



SQL Built-in functions



Agenda

- SQL Built-in functions
- String Functions
- More String Functions and Examples



SQL Built-in function

- MySQL ships with built-in functions.
- You can categorize them based on the type of data they operate on, e.g. strings, dates, and numeric built-in functions.



SQL Built-in function

- There are different types of SQL built - in functions, which are mentioned below:
 - String
 - Numeric
 - Date

SQL Built-in function

- We use the following Employee table to understand the built-in function:

| ID | NAME | AGE | ADDRESS | SALARY |
|----|--------|-----|------------|--------|
| 1 | Kellie | 32 | California | 2000 |
| 2 | Pete | 25 | Texas | 1500 |
| 3 | Popy | 23 | Boston | 2000 |
| 4 | Sam | 25 | Florida | 6500 |
| 5 | Jhon | 27 | Hawaii | 10000 |



String functions

String functions

- String functions operate on an input string and return an output string.
- The following string functions are defined in SQL.

| Functions | Meaning |
|-----------|--|
| ASCII | The ASCII code value of a character |
| CHAR | To convert an ASCII value to a character |
| CHARINDEX | The position of a substring within a string starting from a specified location is returned |

More String functions

| Functions | Meaning |
|-----------|--|
| CONCAT | Connects multiple strings into one string |
| LEFT | A given number of characters can be extracted from a character string starting at the left |
| LEN | The number of characters in a character string |
| LOWER | Changing a string to lowercase |
| LTRIM | A new string is returned after removing all leading blanks from a specified string |

More String functions

| Functions | Meaning |
|-----------|---|
| REPLACE | In a string, replace all occurrences of a substring with another substring. |
| REVERSE | The reverse order of a character string |
| RIGHT | A given number of characters can be extracted from a character string starting from the right |
| RTRIM | Removes all trailing blanks from a string and gives a new string. |
| REPLACE | All occurrences of one substring in a string should be replaced with another substring |

More String functions

| Functions | Meaning |
|-----------|--|
| SUBSTRING | From a specified location, extract a substring with a specified length |
| TRIM | Return a new string after removing all leading and trailing blanks from a specified string |
| UPPER | To convert a string to uppercase |

String function - Example 1

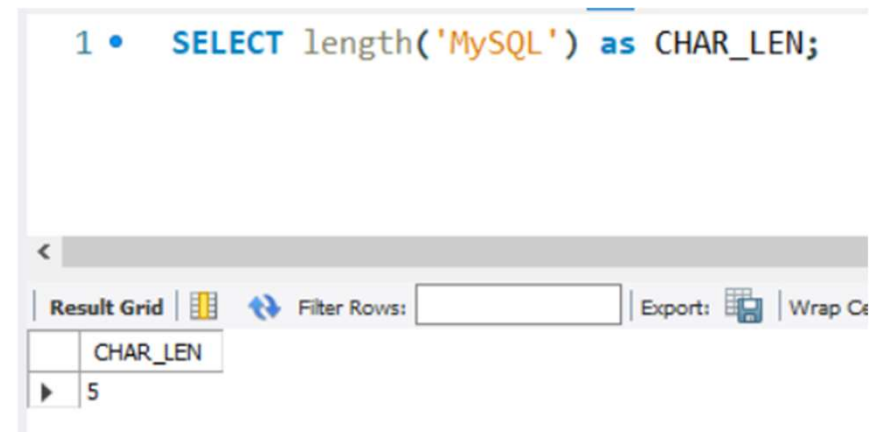
LENGTH():A word's length can be determined using this function.

Syntax:

```
SELECT length('MySQL') as  
CHAR_LEN;
```

Output:

| CHAR_LEN |
|----------|
| 5 |



String function - Example 2

CONCAT(): This function connects multiple strings into one string.

Syntax:

```
SELECT CONCAT( 'great',  
'learning') as CONCAT_STR;
```

Output:

| CONCAT_STR |
|----------------|
| great_learning |

```
1 • SELECT CONCAT( 'great', 'learning') as CONCAT_STR;
```

| | | | | |
|---------------|--|--------------|---------|--------------------|
| Result Grid | | Filter Rows: | Export: | Wrap Cell Content: |
| CONCAT_STR | | | | |
| greatlearning | | | | |

String function – Example 3

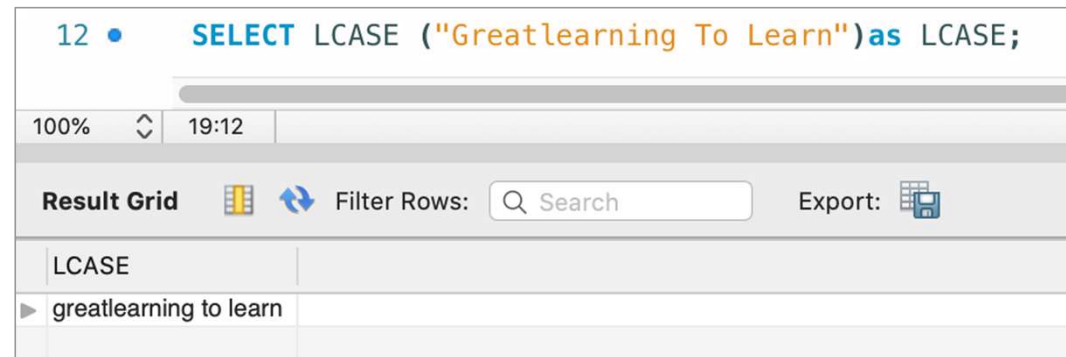
LCASE(): The function converts a string to lowercase.

