PER SCHOLAS

Lesson 304.5 - Aggregate Functions and Operators

PER SCHOLAS

Section 1

SQL Aggregate Function

Learning Objective:

By the end of this lesson, learners will be able to:

- Describe aggregate functions.
- Demonstrate aggregate functions.

Table of Contents

Overview of Aggregate Function

SQL has many built-in functions, which are broken into many categories. Four common categories are listed below:

- String Functions.
- Numeric/Math Functions.
- Date and Time Functions.
- Advanced Functions.

Note: We cannot use aggregate functions within the WHERE clause.

Example: `MAX(), Min(), Count(), ASCII(), CHAR_LENGTH(), CHARACTER_LENGTH (), CONCAT(), FORMAT(), INSERT(), LCASE(), LENGTH(), LOCATE(), LOWER(), LPAD(), LTRIM(), MID(), POSITION(), REPEAT, REPLACE(), REVERSE(), RIGHT(), RPAD(), SUBSTR(), TRIM(), UPPER(), SUM() Function and more`

Reference to below link for all functions:

- https://www.techonthenet.com/mysql/functions/index_alpha.php
- https://dev.mysql.com/doc/refman/8.0/en/built-in-function-reference.html

String Functions

Let's demonstrate the most commonly used SQL String functions that allow you to manipulate character string data effectively.

String Functions → REPLACE() Function

- SQL provides you with a useful string function called **REPLACE()**, that allows you to replace a string in a column of a table by a new string or replaces all occurrences of a substring within a string, with a new substring.
- **Syntax of the REPLACE() function is: REPLACE(str,old_string,new_string);**
  - The **REPLACE()** function has three parameters. It replaces the old_string by the new_string in the string.
- The **REPLACE()** function is very handy to search and replace text in a table such as updating obsolete URL or correcting a spelling mistake, etc.

Note: The **REPLACE()** function performs a case-sensitive replacement.

Example:

```
Select REPLACE('abc.org', 'abc', 'perscholas');
#output → perscholas.org

SELECT REPLACE('abc abc', 'a', 'B');
#output → 'Bbc Bbc'
```

String Functions → TRIM() Function

- The **TRIM()** function removes all specified characters, from either the **beginning** or the **end** of a string.
- **TRIM()** to help you clean up the data. The following illustrates the syntax of the **TRIM()** function.

```
TRIM( [ {BOTH|LEADING|TRAILING}  [removed_character] ]  FROM
Given_String);
```

- The **TRIM()** function provides a number of options. You can use the **LEADING, TRAILING,** or **BOTH** options to explicitly instruct the **TRIM()** function to remove leading, trailing, or both unwanted characters from a string.

Example:

```
SELECT TRIM(LEADING '@' FROM '@perscholas.org@@@');  # Result → perscholas.org@@@
SELECT TRIM(TRAILING '@' FROM '@perscholas.org@@@');  # Result →  @perscholas.org
SELECT TRIM(BOTH '@' FROM '@perscholas.org@@@');      # Result → perscholas.org
SELECT TRIM( '@' FROM '@perscholas.org@@@');          # Result → perscholas.org
SELECT TRIM(LEADING '0'  FROM '000123');              # Result →  '123'
SELECT TRIM(TRAILING '1'  FROM 'Tech1');              # Result →  'Tech'
SELECT TRIM(BOTH  '123'  FROM '123Tech123');          # Result →  'Tech'
```

Note: If you do not specify a value for the first parameter (***LEADING***, ***TRAILING***, ***BOTH***), the TRIM() function will default to **BOTH** and remove *character or string* from both the front and end of *string*.

String Functions COUNT(*) and COUNT(fieldname)

The **COUNT(*)** function returns the total number of records meeting the (optional) WHERE criteria:

Example:

- `SELECT COUNT(*) FROM products;  #Result: 110  (result might be vary)`
- `SELECT COUNT(productLine) FROM products;  # Result: 110`

We get a same result in both above queries.

Note: COUNT(*) is unique among the aggregate functions – it counts NULL values. This is because it counts rows, not fields.

The **COUNT(*)** function is often used with a ***GROUP BY*** clause to return the number of elements in each group. For example, that statement below uses the **COUNT()** function with the GROUP BY clause to return the number of products in each product line:

```
SELECT productLine, COUNT(*) FROM products GROUP BY productLine;
```

productLine	COUNT(*)
Classic Cars	38
Motorcycles	13
Planes	12
Ships	9
Trains	3
Trucks and Buses	11
Vintage Cars	24

String Functions MAX() and MIN()

MIN() and MAX() functions return single values from a recordset. ***Note*** that selecting MIN() or MAX() from an empty resultset returns a **NULL**.

```
SELECT MAX(amount), MIN(amount) FROM payments;

Result:  MAX(amount) = 120166.58
         MIN(amount) = 615.45
```

- The below query uses the **MAX()** function to find the largest payment in 2004:

```
SELECT MAX(amount) as largest_payment_2004 FROM payments WHERE YEAR(paymentDate) = 2004;
#Result  largest_payment_2004  = 116208.40
```

- The below query uses the **MIN()** function to find the lowest buy price of all motorcycles from productline table.

```
SELECT MIN(buyPrice) FROM products WHERE productline = 'Motorcycles';
#Result  MIN(buyPrice) = 24.14
```

String Functions CONCAT() Function

To join two or more strings into one, we can use the **CONCAT()** function, which takes one or more string arguments and concatenates them into a single string. The **CONCAT()** function requires a minimum of one parameter; otherwise, it raises an error.

