------- -------------------------
124.00    -123.00    0.00
```

- 11. ROUND = Returns a numeric value, rounded to the specified length or precision.

```
ROUND ( numeric_expression , length [ ,function ] )
```

numeric_expression
Is an expression of the exact numeric or approximate numeric data type category.

length
Is the precision to which *numeric_expression* is to be rounded. *length* must be an expression of type **tinyint**, **smallint**, or **int**. When *length* is a positive number, *numeric_expression* is rounded to the number of decimal positions specified by *length*. When *length* is a negative number, *numeric_expression* is rounded on the left side of the decimal point, as specified by *length*.

function
Is the type of operation to perform. *function* must be **tinyint**, **smallint**, or **int**. When *function* is omitted or has a value of 0 (default), *numeric_expression* is rounded. When a value other than 0 is specified, *numeric_expression* is truncated.

ROUND always returns a value. If *length* is negative and larger than the number of digits before the decimal point, ROUND returns 0.

Returns the following data types.

Expression result	**Return type**
tinyint	int
smallint	int
int	int
bigint	bigint
decimal and numeric category (p, s)	decimal(p, s)
money and smallmoney category	money
float and real category	float

ROUND always returns a value. If *length* is negative and larger than the number of digits before the decimal point, ROUND returns 0.

Example	**Result**
ROUND(748.58, -4)	0

ROUND returns a rounded *numeric_expression*, regardless of data type, when *length* is a negative number.

Examples	**Result**
ROUND(748.58, -1)	750.00
ROUND(748.58, -2)	700.00
ROUND(748.58, -3)	Results in an arithmetic overflow, because 748.58 defaults to decimal(5,2), which cannot return 1000.00.
To round up to 4 digits, change the data type of the input. For example: SELECT ROUND(CAST (748.58 AS decimal (6,2)),-3);	1000.00

A. Using ROUND and estimates

The following example shows two expressions that demonstrate by using ROUND the last digit is always an estimate.

```
SELECT ROUND(123.9994, 3), ROUND(123.9995, 3);
GO
```

Here is the result set.

```
----------- -----------
123.9990    124.0000
```

B. Using ROUND and rounding approximations

The following example shows rounding and approximations.

```
SELECT ROUND(123.4545, 2), ROUND(123.45, -2);
```

Here is the result set.

```
----------  ----------
123.4500    100.00
```

C. Using ROUND to truncate

The following example uses two SELECT statements to demonstrate the difference between rounding and truncation. The first statement rounds the result. The second statement truncates the result.

```
SELECT ROUND(150.75, 0);
GO
SELECT ROUND(150.75, 0, 1);
GO

Here is the result set.

--------
151.00

(1 row(s) affected)

--------
150.00
```

❖ **Logical**

- **12. IIF** = Returns one of two values, depending on whether the Boolean expression evaluates to true or false in SQL Server.

```
IIF( boolean_expression, true_value, false_value )
```

boolean_expression

A valid Boolean expression.

If this argument is not a boolean expression then a syntax error is raised.

true_value

Value to return if *boolean_expression* evaluates to true.

false_value

Value to return if *boolean_expression* evaluates to false.

A. Simple IIF example

```
DECLARE @a INT = 45, @b INT = 40;
SELECT [Result] = IIF( @a > @b, 'TRUE', 'FALSE' );
```

Here is the result set.

```
Result
--------
TRUE
```

B. IIF with NULL constants

```
SELECT [Result] = IIF( 45 > 30, NULL, NULL );
```

The result of this statement is an error.

C. IIF with NULL parameters

```
DECLARE @P INT = NULL, @S INT = NULL;
        SELECT [Result] = IIF( 45 > 30, @P, @S );
```

Here is the result set.

```
Result
--------
NULL
```

❖ String

- **13. CONCAT** = This function returns a string resulting from the concatenation, or joining, of two or more string values in an end-to-end manner.

