

# Functions for positions

FUNCTIONS FOR MANIPULATING DATA IN SQL SERVER



**Ana Voicu**  
Data Engineer

# Position functions

- LEN()
- CHARINDEX()
- PATINDEX()

# LEN()

## Definition

- Returns the number of characters of the provided string.

## Syntax

```
LEN(character_expression)
```

# LEN() example - constant parameter

```
SELECT LEN('Do you know the length of this sentence?') AS length
```

```
|length|  
|-----|  
|40    |
```

# LEN() example - table column parameter

```
SELECT DISTINCT TOP 5  
    bean_origin,  
    LEN(bean_origin) AS length  
FROM ratings;
```

bean_origin	length
Toscano Black	13
Trinite	7
Ocumare- Puerto Cabello	23
Maracaibo- El Rosario	21
Madagascar	10

# CHARINDEX()

## Definition

- Looks for a character expression in a given string.
- Returns its starting position.

## Syntax

```
CHARINDEX (expression_to_find, expression_to_search [, start_location])
```

# CHARINDEX() example

SELECT

```
CHARINDEX('chocolate', 'White chocolate is not real chocolate'),  
CHARINDEX('chocolate', 'White chocolate is not real chocolate',10),  
CHARINDEX('chocolates', 'White chocolate is not real chocolate');
```

position beginning	position in string	position of non-existing exp
7	29	0

# PATINDEX()

## Definition

- Similar to CHARINDEX()
- Returns the starting position of a pattern in an expression

## Syntax

```
PATINDEX ('%pattern%', expression, [location ])
```



# Wildcard characters

Wildcard	Explanation
%	Match any string of any length (including zero length)
_	Match on a single character
[]	Match on any character in the [] brackets (for example, [abc] would match on a, b, or c characters)

# PATINDEX() example

```
SELECT
```

```
PATINDEX('%chocolate%', 'White chocolate is not real chocolate') AS position1,  
PATINDEX('%ch_c%', 'White chocolate is not real chocolate') AS position2;
```

position1	position2
7	7

# Let's practice!

**FUNCTIONS FOR MANIPULATING DATA IN SQL SERVER**

# Functions for string transformation

FUNCTIONS FOR MANIPULATING DATA IN SQL SERVER

SQL

**Ana Voicu**  
Data Engineer

# LOWER() and UPPER()

`LOWER(character_expression)`

- Converts all characters from a string to lowercase.

`UPPER(character_expression)`

- Converts all characters from a string to uppercase.

# LOWER() and UPPER() example

```
SELECT
    country,
    LOWER(country) AS country_lowercase,
    UPPER(country) AS country_uppercase
FROM voters;
```

country	country_lowercase	country_uppercase
Denmark	denmark	DENMARK
France	france	FRANCE
Belgium	belgium	BELGIUM

# LEFT() and RIGHT()

`LEFT(character_expression, number_of_characters)`

- Returns the specified number of characters from the beginning of the string

`RIGHT(character_expression, number_of_characters)`

- Returns the specified number of characters from the end of the string

# LEFT() and RIGHT() example

```
SELECT
  country,
  LEFT(country, 3) AS country_prefix,
  email,
  RIGHT(email, 4) AS email_domain
FROM voters;
```

country	country_prefix	email	email_domain
Denmark	Den	carol8@yahoo.com	.com
France	Fra	ana0@gmail.com	.com
Belgium	Bel	angela23@gmail.com	.com



# LTRIM(), RTRIM(), and TRIM()

`LTRIM(character_expression)`

- Returns a string after removing the leading blanks.

`RTRIM(character_expression)`

- Returns a string after removing the trailing blanks.

`TRIM([characters FROM] character_expression)`

- Returns a string after removing the blanks or other specified characters.

# REPLACE()

```
REPLACE(character_expression, searched_expression, replacement_expression)
```

- Returns a string where all occurrences of an expression are replaced with another one.

```
SELECT REPLACE('I like apples, apples are good.', 'apple', 'orange') AS result;
```

```
| result |
|-----|
| I like oranges, oranges are good. |
```

# SUBSTRING()

```
SUBSTRING(character_expression, start, number_of_characters)
```

- Returns part of a string.

```
SELECT SUBSTRING('123456789', 5, 3) AS result;
```

```
| result |  
|-----|  
| 567   |
```

# Let's practice!

**FUNCTIONS FOR MANIPULATING DATA IN SQL SERVER**

# Functions manipulating groups of strings

FUNCTIONS FOR MANIPULATING DATA IN SQL SERVER

SQL

**Ana Voicu**  
Data Engineer

# CONCAT() and CONCAT\_WS()

```
CONCAT(string1, string2 [, stringN ])
```

```
CONCAT_WS(separator, string1, string2 [, stringN ])
```

**Keep in mind:** concatenating data with functions is better than using the "+" operator.

# CONCAT() and CONCAT\_WS() example

```
SELECT
```

```
    CONCAT('Apples', 'and', 'oranges') AS result_concat,  
    CONCAT_WS(' ', 'Apples', 'and', 'oranges') AS result_concat_ws,  
    CONCAT_WS('***', 'Apples', 'and', 'oranges') AS result_concat_ws2;
```

result_concat	result_concat_ws	result_concat_ws2
Applesandoranges	Apples and oranges	Apples***and***oranges

# STRING\_AGG()

```
STRING_AGG(expression, separator) [ <order_clause> ]
```

- Concatenates the values of string expressions and places separator values between them.



# STRING\_AGG() example

```
SELECT
  STRING_AGG(first_name, ',') AS list_of_names
FROM voters;
```

```
| list_of_names |
|-----|
| Carol,Ana,Melissa,Angela,Grace,Melody... |
```

```
SELECT
  STRING_AGG(CONCAT(first_name, ' ', last_name, ' (' , first_vote_date, ')'), CHAR(13)) AS list_of_names
FROM voters;
```

```
| list_of_names |
|-----|
| Carol Rai (2015-03-09) |
| Ana Price (2015-01-17) ... |
```

# STRING\_AGG() with GROUP BY

```
SELECT
    YEAR(first_vote_date) AS voting_year,
    STRING_AGG(first_name, ', ') AS voters
FROM voters
GROUP BY YEAR(first_vote_date);
```

voting_year	voters
2013	Melody, Clinton, Kaylee,...
2014	Brett, Joe, April, Mackenzie,...
2015	Cedric, Julie, Sandra,...
2016	Isabella, Vincent, Haley,...

# STRING\_AGG() with the optional <order\_clause>

```
SELECT
  YEAR(first_vote_date) AS voting_year,
  STRING_AGG(first_name, ', ') WITHIN GROUP (ORDER BY first_name ASC) AS voters
FROM voters
GROUP BY YEAR(first_vote_date);
```

voting_year	voters
2013	Amanda, Anthony, Caroline,...
2014	April, Brett, Bruce, Carl, ...
2015	Abigail, Alberto, Alexa,...
2016	Barbara, Haley, Isabella,...

# STRING\_SPLIT()

```
STRING_SPLIT(string, separator)
```

- Divides a string into smaller pieces, based on a separator.
- Returns a single column table.

```
SELECT *  
FROM STRING_SPLIT('1,2,3,4', ',')
```

```
| value |  
|-----|  
| 1     |  
| 2     |  
| 3     |  
| 4     |
```

# Let's practice!

**FUNCTIONS FOR MANIPULATING DATA IN SQL SERVER**