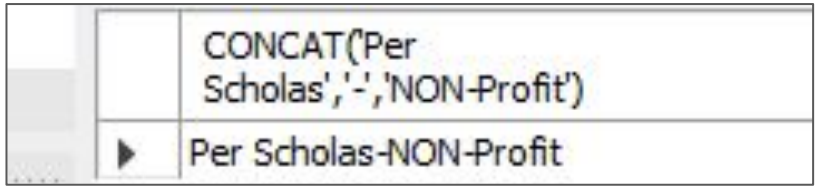
Syntax:

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

Example:

```
SELECT CONCAT('Per Scholas','-','NON-Profit');
```

Result:



	CONCAT('Per Scholas','-','NON-Profit')
▶	Per Scholas-NON-Profit

Example: CONCAT() Function

Consider the **customers** table in the classicmodels database.

To get the full names of contacts, we can use the CONCAT() function to concatenate ***first name, space,*** and ***last name,*** as shown in the below query.

```
SELECT concat(contactFirstName,' ',contactLastName) AS Fullname
FROM customers;
```

customers
* customerNumber
customerName
contactLastName
contactFirstName
phone
addressLine1
addressLine2
city
state
postalCode
country
salesRepEmployeeNumber
creditLimit

Result:

Fullname
Carine Schmitt
Jean King
Peter Ferguson
Janine Labrune
Jonas Bergulfsen
Susan Nelson
Zbyszek Piestrzeniewicz

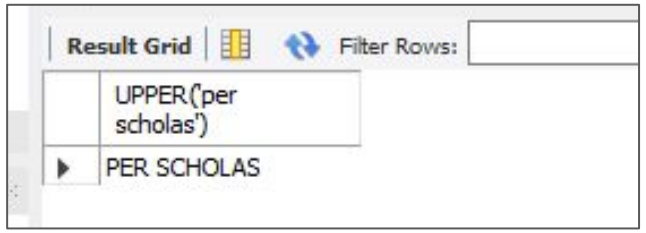
String Functions UPPER() Function

The **UPPER()** function converts all characters in the specified string to uppercase. If there are characters in the string that are not letters, they are unaffected by this function.

Example:

```
SELECT   UPPER('per scholas');
```

Result:



Example: UPPER() Function

Consider the **customer's** table in the classicmodels database. To get the full names of contacts, we can use the **CONCAT()** function to concatenate ***first name, space,*** and ***last name.*** We can convert all characters into the upper letters by using the ***UPPER() function*** as shown in the below query.

customers
* customerNumber
customerName
contactLastName
contactFirstName
phone
addressLine1
addressLine2
city
state
postalCode
country
salesRepEmployeeNumber
creditLimit

```
SELECT  concat(contactFirstName,' ',contactLastName),
UPPER(concat(contactFirstName,' ',contactLastName)) AS Fullname
FROM    customers;
```

Result:

concat(contactFirstName,' ',contactLastName)	Fullname
Carine Schmitt	CARINE SCHMITT
Jean King	JEAN KING
Peter Ferguson	PETER FERGUSON
Janine Labrune	JANINE LABRUNE
Jonas Bergulfsen	JONAS BERGULFSEN
Susan Nelson	SUSAN NELSON
Zbyszek Piestrzeniewicz	ZBYSZEK PIESTRZENIEWICZ
Roland Keitel	ROLAND KEITEL
Julie Murphy	JULIE MURPHY
Kwai Lee	KWAI LEE
Diego Freyre	DIEGO FREYRE
Christina Berglund	CHRISTINA BERGLUND
Jytte Petersen	JYTTE PETERSEN

Numeric/Math Functions

Let's demonstrate the most commonly used SQL Numeric/Math Functions. MySQL Math Functions are the MySQL built-in functions, which refer the numeric type functions and commands to operate the mathematical logics.

Numeric/Math Functions → SUM() Function

The **SUM()** function calculates the sum of a set of values. **Note:** NULL values will be are *ignore* in SUM().

Example 1

```
SELECT SUM(Quantity) AS  TotalItemsOrdered  FROM OrderDetails;
```

 # Above query return the sum of the "Quantity" field in the "OrderDetails" table

Example 2

```
SELECT SUM(amount) AS "Total amount" FROM payments;
```

#Result 8853839.23

Example 3

```
SELECT SUM(quantityOrdered * priceEach) as orderTotal
FROM orderdetails
WHERE orderNumber = 10100;
```

#Result orderTotal = 10223.83

Numeric/Math Functions → AVG() Function

AVG(): Return the average of non-NULL values.

- The below query uses the **AVG()** function to calculate the average buy price of all products from the products table:

```
SELECT AVG(buyprice) as 'Average Price' FROM products;   #Result  54.395182
```

- The following example uses the AVG() function to calculate the average buy price of products in the product line Classic Cars:

```
SELECT    AVG(buyprice) as 'Average Classic Cars Price'
FROM products
WHERE productline = 'Classic Cars';
#Result   Average Classic Cars Price  = 64.446316
```

Note: SUM() and AVG() functions return a value calculated across all rows of a recordset.

Numeric/Math Functions → MOD() Function

- The **MOD()** Function returns the remainder of one number divided by another.
- Syntax: **MOD(dividend,divisor).**
- The **MOD()** function accepts two arguments:
  - **dividend** is a literal number or a numeric expression to divide.
  - **divisor** is a literal number or a numeric expression by which to divide the dividend.
- The **MOD()** function returns the remainder of dividend divided by divisor. If the divisor is zero, the MOD(dividend, 0) returns NULL.

