

## SUMMARY: STRING OPERATIONS

### SESSION OVERVIEW:

By the end of this session, the students will be able to:

- Understand the different concepts of string operations like LENGTH, TRIM, CONCAT, UPPER, LOWER, LPAD, RPAD, LEFT, RIGHT, REVERSE, REPLACE, etc.

### KEY TOPICS AND EXAMPLES:

#### Understand the different concepts of string operations:

- **Uses of the string functions:**
  - **Data Cleaning:** String operations are often used to clean up data, such as removing leading or trailing spaces, converting text to a specific case, or replacing certain characters.
  - **Data Transformation:** String operations can transform data into a format that is suitable for analysis or presentation. This can include concatenating strings, extracting substrings, or padding strings to a specific length.
  - **Data Manipulation:** String operations can be used to manipulate data for various purposes. For example, generating formatted output for reports, creating dynamic SQL queries, or formatting data for display purposes.
  - **Data Integration:** String operations are often used when integrating data from multiple sources. This can include standardizing text data from different sources or transforming data to match a specific format.

#### **1. Determining the length of the string:**

In SQL, we can use the LENGTH function to find the length of a string. These functions return the number of characters in the string.

#### **Syntax:**



```
SELECT LENGTH(column_name) AS length_of_string
FROM table_name;
```

*NOTE: Replace column\_name with the name of the column containing the string you want to measure, and table\_name with the name of your table. The result will be a single column named length\_of\_string that contains the length of each string in the specified column.*

#### **Example 1:**

```
# Query to find the length of the Firstname of the customers.
SELECT length(firstname) AS length_of_string
FROM customers;
```

### Output:

Result Grid			 Filter Rows:	
	length_of_string			
▶	3			
	6			
	5			
	7			
	9			
	5			
	5			
	5			

### Example 2:

```
# Query to find the length of the names of the products.
SELECT productname, length(productName) AS length_of_string
FROM products;
```

### Output:

Result Grid		Filter Rows:
	Productname	length_of_string
▶	Sport-100 Helmet, Red	21
	Sport-100 Helmet, Black	23
	Mountain Bike Socks, M	22
	Mountain Bike Socks, L	22
	Sport-100 Helmet, Blue	22
	AWC Logo Cap	12
	Long-Sleeve Logo Jersey, S	26
	Long-Sleeve Logo Jersey, M	26


### Example 3:

This example helps you understand the length of a NULL string and returns the string as NULL itself. The output returns both columns as NULL.

```
# Query to find the length of a NULL value.

SELECT prefix, LENGTH(prefix) AS length_of_string
FROM customers
WHERE prefix IS NULL;
```

### Output:

Result Grid  Filter Rows:

	prefix	length_of_string
▶	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL
	NULL	NULL

Result 16 x

## 2. Trimming the string:

Trimming in SQL refers to the process of removing leading and trailing spaces (or other specified characters) from a string. It's commonly used to clean up data before processing or displaying it.

In SQL, the TRIM function is used for this purpose. There are three variants of the TRIM function:

- TRIM: Removes leading and trailing spaces (or specified characters) from a string.

### Syntax:

```
TRIM([{BOTH | LEADING | TRAILING} [trim_character] FROM] string)
```

### NOTE:

- {BOTH | LEADING | TRAILING}: Specifies whether to trim the characters from both ends (BOTH), only from the beginning (LEADING), or only from the end (TRAILING) of the string. If not specified, BOTH is assumed.*
- trim\_character: Optional. Specifies the character or characters to be removed from the string. If not specified, it defaults to removing spaces.*
- string: The string from which to remove the specified characters.*

### Example 1:

```
SELECT TRIM(BOTH 'x' FROM 'xxxCodingNinjasxxx') AS trimmed_string;
```

### Output:

The above query will remove all the 'x' from both the end.

**Output:** CodingNinjas

### Example 2:

```
SELECT TRIM(' Hello ') AS trimmed_string;
```

### Output:

The above query will remove all the leading or trailing spaces from the mentioned string.