Syntax:

```
SELECT LCASE ("Greatlearning To  
Learn") as LCASE;
```

Output:

| |
|------------------------|
| LCASE |
| greatlearning to learn |



The screenshot shows a SQL query editor with the following SQL statement: `SELECT LCASE ("Greatlearning To Learn") as LCASE;`. The query is executed, and the results are displayed in a table. The table has one column named 'LCASE' and one row with the value 'greatlearning to learn'.

| LCASE |
|------------------------|
| greatlearning to learn |

String function - Example 4

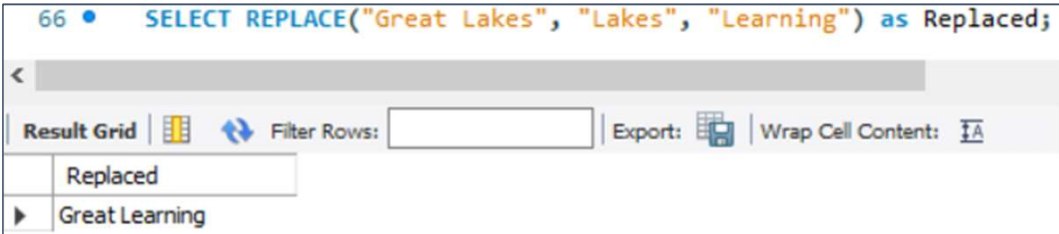
REPLACE(): This function shows all occurrences of one substring in a string should be replaced with another substring

Syntax:





```
SELECT REPLACE("Great Lakes",  
"Lakes", "Learning") as Replaced;
```

Output:

| |
|----------------|
| Replaced |
| Great Learning |



The screenshot shows a SQL query editor with the following SQL statement: `66 • SELECT REPLACE("Great Lakes", "Lakes", "Learning") as Replaced;`

Below the query, there is a toolbar with options: **Result Grid**, , , **Filter Rows:** , **Export:** , and **Wrap Cell Content:** .

The output is displayed in a table with two rows:

| | Replaced |
|---|----------------|
| ▶ | Great Learning |

String function - Example 5

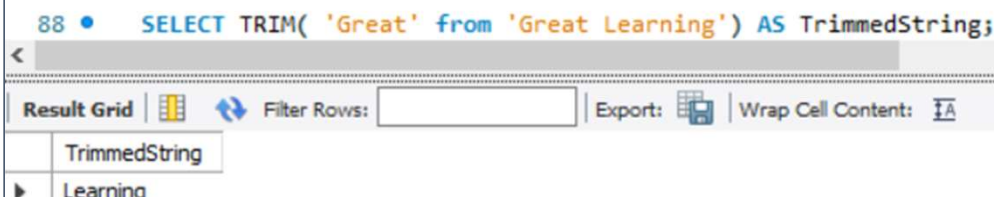
TRIM(): The function removes unwanted characters from a string.

Syntax:

```
SELECT TRIM( 'Great' from 'Great Learning') AS TrimmedString;
```

Output:

| |
|---------------|
| TrimmedString |
| Learning |



The screenshot shows a SQL query editor with the following SQL statement: `SELECT TRIM('Great' from 'Great Learning') AS TrimmedString;`. Below the query, there is a toolbar with options: "Result Grid", "Filter Rows:", "Export:", and "Wrap Cell Content:". The output is displayed in a table with two columns: "TrimmedString" and "Learning".

| | TrimmedString |
|---|---------------|
| ▶ | Learning |

String function - Example 6

SUBSTR(): A substring is extracted from a string using this function.

Syntax:

```
SELECT SUBSTR("Great Learning", 1,  
5) AS ExtractedString;
```

Output:

| ExtractedString |
|-----------------|
| Great |

87 • `SELECT SUBSTR("Great Learning", 1, 5) AS ExtractedString;`

| Result Grid | Filter Rows: | Export: | Wrap Cell Content: |
|-----------------|--------------|---------|--------------------|
| ExtractedString | | | |
| ▶ Great | | | |



Summary

This slide explains the multiple string functions including the basic like CONCAT and REPLACE of string values using the queries.

The syntax of the query is also explained, as well as how to use the functions in the query.




SQL Built-in functions



Agenda

- SQL Built-in functions
- Numeric Functions
- More Numeric Functions and Examples



SQL Built-in function

- MySQL ships with built-in functions.
- Built-in functions are categorized by the type of data they operate on, i.e. strings, dates, and numerics.



SQL Built-in function

- There are different types of SQL built - in functions, which are mentioned below:
 - String
 - Numeric
 - Date

SQL Built-in function

- We use the following Employee table to understand the built-in function:

| ID | NAME | AGE | ADDRESS | SALARY |
|----|--------|-----|------------|--------|
| 1 | Kellie | 32 | California | 2000 |
| 2 | Pete | 25 | Texas | 1500 |
| 3 | Popy | 23 | Boston | 2000 |
| 4 | Sam | 25 | Florida | 6500 |
| 5 | Jhon | 27 | Hawaii | 10000 |



Numeric functions

Numeric function - Syntax

- To perform any mathematical operation in a query, use the following syntax.

Syntax:

```
SELECT numerical_expression as OPERATION_NAME  
[FROM table_name WHERE CONDITION] ;
```


Numeric function - Example 1

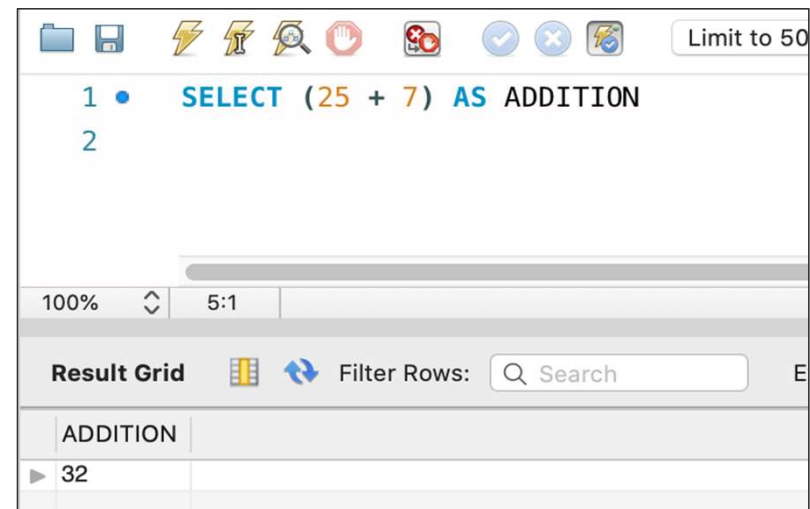
Numerical expressions are used for expressing mathematical expressions and formulas. The following example illustrates how to use SQL numeric expressions .