Examples:

```
SELECT MOD(11, 3); #Result: 2
#This above query divides the number 11 by 3. It returns 2 as the integer portion of the result.

SELECT MOD(12, 5); #Result: 2

SELECT MOD(12, 0.18); #Result: 0.12

SELECT MOD( o.quantityOrdered , 2) AS `MOD_DATA`, o.quantityOrdered
FROM orderdetails o;
```

Numeric/Math Functions → ROUND() Function

- The **ROUND()** function allows you to round a number to a specified number of decimal places.
- **Syntax:** ROUND(n,[d]) - In this syntax, ***n*** is a number to be rounded, and ***d*** is the number of decimal places to which the number is rounded. The number of decimal places (d) is optional, it defaults to zero if skipped. The following statements are equivalent:

```
SELECT ROUND(20);        → Result   20
SELECT ROUND(20.5, 0);   → Result   21
```

- The number of decimal places (d) can be positive or negative. If it is negative, the **d** digits left of the decimal point of the number **n** become 0.

Examples:

```
SELECT ROUND(135.375, 2); # Result: 135.38   //Round the number to 2 decimal places:
SELECT ROUND(-125.315);   # Result: -125
SELECT ROUND(121.55,-2);  # Result :  100
SELECT ROUND(121.55,-1);  # Result: 120
SELECT ROUND(125.315, 1); # Result: 125.3 //Round the number to 1 decimal places:
SELECT ROUND(125.315, -2); #  Result: 100
SELECT ROUND(priceEach, 0), priceEach FROM orderdetails;
```

See also the FLOOR, CEIL, CEILING, and TRUNCATE functions.

Numeric/Math Functions → TRUNCATE() Function

- The **TRUNCATE()** Function truncates a number to a specified number of decimal places:
- **Syntax: TRUNCATE(X,D) -** In this syntax:
  - ***X*** is a literal number or a numeric expression to be truncated.
  - ***D*** is the number of decimal places to truncate to. If ***D*** is negative, then the **TRUNCATE()** function causes ***D*** digits left of decimal point of ***X*** to become 0. In case ***D*** is 0, then the return value has no decimal point.
  - **Notice** that the ***TRUNCATE()*** function is similar to the ***ROUND()*** function in terms of reducing the number of decimal places. However, the ***TRUNCATE()*** function does not perform any rounding as the ***ROUND()*** function does.

Examples:

```
SELECT TRUNCATE(125.315, 0);  #  Result: 125
SELECT TRUNCATE(125.315, 1);  #  Result: 125.3
SELECT TRUNCATE(125.315, 2);  #  Result: 125.31
SELECT TRUNCATE(125.315, -1); #  Result: 120
SELECT TRUNCATE(125.315, -2); #  Result: 100
SELECT TRUNCATE(-125.315, 0); #  Result: -125
```

Numeric/Math Functions

Let's demonstrate the most commonly used SQL Date and Time Functions. The SQL date and time functions are responsible for extracting the data section from the specified date or expression “DateTime.”

Formats for Date and Time

Let's view some of the Data Types formats used in SQL for DATE and TIME functions and for storing the values:

- DATE: YYYY-MM-DD
- DATETIME: YYYY-MM-DD HH:MM:SS
- TIMESTAMP: YYYY-MM-DD HH:MM:SS
- YEAR: YYYY or YY

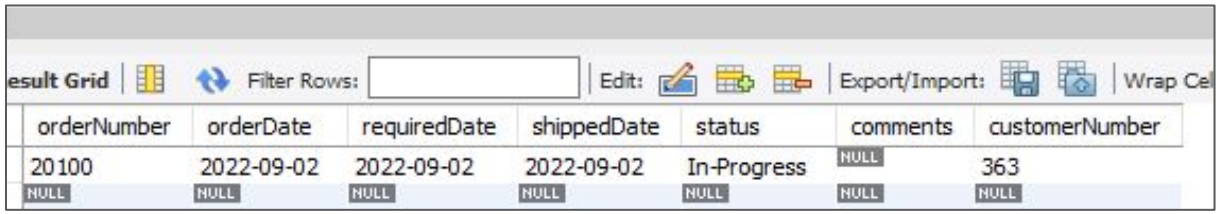
Date Functions → CURRENT_DATE() Function

- The **CURRENT_DATE()** function returns the current date, as a **'YYYY-MM-DD'** format. If used in a string context.
- Syntax for the CURRENT_DATE function in SQL is:
- **CURRENT_DATE() or CURDATE()**

Example:

```
1. SELECT CURRENT_DATE() ;   # Result → 2022-09-01 (Note: Your result will be different.)
2. SELECT CURDATE();         #  Result → 2022-09-01 (Note: Your result will be different.)
3. INSERT INTO orders (orderNumber, orderDate, requiredDate, shippedDate,status,customerNumber)
   VALUES
   (20100, CURRENT_DATE(),CURRENT_DATE(),CURRENT_DATE(),'In-Progress',363 );
```

#Result 3: A new record should be added to the orders table with the current date.



orderNumber	orderDate	requiredDate	shippedDate	status	comments	customerNumber
20100	2022-09-02	2022-09-02	2022-09-02	In-Progress	NULL	363
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Date Functions → CURRENT_TIME() Function

- The **CURRENT_TIME()** function returns the current time.
- The **CURRENT_TIME()** function will return the current date as :
  - **'HH:MM:SS'** format, if used in a string context.
  - **HHMMSS** format, if used in a numeric context.

Examples:

```
SELECT CURRENT_TIME();      #Result: '10:42:14'
SELECT CURRENT_TIME() + 0;  #Result: 104214
SELECT CURRENT_TIME() + 1;  #Result: 104215
SELECT CURRENT_TIME(3);     #Result:15:08:31.500
```

```
Note: Your result will be
different according to the
current date of your computer.
```

Date Functions → CURRENT_TIMESTAMP() Function

- The **CURRENT_TIMESTAMP()** function returns the current date and time.
- The **CURRENT_TIMESTAMP()** function will return the current date as:
  - **'YYYY-MM-DD HH:MM:SS'** format, if used in a string context.
  - **YYYYMMDDHHMMSS** format, if used in a numeric context.