**Output:** 'Hello'

- LTRIM:** The LTRIM() function helps to return and remove all the space characters

found on the left-hand side of the string.

**Syntax:**

```
LTRIM(string, [trim_string])
```

**NOTE:**

- *string* – The string from which the leading space character would be removed.
- *trim\_string* – It is an optional parameter that specifies the characters to be removed from the given string.

**Example 1:**

```
SELECT LTRIM(' Hello ') AS trimmed_string;
```

**Output:**

The above query will remove all the leading spaces from the mentioned string.

Output: 'Hello '

- c. **RTRIM:** The RTRIM() function removes trailing spaces from a string.

**Syntax:**

```
RTRIM(input_string)
```

**NOTE:**

- *Parameter: RTRIM()* function accepts a single-parameter like *input\_string*.
- *Input\_string:* It is an expression of character or binary data. It can be a literal string, variable, or column.
- *Returns* – It returns a string after truncating all trailing blanks.

**Example 1:**

```
SELECT RTRIM(' Hello ') AS trimmed_string;
```

**Output:**

The above query will remove all the trailing spaces from the mentioned string.

Output: ' Hello'

### 3. CONCATENATING a string:

The CONCAT function in SQL is used to concatenate two or more strings into a single string.

The CONCAT function is very useful when we clean the data in our initial stages.

Let's discuss some of the areas where we can use the CONCAT function:

- **Displaying Full Names:** Concatenate first name and last name to display a full name.
- **Building File Paths:** Concatenate directory names and file names to construct file paths.
- **Creating Custom Messages:** Concatenate strings to create custom messages or notes.
- **Formatting Addresses:** Concatenate address components to format addresses for display.
- **Combining Text and Numbers:** Concatenate text and numbers to create descriptive

labels or codes.

- **Generating Usernames:** Concatenate first name, last name, or other parts to generate usernames.

### Basic Syntax:

```
CONCAT(string1, string2, ...)
```

### Syntax with SELECT statement:

```
SELECT CONCAT(string1, string2, ...) AS concatenated_string
FROM table_name;
```

### Note:

- *string1, string2, etc.:* Strings you want to concatenate. These can be column names, string literals, or a combination of both.
- *AS concatenated\_string:* Optional alias for the concatenated string in the result set, this alias can be used to refer to the concatenated string in the output.
- *FROM table\_name:* Specifies the table from which to select the strings.

### Example 1:

```
# Query to merge the first name and the last name of the customers as full name.
```

```
SELECT CONCAT(firstname, ' ', lastname) AS full_name
FROM customers;
```

### Output:

Result Grid

Filter Rows:

	full_name
▶	JON YANG
	EUGENE HUANG
	RUBEN TORRES
	CHRISTY ZHU
	ELIZABETH JOHNSON
	JULIO RUIZ
	MARCO MEHTA
	MARTIN VERHOEFF

Result 17

×

### Example 2:

In this example, we will understand a little bit about the cleaning part using some of the previous session's concepts that we have learned. So, we have the customers table in which we have two separate columns "firstname" and "lastname". We want a single column as full\_name for a better understanding of data thus want to merge both columns. So, here are the steps that we are going to follow:

- First, we will create a separate column after the lastname column as we do not want the full\_name column at the end of the dataset.

- Now we will update the new column using the CONCAT function where we will specify the format of it.
- Lastly, we will delete the columns which are not required as a part of the dataset.
- To check the results of the below query use the SELECT statement.

# Query to create a separate column and merge the first name and the last name as full name. Also, delete both the original columns.

```
ALTER TABLE customers
ADD COLUMN full_name VARCHAR(255)
AFTER lastname;

UPDATE customers
SET full_name = CONCAT(firstname, ' ', lastname);

ALTER TABLE customers
DROP firstname;

ALTER TABLE customers
DROP lastname;

SELECT * FROM customer
```

### Output:

Once we run the above query let's check the changes.