Syntax:

```
SELECT (25 + 7) AS ADDITION
```

Output:

| ADDITION |
|----------|
| 32 |



Numeric function - Example 2

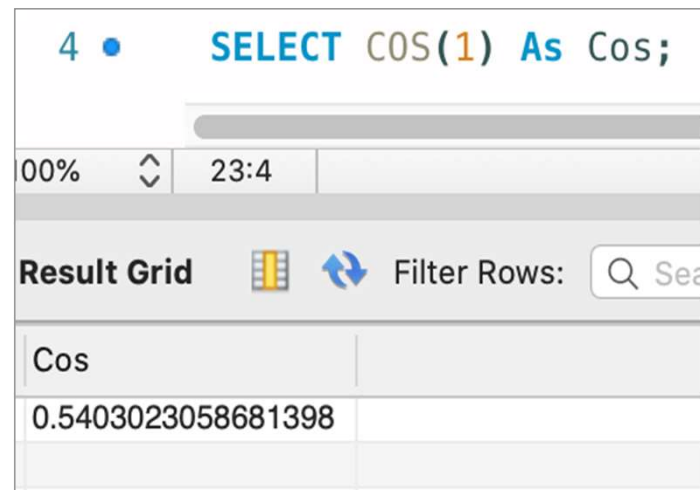
numerical_expressions must contain a mathematical expression or formula. The following example illustrates the use of numerical expressions in SQL.

Syntax:

```
SELECT COS (1) As Cos;
```

Output:

| Cos |
|--------------------|
| 0.5403023058681398 |



The screenshot shows a SQL query editor with the query `SELECT COS(1) As Cos;` entered. Below the query, a progress bar indicates 00% completion and a timer shows 23:4. The results are displayed in a 'Result Grid' with a search bar. The grid contains one column named 'Cos' and one row with the value '0.5403023058681398'.

| Cos |
|--------------------|
| 0.5403023058681398 |

Numeric function - Example 3

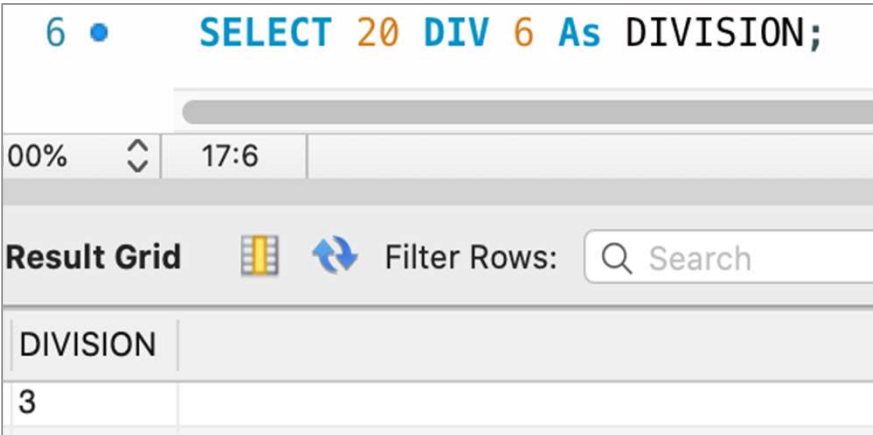
numerical_expressions must contain a mathematical expression or formula. The following example illustrates the use of numerical expressions in SQL.

Syntax:

```
SELECT 20 DIV 6 As DIVISION;
```

Output:

| DIVISION |
|----------|
| 3 |



The screenshot shows a SQL query editor with a query bar containing the text "6 • SELECT 20 DIV 6 As DIVISION;". Below the query bar is a progress bar at 00% and a timer at 17:6. The "Result Grid" section displays the query results in a table with one column named "DIVISION" and one row containing the value "3". A "Filter Rows" search bar is also visible.

| DIVISION |
|----------|
| 3 |

More numeric functions

- Following are the numeric functions defined in SQL:

| Functions | Meaning |
|----------------|---|
| ABS | Provides absolute value of a number |
| ACOS | Provides the cosine of a number |
| ASIN | Provides the arc sine of a number |
| ATAN | It provides the arc sine of a numerical value. |
| CEIL | Provides the smallest integer value that is greater than or equal to a number |
| CEILING | Provides the smallest integer value that is greater than or equal to a number |

More numeric functions

| Functions | Meaning |
|----------------|---|
| COS | Provides the cosine of a number |
| COT | Provides the cotangent of a number |
| DEGREES | Provides a radian value into degrees |
| DIV | Generally used for the integer division |
| EXP | The result is e raised to the power of the number |
| FLOOR | Provides the largest integer value that is less than or equal to a number |

More numeric functions

| Functions | Meaning |
|-----------------|---|
| GREATEST | The greatest value in a list of expressions is returned |
| LEAST | The smallest value in a list of expressions is returned |
| LN | The natural logarithm of a number is returned |
| LOG10 | The base-10 logarithm of a number is returned |
| LOG2 | The base-2 logarithm of a number is returned |
| MOD | The remainder of n divided by m is returned |

More numeric functions

| | Meaning |
|----------------|--|
| PI | It gives the value of PI (up to six decimal places). |
| POW | It gives m^n . |
| RADIANS | It changes the degrees into radians. |
| RAND | It returns a random number |
| ROUND | It gives the rounded number to the specific decimal place. |
| SIGN | An indication of a number's sign is returned. |

More Numeric Functions

| Functions | Meaning |
|-----------------|---|
| SIN | The sine of a number is returned |
| SQRT | The square root of a number is returned |
| TAN | The tangent of a number is returned |
| ATAN2 | The arctangent of the x and y coordinates, as an angle and expressed in radians is returned |
| TRUNCATE | This doesn't work for SQL Server. It returns 7.53635 truncated to 2 places right of the decimal point |



Summary

This slide explains the multiple numeric functions including the basic like addition and subtraction of values using the queries

The syntax of the query is also explained, as well as how to use the functions in the query.



SQL Built-in functions



Agenda

- SQL Built-in functions
- CAST Function
- COALESCE Function
- Sorting Query Result



CAST function

CAST function - Syntax

- CAST() → This function converts any type of value into the specified datatype.