Examples:

```
SELECT CURRENT_TIMESTAMP();  # Result: 15:21:01

SELECT CURRENT_TIMESTAMP(), CURRENT_TIME();
# Result: 2022-09-02 15:21:01  | 15:21:01
```

Note: Your result will be different according to the current date of your computer.

Date Functions →MONTH('date') Function

The **MONTH()** function returns the month portion for a given date (a number from 1 to 12).

Syntax: **MONTH('date')**

Example:

```
SELECT MONTH('2014-01-28');  #Result → 1
SELECT MONTH("2017-06-15 09:34:21");  #Result → 6
SELECT MONTH(CURDATE());
#Result → This would display the month portion of the current system date of your system/computer.
```

Date Functions → DATE_FORMAT() Function

To format a date() value to a specific format, you can use the DATE_FORMAT() function.

Syntax: **DATE_FORMAT(date,format),** function accepts two arguments:

- **date**: is a valid date value that you want to format.
- **format**: is a format string that consists of predefined specifiers.

- Each specifier is preceded by a percentage sign(%). Visit below link for a list of predefined specifiers.
  https://www.techonthenet.com/mysql/functions/date_format.php

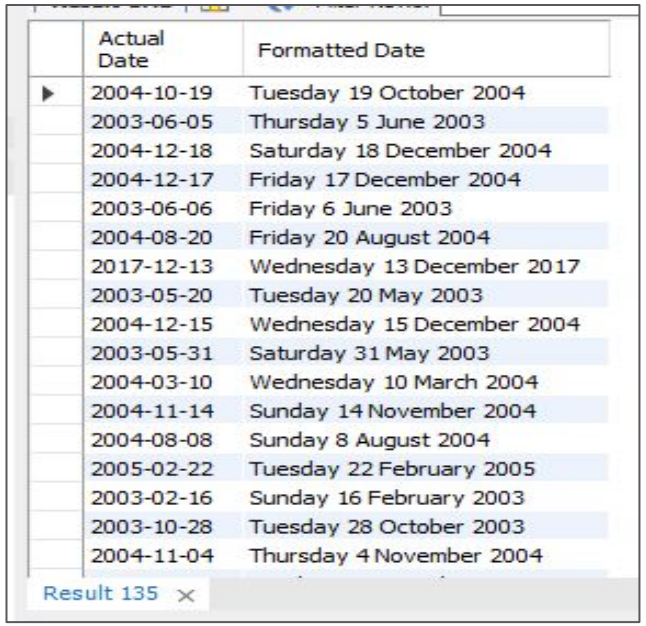
Example:

```
SELECT DATE_FORMAT('2021-04-28', '%Y');                 #Result → 2021
SELECT DATE_FORMAT(current_date(), '%Y'); #Result → 2022  // this would display the year
portion of the current system date of your system/computer
SELECT DATE_FORMAT('2014-02-01', '%M %e %Y');           #Result →  'February 1 2014'
SELECT DATE_FORMAT('2014-02-28', '%W, %M %e, %Y');      #Result →  'Friday, February 28, 2014'
SELECT DATE_FORMAT('2014-02-28', '%W');                 #Result: 'Friday'
```

Example: DATE_FORMAT() Function

```
SELECT
    p.paymentDate AS `Actual Date`,
    DATE_FORMAT(p.paymentDate, '%W %e %M %Y') AS `Formatted Date`
    FROM payments p;
```

Result:



Actual Date	Formatted Date
2004-10-19	Tuesday 19 October 2004
2003-06-05	Thursday 5 June 2003
2004-12-18	Saturday 18 December 2004
2004-12-17	Friday 17 December 2004
2003-06-06	Friday 6 June 2003
2004-08-20	Friday 20 August 2004
2017-12-13	Wednesday 13 December 2017
2003-05-20	Tuesday 20 May 2003
2004-12-15	Wednesday 15 December 2004
2003-05-31	Saturday 31 May 2003
2004-03-10	Wednesday 10 March 2004
2004-11-14	Sunday 14 November 2004
2004-08-08	Sunday 8 August 2004
2005-02-22	Tuesday 22 February 2005
2003-02-16	Sunday 16 February 2003
2003-10-28	Tuesday 28 October 2003
2004-11-04	Thursday 4 November 2004

Result 135

As you can see in the screenshot, We used **paymentDate** column from the payment table. We formatted the date in a more readable format by using the **DATE_FORMAT()** function.

Date Functions → DATEDIFF() Function

- The **DATEDIFF()** function calculates the number of days between two ***DATE***, ***DATETIME***, or ***TIMESTAMP*** values.

  Syntax: **DATEDIFF(date_expression_1, date_expression_2);**

- The **DATEDIFF()** function accepts two arguments that can be any valid date or date-time values. If you pass DATETIME or TIMESTAMP values, the DATEDIFF function only takes the date parts for calculation and ignores the time parts.

- The DATEDIFF() function is useful in many cases e.g., you can calculate an interval in days that the products need to ship to a customer.

Example:

```
SELECT DATEDIFF('2021-01-28', '2021-01-27');  #Result: 1
SELECT DATEDIFF('2021-01-28 11:41:14', '2021-01-27 12:10:08'); #Result: 1
SELECT DATEDIFF('2029-02-15', '2021-02-10'); #Result: 2927
SELECT DATEDIFF('2014-01-28', '2013-12-31'); #Result: 28
SELECT DATEDIFF('2013-12-31', '2014-01-28'); #Result: -28
SELECT DATEDIFF(CURDATE(), '2014-02-14'); # Result: This would display the difference between current system date and '2014-02-14'
```

Example: DATEDIFF() Function

Let's calculate the number of days between the **current date** and the **order received data**. Calculate the number of days between **current date** and **shipping date** of the orders table. We can utilize the ***CURRENT_DATE()*** function and the ***DATEDIFF()*** function.

```
SELECT
   CURRENT_DATE(), orderDate,
   DATEDIFF(CURRENT_DATE() , orderDate) as 'Orders received Days',
   DATEDIFF(CURRENT_DATE() , shippedDate) as 'Order Shipping Days'
from classicmodels.orders
```

Advanced Functions

Let's demonstrate the most commonly used SQL Advanced Functions that allow you to manipulate data effectively.