CustomerKey	Prefix	full_name	DateOfBirth	MaritalStatus	Gender	EmailAddress	Regions	AnnualIncome	TotalChildrer	EducationLevel	Occupation	HomeOwner	MyU
11000	MR.	JOIN YANG	1966-08-04	M	M	jon24@learnsector.com	NULL	\$90,000	2	Bachelors	Professional	Y	
11001	MR.	EUGENE HUANG	1965-05-14	S	M	eugene10@learnsector.com	NULL	\$60,000	3	Bachelors	Professional	N	
11002	MR.	RUBEN TORRES	1965-12-08	M	M	ruben35@learnsector.com	NULL	\$60,000	3	Bachelors	Professional	Y	
11003	MS.	CHRISTY ZHU	1968-02-15	S	F	christy12@learnsector.com	NULL	NULL	0	Bachelors	Professional	N	
11004	MRS.	ELIZABETH JOHNSON	1968-08-08	S	F	elizabeth5@learnsector.com	NULL	\$80,000	5	Bachelors	Professional	Y	
11005	MR.	JULIO RUIZ	1965-05-08	S	M	julio1@learnsector.com	NULL	\$70,000	0	Bachelors	Professional	Y	
11007	MR.	MARCO MEHTA	1964-09-05	M	M	marco14@learnsector.com	NULL	\$60,000	3	Bachelors	Professional	Y	Y

### Alternative method:

Instead of using the CONCAT function, we can use the || || connotation to merge two or more columns or two or more strings.

```
SELECT firstname || ' ' || lastname AS full_name
FROM customers;
```

The above query will help us to merge the firstname column and the last name column and will name the column full\_name.

### Example 3:

```
SELECT CONCAT('Customer Number: ', CAST(customerkey AS CHAR(10))) AS
customer_label
FROM customers;
```

#### Output:

Result Grid		Filter Rows:
	customer_label	
▶	Customer Number: 11000	
	Customer Number: 11001	
	Customer Number: 11002	
	Customer Number: 11003	
	Customer Number: 11004	
	Customer Number: 11005	
	Customer Number: 11007	
	Customer Number: 11008	
	Customer Number: 11009	

Result 17 x

#### 4. Using SUBSTRING\_INDEX in a string:

The SUBSTRING\_INDEX function in MySQL is designed to split a string into multiple parts based on a specified delimiter. It can be particularly useful when dealing with complex data that needs to be parsed and processed. By breaking down a string into smaller components, users can easily extract and manipulate the desired information.

#### Basic Syntax:

```
SUBSTRING_INDEX(str, delim, count)
```

#### NOTE:

- *str is the input string from which to extract the substring.*
- *delim is the delimiter used to separate the parts of the string.*
- *count is the number of occurrences of the delimiter to search for.*

#### Important:

- **If the count is positive, SUBSTRING\_INDEX returns all parts of the string up to (and including) the count-th occurrence of the delimiter.**
- **If the count is negative, SUBSTRING\_INDEX returns all parts of the string from the count-th occurrence to the end of the string.**

#### Syntax with SELECT statement:

```
SELECT SUBSTRING_INDEX(column_name, delimiter, count) AS
result_column_name
FROM table_name;
```

#### NOTE:

- *column\_name is the name of the column containing the string you want to manipulate.*
- *delimiter is the character or substring used to separate the parts of the string.*

- *count is the number of occurrences of the delimiter to consider.*

**Example 1:**

In the previous example, we have understood the CONCAT function using the dataset. Now we will understand the use of the SUBSTRING\_INDEX function in the same dataset with the help of the same columns. We will be using some of the concepts from the previous sessions as well. So here are the steps that will be followed:

- Firstly, we will create two new columns named first\_name and last\_name to the customers table after the full\_name column.
- Update the first\_name column by extracting the first part of the full\_name column before the delimiter.
- Update the last\_name column by extracting the last part of the full\_name column after the delimiter.
- Drop the full\_name column from the customers table.
- To check the changes that were done, use the SELECT statement.

```
# Query to split the full name of the customer into first name and last name in two separate columns.
```

```
ALTER TABLE customers
ADD COLUMN first_name VARCHAR(255)
AFTER full_name;
```

```
ALTER TABLE customers
ADD COLUMN last_name VARCHAR(255)
AFTER first_name;
```

```
UPDATE customers
SET first_Name=SUBSTRING_INDEX (full_name, ' ', 1);
```

```
UPDATE customers
SET last_Name=substring_index(full_name, ' ', -1);
```

```
ALTER TABLE customers
DROP full_name;
```

```
SELECT * FROM customers
```