```
CONCAT ( argument1 , argument2 [ , argumentN ] ... )
[ ; ]
```

argument1, argument2 *[* , argumentN *]*

An expression of any string value. The CONCAT function requires at least two arguments, and no more than 254 arguments.

Return Types: A string value whose length and type depend on the input.

Remarks:

CONCAT takes a variable number of string arguments and concatenates (or joins) them into a single string. It requires a minimum of two input values; otherwise, CONCAT raises an error. CONCAT implicitly converts all arguments to string types before concatenation. CONCAT implicitly converts null values to empty strings. If CONCAT receives arguments with all NULL values, it returns an empty string of type **varchar(1)**. The implicit conversion to strings follows the existing rules for data type conversions. For more information about data type conversions, see CAST and CONVERT (Transact-SQL).

The return type depends on the type of the arguments. This table illustrates the mapping:

Input type	**Output type and length**
1. Any argument of a SQL-CLR system type, a SQL-CLR UDT, or **nvarchar(max)**	**nvarchar(max)**
2. Otherwise, any argument of type **varbinary(max)** or **varchar(max)**	**varchar(max)**, unless one of the parameters is an **nvarchar** of any length. In this case, CONCAT returns a result of type **nvarchar(max)**.
3. Otherwise, any argument of type **nvarchar** of up to 4000 characters (**nvarchar(<=** ***4000***))	**nvarchar(<=** ***4000*****)**
4. In all other cases	any argument of type **varchar** of up to 8000 characters (**varchar(<=** ***8000***)), unless one of the parameters is an **nvarchar** of any length. In that case, CONCAT returns a result of type **nvarchar(max)**.

When CONCAT receives **nvarchar** input arguments of length <= 4000 characters, or **varchar** input arguments of length <= 8000 characters, implicit conversions can affect the length of the result. Other data types have different lengths when implicitly converted to strings. For example, an **int** with value 14 has a string length of 2, while a **float** with value 1234.56789 has a string length of 7 (1234.57). Therefore, a concatenation of these two values returns a result with a length of no less than 9 characters.

If none of the input arguments has a supported large object (LOB) type, then the return type truncates to 8,000 characters in length, regardless of the return type. This truncation preserves space and supports plan generation efficiency.

CONCAT can be executed remotely on a linked server running SQL Server 2012 (11.x) and later versions. For older linked servers, the CONCAT operation will happen locally, after the linked server returns the non-concatenated values.

```
SELECT CONCAT ('Happy ', 'Birthday ', 11, '/', '25') AS Result;
```

Here is the result set.

```
Result
-------------------
Happy Birthday 11/25
```

- **14. LEFT** = Returns the left part of a character string with the specified number of characters.

```
LEFT ( character_expression , integer_expression )
```

character_expression
Is an expression of character or binary data. *character_expression* can be a constant, variable, or

column. *character_expression* can be of any data type, except **text** or **ntext**, that can be implicitly converted to **varchar** or **nvarchar**. Otherwise, use the CAST function to explicitly convert *character_expression*.

Note

If *string_expression* is of type **binary** or **varbinary**, LEFT will perform an implicit conversion to **varchar**, and therefore will not preserve the binary input.

integer_expression
Is a positive integer that specifies how many characters of the *character_expression* will be returned. If *integer_expression* is negative, an error is returned. If *integer_expression* is type **bigint** and contains a large value, *character_expression* must be of a large data type such as **varchar(max)**.

The *integer_expression* parameter counts a UTF-16 surrogate character as one character.

```
SELECT LEFT('abcdefg',2);
```

Here is the result set.

```
--

ab
```

- **15. LEN** = Returns the number of characters of the specified string expression, excluding trailing spaces.

```
LEN ( string_expression )
```

LEN excludes trailing spaces. If that is a problem, consider using the DATALENGTH (Transact-SQL) function which does not trim the string. If processing a unicode string, DATALENGTH will return a number that may not be equal to the number of characters. The following example demonstrates LEN and DATALENGTH with a trailing space.

```
DECLARE @v1 VARCHAR(40),
      @v2 NVARCHAR(40);
SELECT
@v1 = 'Test of 22 characters ',
@v2 = 'Test of 22 characters ';
SELECT LEN(@v1) AS [VARCHAR LEN] , DATALENGTH(@v1) AS [VARCHAR
DATALENGTH];
SELECT LEN(@v2) AS [NVARCHAR LEN], DATALENGTH(@v2) AS [NVARCHAR
DATALENGTH];
```

- **16. REPLACE** = Replaces all occurrences of a specified string value with another string value.

REPLACE (string_expression , string_pattern , string_replacement)

string_expression
Is the string expression to be searched. *string_expression* can be of a character or binary data type.

string_pattern
Is the substring to be found. *string_pattern* can be of a character or binary data type. *string_pattern* must not exceed the maximum number of bytes that fits on a page. If *string_pattern* is an empty string (''), *string_expression* is returned unchanged.

string_replacement
Is the replacement string. *string_replacement* can be of a character or binary data type.

Returns **nvarchar** if one of the input arguments is of the **nvarchar** data type; otherwise, REPLACE returns **varchar**.

Returns NULL if any one of the arguments is NULL.

If *string_expression* is not of type **varchar(max)** or **nvarchar(max), REPLACE** truncates the return value at 8,000 bytes. To return values greater than 8,000 bytes, *string_expression* must be explicitly cast to a large-value data type.

REPLACE performs comparisons based on the collation of the input. To perform a comparison in a specified collation, you can use COLLATE to apply an explicit collation to the input.

The following example replaces the string cde in abcdefghicde with xxx.

```
SELECT REPLACE('abcdefghicde','cde','xxx');
GO
```

Here is the result set.