Syntax:

```
CAST (value AS datatype)
```

Argument:

| Parameter | Description |
|-----------|---|
| value | Required. The selected value we want to convert |
| datatype | Required. The datatype to convert to |

CAST function - Example 1

- Converting a value to a CHAR data type:

Syntax:

```
SELECT CAST(150 AS CHAR) ;
```

Output:

| |
|--------------------------|
| Number of Records: 1 |
| CAST(150 AS CHAR) |
| 150 |

CAST function - Example 2

- Converting a value to a TIME datatype:

Syntax:

```
SELECT CAST ("14:06:10" AS TIME) ;
```

Output:

| |
|---------------------------------|
| Number of Records: 1 |
| CAST("14:06:10" AS TIME) |
| 14:06:10 |



COALESCE function

COALESCE function - Syntax

- COALESCE() → Provides first non-null values in the list.

Syntax:

```
COALESCE(value1, value2, ..., value_n)
```

Parameter Values:

| Parameter | Description |
|--|--|
| <i>value1</i> , <i>value2</i> , till <i>value_n</i> | It is required because these are the values we want to test. |

COALESCE function - Example

- Return the first non-null value in a list

```
SELECT COALESCE(NULL, 1, 2, 'greatlearning.in');
```

Output:

| |
|---|
| Number of Records: 1 |
| COALESCE(NULL, 1, 2, 'greatlearning.in') |
| 1 |

The first value after the NULL value is '1', so the output is 1



Sorting Query result

Sorting Query results

Consider the following table for the study:

| UserID | Name | City | Salary |
|--------|-------------------|-----------|--------|
| 1 | Atanu Chakraborty | Kolkata | 15000 |
| 2 | Hriday Bharadwaj | Lucknow | 27000 |
| 3 | Birendra Maity | Hyderabad | 68000 |
| 4 | Raj Patil | Bangalore | 71000 |
| 22 | Bikram Ganguly | Kolkata | 51000 |
| 31 | Tapan Chatterjee | Kolkata | 25000 |
| 33 | Jai Prakash Bhatt | Jaipur | 81000 |
| 34 | Keith Kosta | Mumbai | 77000 |
| 35 | Mark Tailor | Bangalore | 52000 |
| 36 | Christopher Swan | Null | 98000 |
| 39 | Paul Grant | Mumbai | 67000 |

Sorting Query results

- Using the **order by** clause, the tuples in the result of a query are sorted.
- Let us write a query which will show the users according to the ascending order of their salary.

```
SELECT UserID, Name, Salary FROM tblUser ORDER BY Salary
```

Output:

| UserID | Name | Salary |
|--------|-------------------|--------|
| 1 | Atanu Chakraborty | 15000 |
| 31 | Tapan Chatterjee | 25000 |
| 2 | Hriday Bharadwaj | 27000 |
| 22 | Bikram Ganguly | 51000 |
| 35 | Mark Tailor | 52000 |
| 39 | Paul Grant | 67000 |
| 3 | Birendra Maity | 68000 |
| 4 | Raj Patil | 71000 |
| 34 | Keith Kosta | 77000 |
| 33 | Jai Prakash Bhatt | 81000 |
| 36 | Christophar Swan | 98000 |

The **order by** clause lists items in ascending order by default.

Sorting Query results

- To specify the sort order we used: **desc** → descending order or **asc** → ascending order.
- Now let us see an example to show the users according to the descending order of their salary.

```
SELECT UserID, Name, Salary FROM tblUser ORDER BY Salary desc
```

Output:

| UserID | Name | Salary |
|--------|-------------------|--------|
| 36 | Christophar Swan | 98000 |
| 33 | Jai Prakash Bhatt | 81000 |
| 34 | Keith Kosta | 77000 |
| 4 | Raj Patil | 71000 |
| 3 | Birendra Maity | 68000 |
| 39 | Paul Grant | 67000 |
| 35 | Mark Tailor | 52000 |
| 22 | Bikram Ganguly | 51000 |
| 2 | Hriday Bharadwaj | 27000 |
| 31 | Tapan Chatterjee | 25000 |
| 1 | Atanu Chakraborty | 15000 |

Sorting Query results

- Furthermore ordering can be performed on multiple attributes.
- Suppose we wish to order the entire tblUser relation in ascending order of City and then descending order of amount. Then we express this query in SQL as follows.

```
SELECT UserID, Name, Salary FROM tblUser ORDER BY City desc, Salary asc
```

Output:

| UserID | Name | City | Salary |
|--------|-------------------|-----------|--------|
| 39 | Paul Grant | Mumbai | 67000 |
| 34 | Keith Kosta | Mumbai | 77000 |
| 2 | Hriday Bharadwaj | Lucknow | 27000 |
| 1 | Atanu Chakraborty | Kolkata | 15000 |
| 31 | Tapan Chatterjee | Kolkata | 25000 |
| 22 | Bikram Ganguly | Kolkata | 51000 |
| 33 | Jai Prakash Bhatt | Jaipur | 81000 |
| 3 | Birendra Maity | Hyderabad | 68000 |
| 35 | Mark Tailor | Bangalore | 52000 |
| 4 | Raj Patil | Bangalore | 71000 |
| 36 | Christophar Swan | Null | 98000 |



Summary

This slide explains the cast function and the coalesce function with its usage in queries by following certain examples.

Sorting queries is also explained by using the orderby clause.

The syntax of the query is also explained, as well as how to use the functions in the query.



Aggregate functions



Agenda

- Aggregate functions
- CREATE Table
- Count Function
- Sum Function
- Average Function
- Minimum Function
- Maximum Function
- Grouped Queries
- Aggregation with Group-by Clause



Aggregate functions

- In aggregate functions, multiple rows of a single column of a table are analyzed and a single result is returned.
- The five (5) aggregate functions defined by ISO standards are as followed.
 - COUNT
 - SUM
 - AVG
 - MIN
 - MAX



Why use Aggregate functions?