**Output:**



	CustomerKey	Prefix	first_name	last_name	DateOfBirth	MaritalStatus	Gender	EmailAddress
▶	11000	MR.	JON	YANG	1966-08-04	M	M	jon24@learnsector.com
	11001	MR.	EUGENE	HUANG	1965-05-14	S	M	eugene10@learnsector.com
	11002	MR.	RUBEN	TORRES	1965-12-08	M	M	ruben35@learnsector.com
	11003	MS.	CHRISTY	ZHU	1968-02-15	S	F	christy12@learnsector.com
	11004	MRS.	ELIZABETH	JOHNSON	1968-08-08	S	F	elizabeth5@learnsector.com
	11005	MR.	JULIO	RUIZ	1965-05-08	S	M	julio1@learnsector.com
	11007	MR.	MARCO	MEHTA	1964-09-05	M	M	marco14@learnsector.com

*NOTE: The output just has a part of the table.*

### 5. Using the UPPER function in a string:

The UPPER function in SQL is used to convert a string to uppercase. It takes a string as input and returns the same string with all letters converted to uppercase. Here are some of the use cases of UPPER:

- **Case-Insensitive Comparisons:** Use UPPER to perform case-insensitive comparisons in WHERE clauses.
- **Standardizing Data:** Use UPPER to standardize the case of data for consistency.
- **Data Entry Validation:** Use UPPER to ensure that data entered into a database is consistently stored in uppercase.

#### Basic Syntax:

```
UPPER(string)
```

#### Syntax with SELECT statement:

```
SELECT UPPER(column_name) AS upper_case_column
FROM table_name;
```

#### NOTE:

- *UPPER(column\_name) is the UPPER function applied to a column named column\_name. It converts the values in column\_name to uppercase.*
- *AS upper\_case\_column is an optional alias that renames the result of the UPPER function to upper\_case\_column.*
- *FROM table\_name specifies the table from which to select the data.*

#### Example 1:

```
# Query to find the name of the product model and product SKU in capital letters.
```

```
SELECT UPPER(ModelName) AS upper_model_name,
       UPPER(productSKU) AS upper_Product_SKU
FROM Products;
```

#### Output:

Result Grid			Filter Rows:
	upper_model_name	upper_Product_SKU	
▶	SPORT-100	HL-U509-R	
	SPORT-100	HL-U509	
	MOUNTAIN BIKE SOCKS	SO-B909-M	
	MOUNTAIN BIKE SOCKS	SO-B909-L	
	SPORT-100	HL-U509-B	
	CYCLING CAP	CA-1098	
	LONG-SLEEVE LOGO JERSEY	LJ-0192-S	
	LONG-SLEEVE LOGO JERSEY	LJ-0192-M	

Result 33 x

### Example 2:

# Query to merge the education level and occupation in a column. Also, return the column in all capital letters alphabets.

```
ALTER TABLE customers
ADD COLUMN Education_Occupation varchar(255)
AFTER occupation;
```

```
UPDATE customers
SET Education_occupation = UPPER(CONCAT(EducationLevel, '-',
Occupation));
```

```
ALTER TABLE customers
DROP COLUMN EducationLevel;
```

```
ALTER TABLE customers
DROP COLUMN Occupation;
```

```
SELECT * FROM customers
```

### Output:

AnnualIncome	TotalChildrer	Education_Occupation
\$90,000	2	BACHELORS-PROFESSIONAL
\$60,000	3	BACHELORS-PROFESSIONAL
\$60,000	3	BACHELORS-PROFESSIONAL
NULL	0	BACHELORS-PROFESSIONAL
\$80,000	5	BACHELORS-PROFESSIONAL
\$70,000	0	BACHELORS-PROFESSIONAL
\$60,000	3	BACHELORS-PROFESSIONAL

*NOTE: The output just has a part of the table.*

## 6. Using the LOWER function in a string:

LOWER is a string function that takes a text input and converts it to lowercase. While this might seem like a simple task, it can be immensely helpful in situations where case sensitivity plays a crucial role in database operations, such as searching or sorting text data. By converting all characters to lowercase, LOWER ensures consistent and accurate matching.