Advanced Functions → IF() Function

- The **IF()** function returns one value if a condition evaluates to **TRUE**, or another value if it evaluates to **FALSE**.
- Syntax: **IF(condition, [value_if_true], [value_if_false])**

Example:

```
SELECT IF(100<200, 'yes', 'no'); #Result →  'yes'

SELECT IF(500<1000, 5, 10); #Result → 5 //Return 5 if the condition is TRUE, or 10 if the condition is FALSE:

SELECT OrderNumber, quantityOrdered, IF(quantityOrdered>30, "MORE", "LESS") FROM OrderDetails;

# Result → Return "MORE" if the condition is TRUE, or "LESS" if the condition is FALSE:
```

Example: IF() Function

Example 1: Let's utilize the ***OrderDetails*** table from the ***classicmodel*** database. The following query will return **"MORE"** if the number of items ordered is greater than 30, and **"LESS"** if the number of items ordered is less than 30.

```
SELECT OrderNumber, quantityOrdered, IF(quantityOrdered>30, "MORE", "LESS")
FROM classicmodels.OrderDetails;
```

Example 2: The **IF()** function is useful when it combines with an aggregate function. Suppose you want to know how many orders have been shipped and cancelled. You can use the **IF()** function with the **SUM()** function as the following query:

```
SELECT
    SUM(IF(status = 'Shipped', 1, 0)) AS Shipped,
    SUM(IF(status = 'Cancelled', 1, 0)) AS Cancelled
FROM
    classicmodels.orders;
```

Advanced Functions → IFNULL() unction

The **IFNULL()** Function is one of the MySQL control flow functions that accepts two arguments and returns the first argument if it is not NULL. Otherwise, the IFNULL function returns the second argument.

Consider the customer's table. You will notice that column **addressLine2** has some ***NULL*** values. We can set the condition if **addressLine2** is ***NULL***, and then print **addressLine1**.

The query is shown on the next slide.

addressLine1	addressLine2	city	state
54, rue Royale	NULL	Nantes	NULL
8489 Strong St.	NULL	Las Vegas	NV
636 St Kilda Road	Level 3	Melbourne	Victoria
67, rue des Cinquante Otages	NULL	Nantes	NULL
Erling Skakkes gate 78	NULL	Stavern	NULL
5677 Strong St.	NULL	San Rafael	CA
ul. Filtrowa 68	NULL	Warszawa	NULL
Lyonerstr. 34	NULL	Frankfurt	NULL
5557 North Pendale Street	NULL	San Francisco	CA
897 Long Airport Avenue	NULL	NYC	NY
C/ Moralzarzal, 86	NULL	Madrid	NULL
Berguvsvägen 8	NULL	Luleå	NULL
Vinbæltet 34	NULL	Kobenhavn	NULL
2, rue du Commerce	NULL	Lyon	NULL
Bronz Sok.	Bronz Apt. 3...	Singapore	NULL
4092 Furth Circle	Suite 400	NYC	NY
7586 Pompton St.	NULL	Allentown	PA
9408 Furth Circle	NULL	Burlingame	CA
106 Linden Road Sandown	2nd Floor	Singapore	NULL

Advanced Functions → IFNULL() function - (Continued)

```
SELECT customerNumber, addressLine1, addressLine2,
IFNULL(addressLine2, addressLine1) as CustomerAddress
from customers;
```

customerNumber	addressLine1	addressLine2	CustomerAddress
103	54, rue Royale	NULL	54, rue Royale
112	8489 Strong St.	NULL	8489 Strong St.
114	636 St Kilda Road	Level 3	Lev 8489 Strong St.
119	67, rue des Cinquante Otages	NULL	67, rue des Cinquante Otages
121	Erling Skakkes gate 78	NULL	Erling Skakkes gate 78
124	5677 Strong St.	NULL	5677 Strong St.
125	ul. Filtrowa 68	NULL	ul. Filtrowa 68
128	Lyonerstr. 34	NULL	Lyonerstr. 34
129	5557 North Pendale Street	NULL	5557 North Pendale Street
131	897 Long Airport Avenue	NULL	897 Long Airport Avenue
141	C/ Moralzarzal, 86	NULL	C/ Moralzarzal, 86
144	Berguvsvägen 8	NULL	Berguvsvägen 8
145	Vinbæltet 34	NULL	Vinbæltet 34
146	2, rue du Commerce	NULL	2, rue du Commerce
148	Bronz Sok.	Bronz Apt. 3...	Bronz Apt. 3/6 Tesvikiye
151	4092 Furth Circle	Suite 400	Suite 400
157	7586 Pompton St.	NULL	7586 Pompton St.
161	9408 Furth Circle	NULL	9408 Furth Circle
166	106 Linden Road Sandown	2nd Floor	2nd Floor

MD5() Function

- The **MD5()** function Calculates an MD5 128-bit checksum for a string. The value is returned as a hash key (binary string of 32 hex digits), or NULL if the argument was NULL.
- **Syntax:** `MD5(str);`

```
SELECT MD5('MypasswordFor20Dollar$');
```

The above query will return MD5 value of *MypasswordFor20Dollar$*.
The return value is `94ee18f13cd3b9d3243a14177735e9f6.`