- Our database can be easily summarized using aggregate functions.
- For instance, from our company database , management may require following reports:
 - Minimum salary of a particular department
 - Highest paid employee details
 - Average salary of HR department

Create Table

- Firstly, let's take a look at a sample data table for demonstration purposes before we go through each of the functions one by one.

```
CREATE TABLE employee (month INT, emp_id INT, emp_name  
VARCHAR(15), dept_name VARCHAR(15), salary INT );
```

```
INSERT INTO employee VALUES  
(2, 201, 'Ajit', 'HR' , 7000),  
(2, 202, 'Samar', 'IT', 9000),  
(5, 203, "Harish", "HR", 30000),  
(7, 204, "Jagdish", "IT", 120123),  
(7, 205, "Jesse", "SALES", 7000),  
(19,206, "Navin", "SALES", 12000),  
(19,207, "Chatur", "IT", 135656),  
(Null, 208, "Rohit", "IT", 40500);
```

```
select * from employee;
```

Create Table

- The *employee* table created looks as follows:

```
$sqlite3 database.sdb < main.sql
2|201|Ajit|HR|7000
2|202|Samar|IT|9000
5|203|Harish|HR|30000
7|204|Jagdish|IT|120123
7|205|Jesse|SALES|7000
19|206|Navin|SALES|12000
19|207|Chatur|IT|135656
|208|Rohit|IT|40500
```



COUNT

COUNT function - Syntax

- To get a count of total records matching a condition, call the COUNT function.

Syntax:

```
SELECT COUNT ([DISTINCT] field_name) FROM target_table [WHERE test_expr];
```

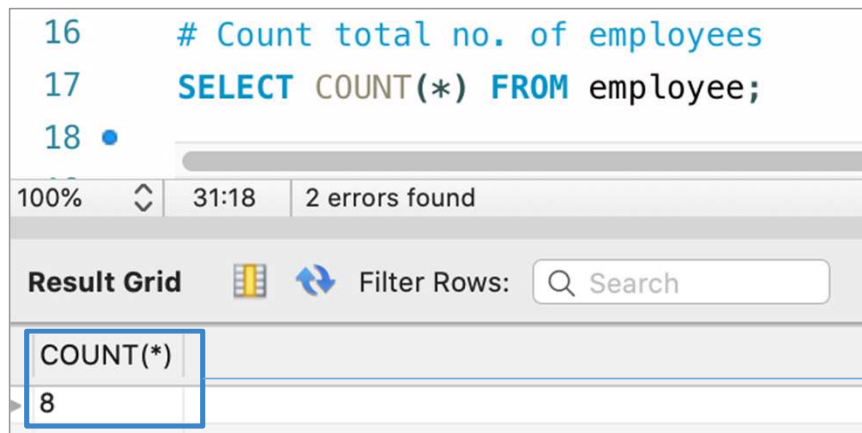
- **COUNT(DISTINCT field_name)** returns the number of distinct rows that are not NULL as a result of the expression.

COUNT function - Example

- If you want to count the total number of employees, you can use the count function as follows:

```
SELECT COUNT (*) FROM employee;
```

Output:



| | |
|---------------------------------|--------------------------------|
| 16 | # Count total no. of employees |
| 17 | SELECT COUNT(*) FROM employee; |
| 18 | • |
| 100% 31:18 2 errors found | |
| Result Grid Filter Rows: Search | |
| COUNT(*) | |
| 8 | |

The total number of employees is 8.



SUM

SUM function - Syntax

- The SUM function gets total a set of values.

Syntax:

```
SELECT SUM(field_name) FROM target_table [WHERE test_expr];
```

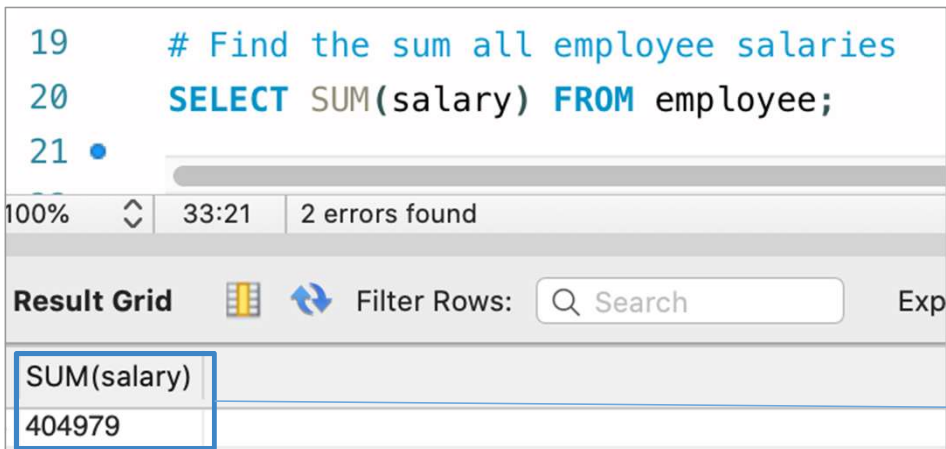
A column or expression will be summed in this field.

SUM function - Example

- Below is the query to find the sum of all employee salaries using sum() function:

```
SELECT SUM(salary) FROM employee;
```

Output:



The screenshot shows a SQL IDE interface. The top pane contains a SQL query: `# Find the sum all employee salaries` followed by `SELECT SUM(salary) FROM employee;` on lines 19, 20, and 21. The bottom pane shows the 'Result Grid' with a single row containing the value '404979' under the column header 'SUM(salary)'. The interface includes a status bar at the top of the result grid showing '100%', '33:21', and '2 errors found'. There are also icons for 'Result Grid', a refresh button, and a search bar.

| SUM(salary) |
|-------------|
| 404979 |

The total sum of salary of all the employees is 404979.



AVERAGE (AVG)

AVG function - Syntax

- The AVG function returns the average of a set of values.