Consider a scenario where you have a database table containing customer names. Some names may be stored in uppercase, while others are in lowercase or a mix of both. If you want to retrieve all customers whose last name is "Smith," you can use the LOWER function to convert both the column values and the search term to lowercase. This way, you can ensure that the comparison is case-insensitive and that all matching records are returned, regardless of the original case of the names.

### Basic syntax:

```
LOWER(string)
```

### Syntax with SELECT statement:

```
SELECT LOWER(column_name) AS lower_case_column
FROM table_name;
```

### NOTE:

- *LOWER(column\_name) is the LOWER function applied to a column named column\_name. It converts the values in column\_name to lowercase.*
- *AS lower\_case\_column is an optional alias that renames the result of the LOWER function to lower\_case\_column.*
- *FROM table\_name specifies the table from which to select the data.*

### Example 1:

```
SELECT LOWER(first_name) AS lower_first_name,
       LOWER(last_name) AS lower_last_name
FROM customers;
```

### Output:

Result Grid			Filter Rows:
	lower_first_name	lower_last_name	
▶	jon	yang	
	eugene	huang	
	ruben	torres	
	christy	zhu	
	elizabeth	johnson	
	julio	ruiz	
	marco	mehta	
	robin	verhoff	

Result 37 x

### Example 2:

```

ALTER TABLE customers
ADD COLUMN EducationLevel varchar(255)
AFTER Education_occupation;

ALTER TABLE customers
ADD COLUMN Occupation varchar(255)
AFTER EducationLevel;

UPDATE customers
SET EducationLevel= LOWER(SUBSTRING_INDEX(Education_occupation, "-",
1)),
    Occupation= LOWER(SUBSTRING_INDEX(Education_occupation, "-", -1));

ALTER TABLE customers
DROP Education_occupation;

SELECT * FROM customers

```

#### Output:

AnnualIncome	TotalChildrer	EducationLevel	Occupation	HomeOwner
\$90,000	2	bachelors	professional	Y
\$60,000	3	bachelors	professional	N
\$60,000	3	bachelors	professional	Y
NULL	0	bachelors	professional	N
\$80,000	5	bachelors	professional	Y
\$70,000	0	bachelors	professional	Y
\$60,000	3	bachelors	professional	Y

*NOTE: The output just has a part of the table.*

#### 7. Using RIGHT in a string:

MySQL RIGHT() extracts a specified number of characters from the right side of a string.

This function is useful in -

- **Leading character removal:** LTRIM() removes certain characters from the left side of a string.
- **Data cleaning:** LTRIM() is often used to remove spaces from strings at the beginning.

#### Basic Syntax:

```
RIGHT(str, length)
```

#### Syntax with SELECT statement:

```

SELECT RIGHT(column_name, length) AS result
FROM table_name;

```

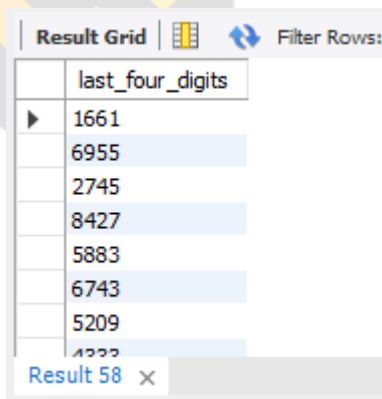
*Note:*

- *column\_name* is the name of the column from which you want to extract characters.
- *length* is the number of characters to extract from the right side of the string in *column\_name*.