CAST() Function

- The **CAST()** function converts a value from one datatype to another datatype.
- **Syntax:** CAST(value AS type).
  - value: The value to convert to another datatype.
  - type: The data type that you wish to convert value to. It can be one of the following:

```
DATE, DATETIME, TIME, CHAR, SIGNED, UNSIGNED, BINARY
```

Example:

```
SELECT CAST('2014-02-28' AS DATE);   #Result: '2014-02-28'
SELECT CAST(125 AS CHAR);            #Result: '125'
SELECT CAST(4-6 AS UNSIGNED);   #Result: 18446744073709551614
 # UNSIGNED  converts a value to the UNSIGNED data type that contains the unsigned
64-bit integer.
SELECT CAST("14:06:10" AS TIME); #Result: 14:06:10
```

SQL Functions

Visit below link for more SQL functions

https://dev.mysql.com/doc/refman/8.0/en/functions.html

Hands-On Lab: Aggregate Function

Complete the lab **GLAB - 304.5.1 - Aggregate Functions**. You can find this Lab on Canvas under Assignment section.

Note: If you have any technical questions while performing the lab activity, ask your instructors for assistance.

Section 2

SQL Operators

Learning Objective:

By the end of this lesson, learners will be able to

- Describe SQL operators, including Logical Operators, Arithmetic Operators, Comparison Operators.
- Explain common SQL operators.

Table of Contents

Overview of SQL Operators

SQL comes with special characters or words to perform certain operations. MySQL Operators are applied to the operands in order to carry out specific operations.

SQL operators common categories are:

- Logical Operators.
- Arithmetic Operators.
- Comparison Operators.

Logical Operators

Some Logical Operators are -

Operator	Description
BETWEEN	It is used to search within a set of values, by the minimum value and maximum value provided.
EXISTS	It is used to search for the presence of a row in a table which satisfies a certain condition specified in the query.
OR	It is used to combine multiple conditions in a statement by using the WHERE clause.
AND	It allows the existence of multiple conditions in an SQL statement by using the WHERE clause.
NOT	It reverses the meaning of the logical operator with which it is used. (Examples: NOT EXISTS, NOT BETWEEN, NOT IN, etc.)
IN	It is used to compare a value in a list of literal values.
ALL	It compares a value to all values in another set of values.
ANY	It compares a value to any value in the list according to the condition specified.
IS NULL	It compares a value with a NULL value.
UNIQUE	It searches for every row of a specified table for uniqueness (no duplicates).

Arithmetic Operators

In SQL, arithmetic operators are used to perform the arithmetic operations as described below:

Operator	Description	Example
+	Addition of two operands	a + b
–	Subtraction of right operand from the left operand	a – b
*	Multiplication of two operands	a * b
/	Division of left operand by the right operand	a / b
%	Modulus – the remainder of the division of left operand by the right	a % b

Comparison Operators

The comparison operators in SQL are used to compare values between operands and return true or false according to the condition specified in the statement.

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

IS NULL and IS NOT NULL Operators

IS Null and IS NOT NULL both are logical operators. To test whether a value is NULL or not, we can use the **IS NULL** operator.

IS NULL Syntax:

```
SELECT column_names FROM table_name  WHERE column_name IS NULL;
```

IS NOT NULL Syntax

```
SELECT column_names FROM table_name WHERE column_name  IS NOT NULL;
```

	category_id	category_name	remarks
▶	1	Comedy	Movies with humour
	2	Romantic	Love stories
	3	Epic	Story acient movies
	4	Horror	NULL
	5	Science Fiction	NULL
	6	Thriller	NULL
	7	Action	NULL

Example: IS NULL Operators

We will use the **customers** table in the classicmodels database for the demonstration. The following query uses the **IS NULL** operator to find customers who do not have a sales representative:

```
SELECT customerName, country, salesrepemployeenumber
FROM   classicmodels.customers
WHERE salesrepemployeenumber IS NULL
ORDER BY customerName;
```

Output

customerName	country	salesrepemployeenumber
ANG Resellers	Spain	NULL
Anton Designs, Ltd.	Spain	NULL
Asian Shopping Network, Co	Singapore	NULL
Asian Treasures, Inc.	Ireland	NULL
BG&E Collectables	Switzerland	NULL
Cramer Spezialit?ten, Ltd	Germany	NULL
Der Hund Imports	Germany	NULL
Feuer Online Stores, Inc	Germany	NULL
Franken Gifts, Co	Germany	NULL

Example: IS NOT NULL Operators

We will use the **customers** table in the classicmodels database for the demonstration. The following query uses the **IS NOT NULL** operator to find customers who do not have a sales representative:

```
SELECT customerName, country, salesrepemployeenumber
FROM classicmodels.customers
WHERE salesrepemployeenumber IS NOT NULL
ORDER BY customerName;
```

Output

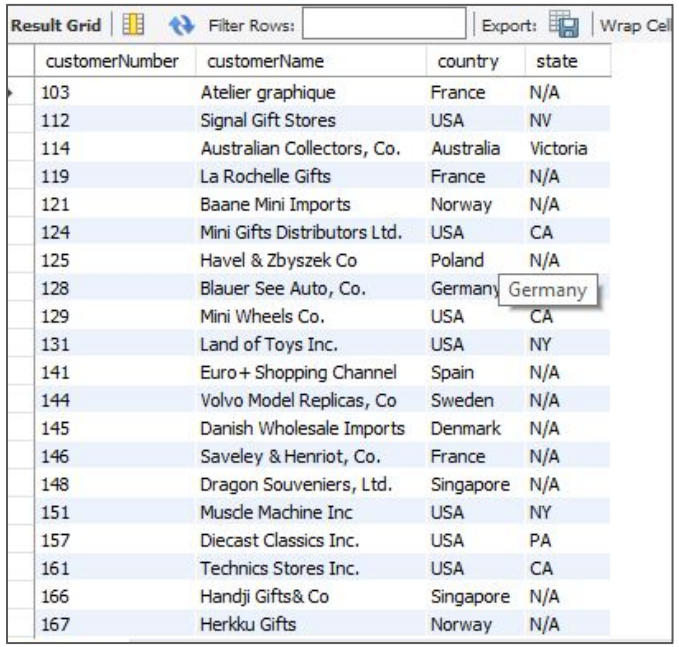
customerName	country	salesrepemployeenumber
Alpha Cognac	France	1370
American Souvenirs Inc	USA	1286
Amica Models & Co.	Italy	1401
Anna's Decorations, Ltd	Australia	1611
Atelier graphique	France	1370
Australian Collectables, Ltd	Australia	1611
Australian Collectors, Co.	Australia	1611
Australian Gift Network, Co	Australia	1611
Auto Associ?s & Cie.	France	1370