Syntax:

```
SELECT AVG(field_name) FROM target_table [WHERE test_expr];
```



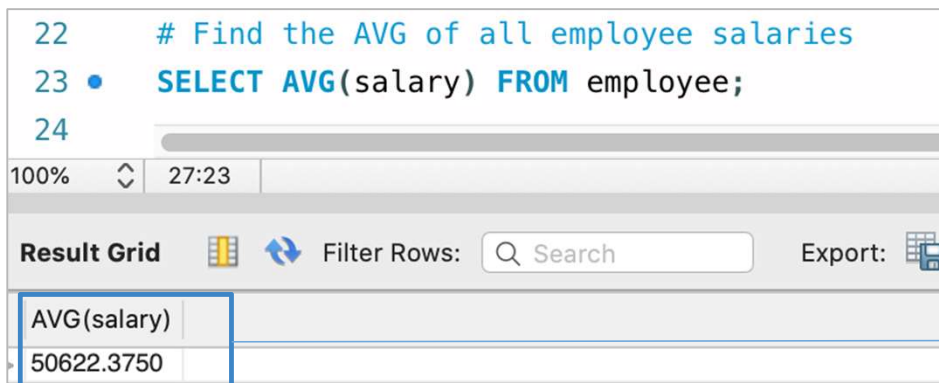
The average will be based on this column or expression.

AVG function - Example

- Find the average of all employee salaries, using the AVG function as follows:

```
SELECT AVG(salary) FROM employee;
```

Output:



22 # Find the AVG of all employee salaries
23 • **SELECT AVG(salary) FROM employee;**
24

100% 27:23

Result Grid Filter Rows: Search Export:

| AVG(salary) |
|-------------|
| 50622.3750 |

The average salary of all employees is 50622.3750.



MINIMUM (MIN)

MIN function - Syntax

- The MIN function returns the minimum from a set of value.

Syntax:

```
SELECT MIN(field_name)FROM target_table [WHERE test_expr];
```

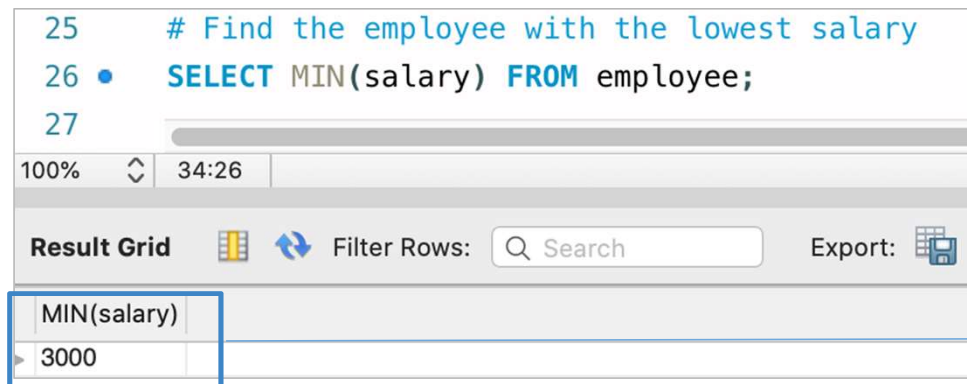
This is the column or expression that will give the minimum value of specific column.

MIN function - Example

- Find the lowest salary received by an employee using the MIN function as follows:

```
SELECT MIN(salary) FROM employee;
```

Output:



25 # Find the employee with the lowest salary
26 • SELECT MIN(salary) FROM employee;
27

100% 34:26

Result Grid Filter Rows: Search Export:

| MIN(salary) |
|-------------|
| 3000 |

The minimum salary of the employee is 3000.



MAXIMUM (MAX)

MAX function - Syntax

- The MAX function returns the maximum from a set of values.

Syntax:

```
SELECT MAX(field_name)FROM target_table[WHERE test_expr];
```

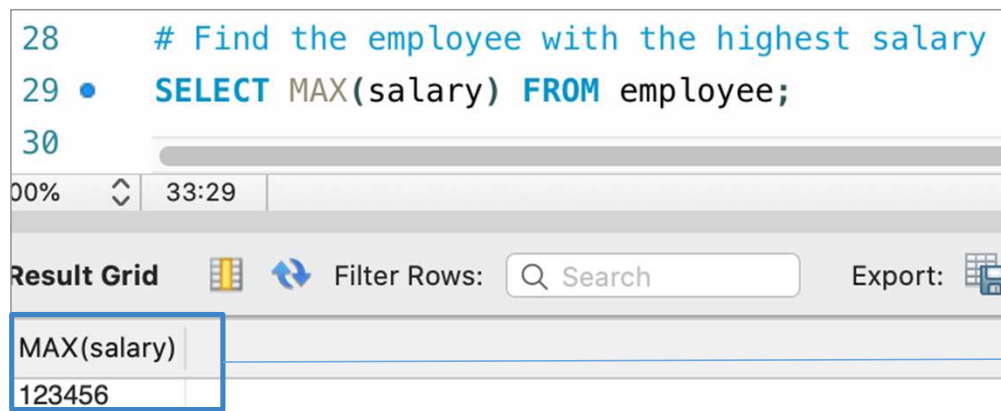
The maximum value is found in this column or expression.

MAX function - Example

- Find the highest salary received by an employee using the MAX function as follows:

```
SELECT MAX(salary) FROM employee;
```


Output:



The screenshot shows a SQL IDE interface. The top pane contains a SQL query: `# Find the employee with the highest salary` followed by `SELECT MAX(salary) FROM employee;` on line 29. Below the query editor, a status bar shows '00%' zoom, a refresh icon, and a timer '33:29'. The bottom pane is titled 'Result Grid' and contains a single row with the column header 'MAX(salary)' and the value '123456'. A blue box highlights the value '123456' in the result grid, and a blue arrow points from this box to a text box on the right.

| MAX(salary) |
|-------------|
| 123456 |

The maximum salary of the employee is 123456.



Do it Yourself

The aggregate function discussed so far returns zero when no matching rows exist in the table.



Grouped Queries

Group by function - Syntax

- GROUP BY → A summary row is created by grouping rows with the same values.

Syntax:

```
SELECT statements... GROUP BY column_name1[,column_name2,...] [HAVING  
condition];
```

"[HAVING condition]" Specifies the rows that will be affected by the GROUP BY clause; it is optional.

"GROUP BY" → Using column_name1, column_name1 performs the grouping. In the event that more than one column is grouped, "[column_name2,...]" represents other column names.