**Example 1:**

```
SELECT RIGHT(phone_number, 4) AS last_four_digits
FROM customers;
```

**Output:**



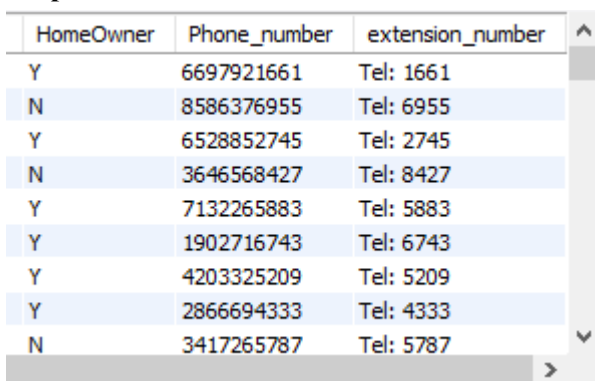
last_four_digits
1661
6955
2745
8427
5883
6743
5209
4333

**Example 2:**

```
ALTER TABLE customers
ADD COLUMN extension_number varchar(255);

UPDATE customers
SET extension_number= CONCAT('Tel: ', RIGHT(phone_number, 4));
```

**Output:**



HomeOwner	Phone_number	extension_number
Y	6697921661	Tel: 1661
N	8586376955	Tel: 6955
Y	6528852745	Tel: 2745
N	3646568427	Tel: 8427
Y	7132265883	Tel: 5883
Y	1902716743	Tel: 6743
Y	4203325209	Tel: 5209
Y	2866694333	Tel: 4333
N	3417265787	Tel: 5787

*NOTE: The output just has a part of the table.*

**8. Using LEFT in a string:**

MySQL LEFT() returns a specified number of characters from the left of the string. Both the number and the string are supplied as arguments of the function.

This function is useful in -

- **Substring extraction:** It extracts characters from the left side of a string.
- **Data truncation:** The LEFT() function truncates or limits the length of a string by extracting only a certain number of characters.
- **Data manipulation:** LEFT() is commonly used to extract part of a string or modify its content based on the leftmost characters in a string.

Basic syntax:

```
LEFT (string, length)
```

Syntax with SELECT statement:

```
SELECT LEFT(column_name, number_of_characters) AS new_column_name  
FROM table_name;
```

**NOTE:**

- *column\_name is the name of the column from which you want to extract characters.*
- *number\_of\_characters is the number of characters you want to extract from the left side of the string.*
- *new\_column\_name is the alias for the extracted characters, which will be displayed in the result set.*

Example 1:

```
ALTER TABLE products  
ADD COLUMN Product_color_code varchar(255)  
AFTER ProductColor;  
  
UPDATE products  
SET Product_color_code= UPPER(LEFT(ProductColor, 3)),  
    Product_color_code= "Unknown" where Product_color_code="NA";  
  
SELECT * FROM products
```

Output:

ProductColor	Product_color_code	ProductSize	ProductStyle
Red	RED	0	0
Black	BLA	0	0
White	WHI	M	U
White	WHI	L	U
Blue	BLU	0	0
Multi	MUL	0	U
Multi	MUL	S	U
Multi	MUL	M	U
...	...	.	..

**NOTE:** The output just has a part of the table.

## 9. Using LPAD in a string:

MySQL LPAD() left pads a string with another string. The actual string, a number indicating the length of the padding in characters (optional) and the string to be used for left padding - all are passed as arguments.

**This function is useful in:**

- **String padding:** It allows you to left-pad a string with a specific character or set. This is useful for aligning text or numbers in fixed-width formats.
- **Data presentation:** LPAD() can format or present data in a desired format. We can use it to align text or display numbers with leading zeros.
- **String manipulation:** LPAD() is often used to construct dynamic strings or generate values in specific formats.