```
------------
abxxxfghixxx
```

- **17. RIGHT** = Returns the right part of a character string with the specified number of characters.

RIGHT (character_expression , integer_expression)

character_expression
Is an expression of character or binary data. *character_expression* can be a constant, variable, or column. *character_expression* can be of any data type, except **text** or **ntext**, that can be implicitly converted to **varchar** or **nvarchar**. Otherwise, use the CAST function to explicitly convert *character_expression*.

integer_expression
Is a positive integer that specifies how many characters of *character_expression* will be returned.

If *integer_expression* is negative, an error is returned. If *integer_expression* is type **bigint** and contains a large value, *character_expression* must be of a large data type such as **varchar(max)**.

- **18. RTRIM** = Removes space character char(32) or other specified characters from the end of a string.

```
RTRIM ( character_expression )
```

The following example takes a string of characters that has spaces at the end of the sentence, and returns the text without the spaces at the end of the sentence.

```
SELECT RTRIM('Removes trailing spaces.   ');
```

Here is the result set.

```
Removes trailing spaces.
```

- **19. SUBSTRING** = Returns part of a character, binary, text, or image expression in SQL Server.

```
SUBSTRING ( expression, start, length )
```

expression
Is a **character**, **binary**, **text**, **ntext**, or **image** expression.

start
Is an integer or **bigint** expression that specifies where the returned characters start. (The numbering is 1 based, meaning that the first character in the expression is 1). If *start* is less than 1, the returned expression will begin at the first character that is specified in *expression*. In this case, the number of characters that are returned is the largest value of either the sum of *start* + *length*- 1 or 0. If *start* is greater than the number of characters in the value expression, a zero-length expression is returned.

length
Is a positive integer or **bigint** expression that specifies how many characters of the *expression* will be returned. If *length* is negative, an error is generated and the statement is terminated. If the sum of *start* and *length* is greater than the number of characters in *expression*, the whole value expression beginning at *start* is returned.

Returns character data if *expression* is one of the supported character data types. Returns binary data if *expression* is one of the supported **binary** data types. The returned string is the same type as the specified expression with the exceptions shown in the table.

Specified expression	**Return type**
char/varchar/text	varchar
nchar/nvarchar/ntext	nvarchar
binary/varbinary/image	varbinary

Remarks

The values for *start* and *length* must be specified in number of characters for **ntext**, **char**, or **varchar** data types and bytes for **text**, **image**, **binary**, or **varbinary** data types.

The *expression* must be **varchar(max)** or **varbinary(max)** when the *start* or *length* contains a value larger than 2147483647.

Here is how to display the second, third, and fourth characters of the string constant abcdef.

```
SELECT x = SUBSTRING('abcdef', 2, 3);
```

Here is the result set.

```
x
----------
bcd
```

- **20. TRIM** = Removes the space character char(32) or other specified characters from the start and end of a string.

```
TRIM ( [ characters FROM ] string )
```

characters

A literal, variable, or function call of any non-LOB character type (**nvarchar**, **varchar**, **nchar**, or **char**) containing characters that should be removed. **nvarchar(max)** and **varchar(max)** types aren't allowed.

string

An expression of any character type (**nvarchar**, **varchar**, **nchar**, or **char**) where characters should be removed.

Returns a character expression with a type of string argument where the space character char(32) or other specified characters are removed from both sides. Returns NULL if input string is NULL.

By default, the TRIM function removes the space character from both the start and the end of the string. This behavior is equivalent to LTRIM(RTRIM(@string)).

A. Remove the space character from both sides of string

The following example removes spaces from before and after the word test.

```
SELECT TRIM( '     test    ') AS Result;
```

Here is the result set.

```
Test
```

B. Remove specified characters from both sides of string

The following example provides a list of possible characters to remove from a string.

```
SELECT TRIM( '.,! ' FROM '     #     test    .') AS Result;
```

Here is the result set.

```
#     test
```

In this example, only the trailing period and spaces from before # and after the word test were removed. The other characters were ignored because they didn't exist in the string.

C. Remove specified characters from the start of a string

The following example removes the leading . from the start of the string before the word test.

```
SELECT TRIM(LEADING '.,! ' FROM  '     .#     test    .') AS Result;
```

Here is the result set.

```
# test .
```

D. Remove specified characters from the end of a string

The following example removes the trailing . from the end of the string after the word test.
SQLCopy

```
SELECT TRIM(TRAILING '.,! ' FROM '     .#     test    .') AS Result;
```

Here is the result set.

```
.#     test
```

E. Remove specified characters from the beginning and end of a string

The following example removes the characters 123 from the beginning and end of the string 123abc123.

```
SELECT TRIM(BOTH '123' FROM '123abc123') AS Result;
```

Here is the result set.

```
abc
```