Grouping using Single Column

- Execute a simple query that returns all the department entries from the empl table.

```
SELECT dept_name from employee;
```

Output:

| dept_name | |
|-----------|--|
| HR | |
| IT | |
| HR | |
| IT | |
| SALES | |
| SALES | |
| IT | |
| IT | |

Gives the single group of dept_name column

Grouping using Single Column

- A GROUPBY function to display unique departments in the office:

```
SELECT dept FROM empl GROUP BY dept;
```

Output:

| Result Grid | |
|-------------|--|
| dept_name | |
| HR | |
| IT | |
| SALES | |

There are 3 unique departments, namely HR, IT, and SALES.



Aggregation with Group by Clause

Count Aggregation - Group By

- Count the number of employees, in each department, using the Group By clause along with the count aggregate function as follows:

```
SELECT COUNT(*), dept_name FROM employee GROUP BY dept_name;
```

Output:

| COUNT(*) | dept_name |
|----------|-----------|
| 2 | HR |
| 4 | IT |
| 2 | SALES |

Out of 8 employees, 2 employees belong to HR department, 4 belongs to IT department and 2 employees belongs to SALES department.

SUM Aggregation - Group By

- Use the sum function to find the sum of salaries in each department as follows:

```
SELECT dept_name, SUM(salary) FROM employee GROUP BY dept_name;
```

Output:

| dept_name | SUM(salary) |
|-----------|-------------|
| HR | 29000 |
| IT | 271979 |
| SALES | 104000 |

SUM Aggregation - Group By

- Find the month-wise sum of salaries using the sum function as follows:

```
SELECT month, SUM(salary) FROM employee GROUP BY month;
```

Output:

| month | SUM(salary) |
|-------|-------------|
| 1 | 17000 |
| 3 | 20000 |
| 6 | 113123 |
| 12 | 224456 |
| NULL | 30400 |

AVG Aggregation - Group By

- Find the average of salaries in each department, using AVG function as follows:

```
SELECT dept_name, AVG(salary) FROM employee GROUP BY dept_name;
```

Output:

| dept_name | AVG(salary) |
|-----------|-------------|
| HR | 14500.0000 |
| IT | 67994.7500 |
| SALES | 52000.0000 |

AVG Aggregation - Group By

- Find the month-wise average of salaries by using the AVG function as follows:

```
SELECT month, AVG(salary) FROM employee GROUP BY month;
```

Output:

| month | AVG(salary) |
|-------|-------------|
| 1 | 8500.0000 |
| 3 | 20000.0000 |
| 6 | 56561.5000 |
| 12 | 112228.0000 |
| NULL | 30400.0000 |

MIN Aggregation - Group By

- Find the lowest salary in each department, by using the MIN function as follows:

```
SELECT dept_name, MIN(salary) FROM employee GROUP BY dept_name;
```

Output:

| dept_name | MIN(salary) |
|-----------|-------------|
| HR | 9000 |
| IT | 8000 |
| SALES | 3000 |

MIN Aggregation - Group By

- Find the month-wise minimum salary, by using the MIN function as follows:

```
SELECT month, MIN(salary) FROM employee GROUP BY month;
```

Output:

| month | MIN(salary) |
|-------|-------------|
| 1 | 8000 |
| 3 | 20000 |
| 6 | 3000 |
| 12 | 101000 |
| NULL | 30400 |

MAX Aggregation - Group By

- Find the highest salaries in each department using the MAX function as follows:

```
SELECT dept_name, MAX(salary) FROM employee GROUP BY dept_name;
```

Output:

| | dept_name | MAX(salary) |
|---|-----------|-------------|
| ▶ | HR | 20000 |
| | IT | 123456 |
| | SALES | 101000 |



Summary

This deck explains multiple aggregation functions like COUNT,MIN,MAX etc. to use in the queries with multiple examples.

The syntax of the query is also explained, as well as how to use the functions in the query.



Aggregate functions



Agenda

- Aggregate functions
- Multiple Grouping Columns
- Restrictions on Grouped Queries
- Null values in Grouping Columns
- Aggregation with Having Clause
- Restriction on Grouped Search Conditions
- Null Values and Grouped search conditions



Multiple Grouping Columns

Create Table

- Let's first have a sample data table we'll use to demonstrate the usage:

```
CREATE TABLE employee1 (joining_month INT, emp_id INT,  
emp_name VARCHAR(15), dept_name VARCHAR(15), salary INT  
);
```

```
INSERT INTO employee1 VALUES  
(1, 101, "Oliver", "HR", 9000),  
(1, 102, "George", "IT", 8000),  
(1, 103, "Harry", "HR", 20000),  
(3, 104, "Jack", "IT", 110123),  
(6, 105, "Jacob", "SALES", 3000),  
(6, 106, "Noah", "SALES", 101000),  
(3, 107, "Charlie", "IT", 123456),  
(Null, 108, "Robert", "IT", 30400);
```


Create Table

- The *employee1* table created looks as follows:

| joining_month | emp_id | emp_name | dept_name | salary |
|---------------|--------|----------|-----------|--------|
| 1 | 101 | Oliver | HR | 9000 |
| 1 | 102 | George | IT | 8000 |
| 1 | 103 | Harry | HR | 20000 |
| 3 | 104 | Jack | IT | 110123 |
| 6 | 105 | Jacob | SALES | 3000 |
| 6 | 106 | Noah | SALES | 101000 |
| 3 | 107 | Charlie | IT | 123456 |
| NULL | 108 | Robert | IT | 30400 |



Multiple Grouping Columns

The GROUP BY clause can comprise two or more columns, or, put another way, a group can be made up of two or more columns.