**Basic Syntax:**

```
LPAD(str, len, padstr)
```

**Syntax with SELECT statement:**

```
SELECT LPAD(str, len, padstr) AS padded_string
FROM table_name;
```


**NOTE:**


- *str is the input string that you want to pad.*
- *len is the length of the resulting padded string.*
- *padstr is the padding string that is added to the left side of str to make it len characters long.*

**Example 1:**

```
SELECT LPAD(phone_number, 10, '0') AS formatted_phone_number
FROM customers;
```

**Output:**

Result Grid			Filter Rows:
	formatted_phone_number		
	8563894559		
	0487969276		
	3233633142		
	3564437988		
	3134978939		
	9289749546		
	7925936138		
	4215864415		
	1892351026		
	...		

Result 65 

**NOTE:** The output just has a part of the table.

**Example 2:**

```
ALTER TABLE customers
ADD COLUMN formatted_number varchar(255)
AFTER Phone_number;

UPDATE customers
SET formatted_number= CONCAT('(+91)', ' ', LPAD(RIGHT(phone_number, 10),
10, '0'));

Select * from customers;
```

**Output:**

Phone_number	formatted_number	extension_number
321266289	(+91) 0321266289	Tel: 6289
9802178754	(+91) 9802178754	Tel: 8754
1614127478	(+91) 1614127478	Tel: 7478
8624379998	(+91) 8624379998	Tel: 9998
5849949131	(+91) 5849949131	Tel: 9131
5747022890	(+91) 5747022890	Tel: 2890
7702187968	(+91) 7702187968	Tel: 7968
78067965	(+91) 0078067965	Tel: 7965

*NOTE: The output just has a part of the table.*

#### 10. Using RPAD in a string:

MySQL RPAD() function pads strings from right. The actual string that is to be padded, the length of the string returned after padding, and the string that is used for padding.

**This function is useful in -**

- **String padding:** It allows us to right-pad a string with a specific character or a set of characters.
- **Data presentation:** RPAD() can format or present data in a desired format. For example, we can use it to display numbers with trailing zeros or align text in columns.

**Basic syntax:**

```
RPAD(str, len, padstr)
```

**Syntax using SELECT statement:**

```
SELECT RPAD(column_name, length, pad_character) AS new_column_name
FROM table_name;
```

**NOTE:**

- *column\_name is the name of the column you want to pad.*
- *length is the total length that you want the resulting string to be after padding.*
- *pad\_character is the character that will be used for padding. If not specified, it defaults to a*






*space.*

#### Example 1:

```
SELECT RPAD(phone_number, 10, '0') AS formatted_phone_number
FROM customers;
```

#### Output:

Result Grid			 Filter Rows
	formatted_phone_number		
▶	6697921661		
	8586376955		
	6528852745		
	3646568427		
	7132265883		
	1902716743		
	4203325209		
	2866694333		
	3417265787		
	3417265787		

Result 73 

#### 11. Using the REVERSE function in a string:

In SQL, the REVERSE function is used to reverse a string. It reverses the order of characters in a string. The syntax for the REVERSE function varies slightly depending on the database system you are using.

##### This function is useful in -

- **String reversal:** It allows us to reverse the order of characters in a string.
- **Palindrome detection:** REVERSE() is often used to check whether a string is a palindrome, which means it remains the same when its characters are reversed.
- **Data transformation:** REVERSE() can be used to transform data by changing the order of characters.

#### Basic syntax:

```
REVERSE(str)
```



#### Syntax with SELECT statement:


```
SELECT REVERSE(column_name) AS reversed_string
FROM table_name;
```

#### Example 1:

```
SELECT REVERSE(RIGHT(Phone_number, 4)) AS reversed_string
FROM customers;
```

#### Output:

Result Grid			 Filter Rows:
	reversed_string		
▶	1661		
	5596		
	5472		
	7248		
	3885		
	3476		
	9025		
	3334		
	7875		
	6766		

Result 78 

**NOTE:**

- *column\_name* is the name of the column containing the string you want to reverse.
- *table\_name* is the name of the table containing the column.

## 12. Using the REPLACE function in a string:

MySQL REPLACE() replaces all the occurrences of a substring within a string.

This function is useful in -

- **Substring replacement:** It allows us to replace all occurrences of a substring with a new substring.
- **Removing characters:** By replacing a specific substring with an empty string, REPLACE() can remove characters or substrings from a string.