Example: IF() function and IS NULL Operator

In this example, we will utilize **if()** function and **IS NULL** operators.

- In the customer's table, many customers do not have state information in the “state“ column; therefore, when we select customers, the state column displays NULL values, which is not meaningful for the reporting purpose. We can improve the output by using the IF function to return N/A if the state is NULL as the following query:

```
SELECT customerNumber, customerName, country,
IF(state IS NULL, 'N/A', state) as state
FROM classicmodels.customers;
```



customerNumber	customerName	country	state
103	Atelier graphique	France	N/A
112	Signal Gift Stores	USA	NV
114	Australian Collectors, Co.	Australia	Victoria
119	La Rochelle Gifts	France	N/A
121	Baane Mini Imports	Norway	N/A
124	Mini Gifts Distributors Ltd.	USA	CA
125	Havel & Zbyszek Co	Poland	N/A
128	Blauer See Auto, Co.	Germany	
129	Mini Wheels Co.	USA	CA
131	Land of Toys Inc.	USA	NY
141	Euro+ Shopping Channel	Spain	N/A
144	Volvo Model Replicas, Co	Sweden	N/A
145	Danish Wholesale Imports	Denmark	N/A
146	Saveley & Henriot, Co.	France	N/A
148	Dragon Souveniers, Ltd.	Singapore	N/A
151	Muscle Machine Inc	USA	NY
157	Diecast Classics Inc.	USA	PA
161	Technics Stores Inc.	USA	CA
166	Handji Gifts& Co	Singapore	N/A
167	Herkku Gifts	Norway	N/A

IN Operator

- The ***IN*** operator allows you to specify multiple values in a WHERE clause.
- The ***IN*** operator allows you to determine if a specified value matches any value in a set of values or is returned by a subquery.

Syntax

Basic ☐ `SELECT column_name(s) FROM table_name WHERE column_name IN (value1,value2,...);`

Advance ☐ `SELECT column_name(s) FROM table_nam e WHERE column_name IN (SELECT STATEMENT);`

Example:

```
SELECT * FROM Customers WHERE country IN ('USA ', 'France ');

#You can achieve the same result with the OR operator as the following query:

SELECT * FROM Customers WHERE country = 'USA' OR country = 'France';
```

We can use the "IN" operator with a subquery. We will explore later in this course.

IN Operator Example

- SELECT * FROM Customers
  WHERE Country **IN** ('Germany', 'France', 'UK');

- SELECT * FROM Customers
  WHERE Country **NOT IN** ('Germany', 'France', 'UK');

Hands-On LAB - Operators

Complete the GLab - 304.5.2 - Operators. You can find this Lab on Canvas under Assignment section.

Note: if you have any technical questions while performing the lab activity, ask your instructors for assistance.

Practice Assignment

This assignment will be administered through HackerRank.

Click on the below links.

- Weather Observation Station - CITY names from STATION
- Weather Observation Station -difference between the total number of CITY entries
- Weather Observation Station - Query the list of CITY names ending with vowels
- Weather Observation Station - Query the list of CITY names from STATION which have vowels
- Weather Observation Station - Query the list of CITY names from STATION that do not start with vowels
- Weather Observation Station - Query the list of CITY names from STATION that either do not start with vowels or do not end with vowels
- Weather Observation Station - Query the sum of Northern Latitudes (LAT_N)
- Weather Observation Station - Query the greatest value of the Northern Latitudes (LAT_N) from STATION that is less than 137.23
- Weather Observation Station - Query the Western Longitude (LONG_W) for the largest Northern Latitude (LAT_N) in STATION that is less than 137.23
- Weather Observation Station - Query the smallest Northern Latitude (LAT_N) from STATION that is greater than 38.77
- Weather Observation Station - Query the Western Longitude (LONG_W)where the smallest Northern Latitude (LAT_N) in STATION is greater than 38.77
- Revising the Select Query I
- Revising the Select Query II
- String function

Note: Use your office hours to complete this assignment, If you have any technical questions while performing the assignment activity, ask your instructors for assistance.

This assignment does not count toward the final grade but these assignments will prepare you for the SBA

Overview of **CASE Statement**

- The **CASE statement** is a Control Flow statement.
- The **CASE statement** chooses from a sequence of conditions and runs the corresponding statement.
- The **CASE statement** has two forms:
  - **Simple -** a single expression is evaluated and compared to potential matches.
  - **Searched -** multiple conditions are evaluated and the first true condition is selected.

Syntax: **Simple *CASE*** Statement

```
CASE given_value
    WHEN condition_value_1 THEN statements_1
    WHEN condition_value_2 THEN statements_2
...
    WHEN condition_value_n THEN statements_n
[ ELSE else_statements ]
END CASE;
```

The ***given_value*** is an expression – usually a scalar variable. Each ***condition_value*** can be either literal or an expression, and all must be of the same data type as the ***given_value***.