Multiple Grouping Columns

- Get sum of salaries and as well as average of all employees in each dept as per the joining month.

```
SELECT dept_name, joining_month, SUM(salary) , AVG(salary)  
FROM employee1 GROUP BY dept_name, joining_month;
```

Output:

| dept_name | joining_month | sum(salary) | avg(salary) |
|-----------|---------------|-------------|-------------|
| HR | 1 | 29000 | 14500.0000 |
| IT | 1 | 8000 | 8000.0000 |
| IT | 3 | 233579 | 116789.5000 |
| SALES | 6 | 104000 | 52000.0000 |
| IT | NULL | 30400 | 30400.0000 |

- Here we are grouping salary data by using multiple columns : dept & joining month



```
Select dept_name, joining_month,  
sum(salary) From employee1 group by  
dept_name;
```



It contains a nonaggregated column 'company.employee.joining_month' that does not rely on columns in the GROUP BY clause, which is incompatible with sql_mode=only_full_group_by

```
Select dept_name, joining_month,  
sum(salary) From employee1 group by  
dept_name , joining_month;
```



| dept_name | joining_month | sum(salary) |
|-----------|---------------|-------------|
| HR | 1 | 29000 |
| IT | 1 | 8000 |
| IT | 3 | 233579 |
| SALES | 6 | 104000 |
| IT | NULL | 30400 |



Group by functions should not be included in the group-by clause

```
Select dept_name, joining_month,  
sum(salary) From employee1 group by  
sum(salary);
```



Error Code: 1056. Can't group on 'sum(salary)'

```
Select dept_name, joining_month,  
sum(salary) From employee1 group by  
dept_name, joining_month;
```



| dept_name | joining_month | sum(salary) |
|-----------|---------------|-------------|
| HR | 1 | 29000 |
| IT | 1 | 8000 |
| IT | 3 | 233579 |
| SALES | 6 | 104000 |
| IT | NULL | 30400 |



Comparison Conditions cannot be included in the group by clause as they cannot act on grouped result set

```
Select dept_name, joining_month,  
sum(salary) From employee1 group by  
sum(salary) > 10000;
```



Error Code: 1111. Invalid use of group function

```
Select dept_name, joining_month,  
sum(salary) From employee1 group by  
dept_name, joining_month > 10000;
```



| dept_name | joining_month | sum(salary) |
|-----------|---------------|-------------|
| HR | 1 | 29000 |
| IT | 1 | 241579 |
| SALES | 6 | 104000 |
| IT | NULL | 30400 |



Some other restriction on Grouped Queries

- WHERE clause with conditions can be issued before the group-by clause in order to filter the records and then apply Group By feature
 - But , WHERE clause should always mention before the GROUP BY
 - Grouping columns should have less unique values.
 - Grouping columns should be primary business entities and facts and should not contain transactional data.
- Ex: dept , month – are less unique and summarizing the results are easy for grouping on these columns

Some other restriction on Grouped Queries

No summarized results

```
Select salary, sum(salary) From  
employee1 group by salary;
```

| salary | sum(salary) |
|--------|-------------|
| 9000 | 9000 |
| 8000 | 8000 |
| 20000 | 20000 |
| 110123 | 110123 |
| 3000 | 3000 |
| 101000 | 101000 |
| 123456 | 123456 |
| 30400 | 30400 |

Accurate summary Results

```
Select dept_name, joining_month,  
sum(salary) From employee1 group by  
dept_name, joining_month ;
```

| dept_name | joining_month | sum(salary) |
|-----------|---------------|-------------|
| HR | 1 | 29000 |
| IT | 1 | 8000 |
| IT | 3 | 233579 |
| SALES | 6 | 104000 |
| IT | NULL | 30400 |



Null values in Grouping Columns

Null values In Grouping Columns

- If joining month of few employees is unknown and NULL exists in the joining_month column, then the salary is still calculated to show aggregate summary of salaries for those NULL values of the joining_month column.

```
Select dept_name , joining_month , sum(salary) From employee1 group by  
dept_name , joining_month;
```

Output:

| dept_name | joining_month | sum(salary) |
|-----------|---------------|-------------|
| HR | 1 | 29000 |
| IT | 1 | 8000 |
| IT | 3 | 233579 |
| SALES | 6 | 104000 |
| IT | NULL | 30400 |

Shows the aggregate summary of salaries for those NULL values of the month.



Aggregation with Having Clause

Aggregation with Having Clause

- Find the department where the collective salary is more than 35000 each using aggregation with having clause as below:

```
Select joining_month, dept_name , sum(salary) From employee1 group by  
joining_month, dept_name having sum(salary) > 35000;
```

Output:

| joining_month | dept_name | sum(salary) |
|---------------|-----------|-------------|
| 3 | IT | 233579 |
| 6 | SALES | 104000 |



Restriction on Grouped Search condition



Restriction on Grouped Search condition

- Having clause is used along with group-by clause in order to apply conditions for the grouped result set.
- Having clause should be enclosed with grouped functions on columns that are issued in the Select query

Restriction on Grouped Search condition

Conditions in having clause should always have at least one grouping function for comparison since it acts on grouped result set.

```
Select dept_name, joining_month, sum(salary), avg(salary) From employee1  
group by dept_name , joining_month having sum(salary) is not null;
```

Output:

| dept_name | joining_month | sum(salary) | avg(salary) |
|-----------|---------------|-------------|-------------|
| HR | 1 | 29000 | 14500.0000 |
| IT | 1 | 8000 | 8000.0000 |
| IT | 3 | 233579 | 116789.5000 |
| SALES | 6 | 104000 | 52000.0000 |
| IT | NULL | 30400 | 30400.0000 |



Null values and Grouped Search condition

Null values and Grouped Search condition

If you want to find full salary details of employee along with the name and month they have joined, where the salary is not a null value.

```
Select joining_month, emp_name , sum(salary) From employee1 group by  
joining_month having sum(salary) is not null;
```

Output:

| joining_month | emp_name | sum(salary) |
|---------------|----------|-------------|
| 1 | Oliver | 9000 |
| 1 | George | 8000 |
| 1 | Harry | 20000 |
| 3 | Jack | 110123 |
| 6 | Jacob | 3000 |
| 6 | Noah | 101000 |
| 3 | Charlie | 123456 |
| NULL | Robert | 30400 |

salary details of employee along
with the name and month they
have joined, where the salary is
not a null value



Summary

This deck explains multiple aggregation functions, restrictions applicable on grouped queries and the null values in the grouping columns.

Each condition is explained using a query and its output.



Thank You