### Basic Syntax:

```
REPLACE(str, find_string, replace_with)
```

### Syntax with SELECT statements:

```
SELECT REPLACE(column_name, 'substring_to_replace',
'replacement_substring') AS new_column_name
FROM table_name;
```

**NOTE:**

- *column\_name* is the name of the column in which you want to perform the replacement.
- *'substring\_to\_replace'* is the substring you want to replace.
- *'replacement\_substring'* is the substring with which you want to replace *'substring\_to\_replace'*.
- *new\_column\_name* is the alias for the column that will contain the replaced values in the result set.

### Example 1:

```
SELECT REPLACE(productStyle, 0, 'N/A') AS replaced_column
FROM products;
```

### Output:

Result Grid	
	replaced_column
▶	N/A
	N/A
	U
	U
	N/A
	U
	U
	U
	U
	..

### Example 2:

```
SELECT
    CONCAT(REPLACE(annualincome, ',000', ''), 'k') AS formatted_income
FROM customers
Where annualincome IS NOT NUL;
```

### Output:

Result Grid	
	formatted_income
	\$60 k
	\$60 k
	\$60 k
	\$60 k
	\$70 k
	\$70 k
	\$70 k
	\$70 k
	\$70 k
	\$70 k

### Example 3:

```
UPDATE customers
SET annualincome = (REPLACE(REPLACE(annualincome, '$', ''), ', ', ''));

ALTER TABLE customers
MODIFY annualincome INT;

SELECT * FROM customers
```

### Output:

You must have noticed that the annualincome column has been stored in text data type. But as it is an income column it is supposed to be stored in integer data type. Using the above query we have replaced the text-specific characters with null characters which omits those characters. The second

step is to convert the data type from text to int using the ALTER TABLE operation.

### 13. Using the SUBSTRING function:

MySQL SUBSTR() returns the specified number of characters from a particular position of a given string. SUBSTR() is a synonym for SUBSTRING().

This function is useful in -

- **Substring extraction:** Substrings can be extracted by specifying their length and position in a string.
- **Data manipulation:** The SUBSTR() function is often used in data manipulation tasks, such as extracting specific parts of a string for further processing or modifying the content of a string based on the desired substring.

#### Syntax:

```
SUBSTRING(string, position, length)
```



#### NOTE:


- *string: The original string from which you want to extract the substring.*
- *position: The starting position from where the substring should begin. The first position in the string is 1.*
- *length: The number of characters to extract from the starting position. This parameter is optional; if omitted, the substring will extend to the end of the string.*

#### Example 1:

```
SELECT SUBSTRING(productSKU, 1, 7) AS productSKU
FROM products;
```

#### Output:

Result Grid			 Filter Rows
	productSKU		
	FR-T98U		
	FR-T98U		
	RD-2308		
	FR-T67U		
	FR-T67U		
	FR-T67U		
	FR-T67U		
	FR-T67Y		
	FR-T67Y		

Result 19 

### 14. Using the INSTR function:

MySQL INSTR() takes a string and a substring of it as arguments and returns an integer which indicates the position of the first occurrence of the substring within the string.

This function is useful in -

- You can determine the position of a substring within a string using this function.
- As a boolean check, INSTR() determines whether a string contains a substring.
- Data manipulation tasks can be performed using INSTR(), such as extracting substrings or replacing them based on their position.

#### Syntax:

```
INSTR(string, substring)
```



#### NOTE:

- *string: The string in which you want to search for the substring.*
- *substring: The substring you are searching for.*

#### Example:

```
SELECT emailaddress, INSTR(emailaddress, '@') AS at_position
FROM customers;
```

#### Output:

Result Grid   Filter Rows: <input type="text"/>		
	emailaddress	at_position
▶	jon24@learnsector.com	6
	eugene10@learnsector.com	9
	ruben35@learnsector.com	8
	christy12@learnsector.com	10
	elizabeth5@learnsector.com	11
	julio1@learnsector.com	7
	marco14@learnsector.com	8
	rob4@learnsector.com	5
	shannon38@learnsector.com	10

Result 21 