The simple CASE statement runs the first statements for which the ***given_value*** equals the ***condition_value***. If no value matches, the *else_statements* will run if they are present.

Syntax: ***Searched CASE*** Statement

```
CASE
    WHEN condition_1 THEN statements_1
    WHEN condition_2 THEN statements_2
    ...
    WHEN condition_n THEN statements_n
    [ ELSE else_statements ]
END CASE;
```

The searched CASE statement executes the first *statements* for which the ***condition*** evaluates **true**. Remaining conditions are not evaluated. If no condition is true, the CASE statement runs the ***else_statements*** if they exist. Otherwise the statement evaluates to **NULL**.

Example 1: ***Simple CASE*** statement

In this example, we will use a simple CASE statement to determine the warehouse based on the ***orderLinenumber*** column from ***orderdetails*** table.

```
SELECT   orderLineNumber,
    CASE od.orderLineNumber
        WHEN 1  THEN 'NYC Warehouse'
        WHEN 2 THEN 'NJ Warehouse'
        WHEN 3 THEN 'CA Warehouse'
        WHEN 4 THEN 'PA Warehouse'
        ELSE   'ML warehouse'
    END AS Product_status
FROM classicmodels.orderdetails od;
```

Result

orderLineNumber	Product_status
3	CA Warehouse
2	NJ Warehouse
4	PA Warehouse
1	NYC Warehouse
4	PA Warehouse
1	NYC Warehouse
3	CA Warehouse
2	NJ Warehouse
2	NJ Warehouse
1	NYC Warehouse
11	ML warehouse
4	PA Warehouse
8	ML warehouse
10	ML warehouse
2	NJ Warehouse
12	ML warehouse
14	ML warehouse
13	ML warehouse
16	ML warehouse
5	ML warehouse
9	ML warehouse

Example 1 - ***Searched Case*** Statements

- In this example, we will use a search CASE Statement to determine the status of the buying price on the ***buyPrice*** column from ***products*** table.

```
SELECT productName, buyPrice,
CASE
     WHEN buyPrice > 9 AND buyPrice <=  50 THEN "LOW PRICE"
     WHEN buyPrice >= 50 AND buyPrice <= 100 THEN "Medium Price"
     WHEN buyPrice > 100 AND buyPrice <= 200 THEN "high Price"
ELSE "Out of our rang" END AS priceStatus
FROM products ORDER BY buyPrice DESC;
```

Result

productName	buyPrice	priceStatus
1962 LanciaA Delta 16V	103.42	high Price
1998 Chrysler Plymouth Prowler	101.51	high Price
1952 Alpine Renault 1300	98.58	Medium Price
1956 Porsche 356A Coupe	98.30	Medium Price
2001 Ferrari Enzo	95.59	Medium Price
1968 Ford Mustang	95.34	Medium Price
1995 Honda Civic	93.89	Medium Price
1970 Triumph Spitfire	91.92	Medium Price
2003 Harley-Davidson Eagle Drag Bike	91.02	Medium Price
1969 Corvair Monza	89.14	Medium Price
1917 Grand Touring Sedan	86.70	Medium Price
1972 Alfa Romeo GTA	85.68	Medium Price
1940s Ford truck	84.76	Medium Price
1993 Mazda RX-7	83.51	Medium Price
1969 Ford Falcon	83.05	Medium Price
18th century schooner	82.34	Medium Price
1958 Setra Bus	77.90	Medium Price

Example 2 - ***Searched Case*** statements

The following example uses the CASE statement to sort customers by states if the state is not NULL, or sort the country in case the state is NULL:

```
SELECT customerName, state, country
FROM customers
ORDER BY ( CASE
    WHEN state IS NULL THEN country
    ELSE state
END);
```

Result

customerName	state	country
Salzburg Collectables	NULL	Austria
Mini Auto Werke	NULL	Austria
Canadian Gift Exchange Network	BC	Canada
Royal Canadian Collectables, Ltd.	BC	Canada
Petit Auto	NULL	Belgium
Royale Belge	NULL	Belgium
Mini Gifts Distributors Ltd.	CA	USA
Mini Wheels Co.	CA	USA
Technics Stores Inc.	CA	USA
Toys4GrownUps.com	CA	USA
Boards & Toys Co.	CA	USA
Collectable Mini Designs Co.	CA	USA
Corporate Gift Ideas Co.	CA	USA
Men 'R' US Retailers, Ltd.	CA	USA

Knowledge Check

- What is an aggregate function?
- Does the COUNT() function return the number of columns in a table?
- Which keyword is used to join multiple strings into a single string?

Summary

SQL offers aggregate functions that can help with the Data computation and manipulation, as seen in the list below. In order to create more sophisticated data manipulation with aggregate functions, you have to use the GROUP BY clause. This groups together all rows that have the same values. You can use the HAVING clause if you need to filter the result of an aggregate function.

SQL Functions

- AVG() - Returns the average value.
- COUNT() - Returns the number of rows.
- FIRST() - Returns the first value.
- LAST() - Returns the last value.
- MAX() - Returns the largest value.
- MIN() - Returns the smallest value.
- SUM() - Returns the sum.UCASE() - Converts a field to uppercase.
- ROUND() - Rounds a numeric field to the number of decimals specified.
- NOW() - Returns the current system date and time.

The case statement in SQL returns a value on a specified condition. We can use a case statement in select queries along with the Where, Order By, and Group By clauses.

References

- https://dev.mysql.com/doc/refman/5.7/en/case.html
- https://dev.mysql.com/doc/refman/5.7/en/functions.html
- https://dev.mysql.com/doc/refman/5.7/en/numeric-functions.html
- https://popsql.com/learn-sql/mysql/how-to-write-a-case-statement-in-mysql

PER SCHOLAS

Questions?