جامعة القاهرة الجديدة التكنولوجية
NEW CAIRO TECHNOLOGICAL UNIVERSITY
كلية تكنولوجيا الصناعة والطاقة

PEARSON BTEC International Standards Verifier

ICT Program

Dr. Amany AbdElSamea Saeed

Lec. 4 MySQL Built-In Functions

March 2024

Outline

- MySQL Functions
- Aggregate Functions
- String Functions
- Control Functions
- Numeric Functions
- Date and Time Functions

MySQL Functions

- Basically, it is a set of SQL statements that accept only input parameters, perform actions and return the result. A function can return only a single value or a table
- Types of Function
  - A scalar function is a function that operates on scalar values -- that is, it takes one (or more) input values as arguments directly and returns a value.

```
--Create function
CREATE FUNCTION CalculateTotal(@Price decimal(10,2),@Quantity int)
RETURNS decimal(10,2)
AS
BEGIN
    RETURN @Price * @Quantity
END
```

  - An aggregate function is a function that operates on aggregate data -- that is, it takes a complete set of data as input and returns a value that is computed from all the values in the set. E.g. max(), min(), count(), sum(), avg().
  - Control Functions allow us a degree of conditionality when returning result sets

MySQL Aggregate Functions

- An aggregate function performs a calculation on multiple values and returns a single value.

- We mostly use the aggregate functions with SELECT statements in the data query languages.

- **The following are the aggregate functions:**
- AVG() - Returns the average of the non-NULL values in the specified field.
- SUM() - Returns the sum of the non-NULL values in the specified field.
- MIN() - Returns the minimum of the non-NULL values in the specified field.
- MAX() - Returns the maximum of the non-NULL values in the specified field.
- COUNT() - Returns the number of rows containing non-NULL values in the specified field.

Count()

- The COUNT() function returns the number of rows that matches a specified criterion.

COUNT() Syntax:

```
SELECT COUNT (aggregate_expression)
FROM table_name
[WHERE conditions];
```

- aggregate_expression: It specifies the column or expression whose NON-NULL values will be counted

Example

```
mysql> SELECT * FROM employees;
+--------+----------+---------+-------------+---------+
| emp_id | emp_name | emp_age | city        | income  |
+--------+----------+---------+-------------+---------+
|    101 | Peter    |      32 | Newyork     |  200000 |
|    102 | Mark     |      32 | California  |  300000 |
|    103 | Donald   |      40 | Arizona     | 1000000 |
|    104 | Obama    |      35 | Florida     | 5000000 |
|    105 | Linklon  |      32 | Georgia     |  250000 |
|    106 | Kane     |      45 | Alaska      |  450000 |
|    107 | Adam     |      35 | California  | 5000000 |
|    108 | Macculam |      40 | Florida     |  350000 |
+--------+----------+---------+-------------+---------+
8 rows in set (0.01 sec)
```

Calculates the total number of employees name available in the table:

```
mysql> SELECT COUNT(emp_name) FROM employees;
+-----------------+
| COUNT(emp_name) |
+-----------------+
|               8 |
+-----------------+
1 row in set (0.00 sec)
```

Example

Execute the following statement that returns all rows from the employee table and WHERE clause specifies the rows whose value in the column emp_age is greater than 32

```
mysql> SELECT COUNT(*) FROM employees WHERE emp_age>32;
+----------+
| COUNT(*) |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
```

This statement uses the COUNT(distinct expression) function that counts the Non-Null and distinct rows in the column emp_age:

```
mysql> SELECT COUNT(DISTINCT emp_age) FROM employees;
+-------------------------+
| COUNT(DISTINCT emp_age) |
+-------------------------+
|                       4 |
+-------------------------+
1 row in set (0.00 sec)
```

Sum()

- The SUM() function returns the total sum of a numeric column.

SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

Example

```
mysql> SELECT * FROM employees;
+---------+-----------+--------------+--------------+---------------+
| emp_id  | emp_name  | occupation   | working_date | working_hours |
+---------+-----------+--------------+--------------+---------------+
|       1 | Joseph    | Business     | 2020-04-10   |            10 |
|       2 | Stephen   | Doctor       | 2020-04-10   |            15 |
|       3 | Mark      | Engineer     | 2020-04-10   |            12 |
|       4 | Peter     | Teacher      | 2020-04-10   |             9 |
|       1 | Joseph    | Business     | 2020-04-12   |            10 |
|       2 | Stephen   | Doctor       | 2020-04-12   |            15 |
|       4 | Peter     | Teacher      | 2020-04-12   |             9 |
|       3 | Mark      | Engineer     | 2020-04-12   |            12 |
|       1 | Joseph    | Business     | 2020-04-14   |            10 |
|       4 | Peter     | Teacher      | 2020-04-14   |             9 |
+---------+-----------+--------------+--------------+---------------+
10 rows in set (0.00 sec)
```

Execute the following query that calculates the total number of working hours of all employees

```
mysql> SELECT SUM(working_hours) AS "Total working hours" FROM employees;
+---------------------+
| Total working hours |
+---------------------+
|                 111 |
+---------------------+
1 row in set (0.00 sec)
```

Example

MySQL sum() function with WHERE clause

```
mysql> SELECT SUM(working_hours) AS "Total working hours" FROM employees
 WHERE working_hours>=12;
+----------------------+
| Total working hours  |
+----------------------+
|                   54 |
+----------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT emp_id, emp_name, occupation, SUM(working_hours) AS "Total working hours
 FROM employees GROUP BY occupation;
+--------+----------+------------+----------------------+
| emp_id | emp_name | occupation | Total working hours  |
+--------+----------+------------+----------------------+
|      1 | Joseph   | Business   |                   30 |
|      2 | Stephen  | Doctor     |                   30 |
|      3 | Mark     | Engineer   |                   24 |
|      4 | Peter    | Teacher    |                   27 |
+--------+----------+------------+----------------------+
4 rows in set (0.00 sec)
```

AVG()

- The AVG() function returns the average value of a numeric column.

AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

Example

```
mysql> SELECT * FROM employees;
+--------+----------+------------+--------------+---------------+
| emp_id | emp_name | occupation | working_date | working_hours |
+--------+----------+------------+--------------+---------------+
|      1 | Joseph   | Business   | 2020-04-10   |            10 |
|      2 | Stephen  | Doctor     | 2020-04-10   |            15 |
|      3 | Mark     | Engineer   | 2020-04-10   |            12 |
|      4 | Peter    | Teacher    | 2020-04-10   |             9 |
|      1 | Joseph   | Business   | 2020-04-12   |            10 |
|      2 | Stephen  | Doctor     | 2020-04-12   |            15 |
|      4 | Peter    | Teacher    | 2020-04-12   |             9 |
|      3 | Mark     | Engineer   | 2020-04-12   |            12 |
|      1 | Joseph   | Business   | 2020-04-14   |            10 |
|      4 | Peter    | Teacher    | 2020-04-14   |             9 |
+--------+----------+------------+--------------+---------------+
10 rows in set (0.00 sec)
```

```
mysql> SELECT AVG(working_hours) AS Avg_working_hours FROM employees;
```

```
+--------------------+
| Avg_working_hours  |
+--------------------+
|            11.1000 |
+--------------------+
1 row in set (0.00 sec)
```

Example

```
mysql> SELECT AVG(working_hours) AS
Avg_working_hours FROM employees WHERE working_hours>=12;
```

```
+-------------------+
| Avg_working_hours |
+-------------------+
|           13.5000 |
+-------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT emp_name, occupation, AVG(working_hours) AS
Avg_working_hours FROM employees GROUP BY occupation;
```

```
+----------+------------+-------------------+
| emp_name | occupation | Avg_working_hours |
+----------+------------+-------------------+
| Joseph   | Business   |           10.0000 |
| Stephen  | Doctor     |           15.0000 |
| Mark     | Engineer   |           12.0000 |
| Peter    | Teacher    |            9.0000 |
+----------+------------+-------------------+
4 rows in set (0.00 sec)
```

MIN() and MAX()

- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

Example

```
mysql> SELECT * FROM employees;
+--------+-----------+---------+-------------+---------+
| emp_id | emp_name  | emp_age | city        | income  |
+--------+-----------+---------+-------------+---------+
|    101 | Peter     |      32 | Newyork     |  200000 |
|    102 | Mark      |      32 | California  |  300000 |
|    103 | Donald    |      40 | Arizona     | 1000000 |
|    104 | Obama     |      35 | Florida     | 5000000 |
|    105 | Linklon   |      32 | Georgia     |  250000 |
|    106 | Kane      |      45 | Alaska      |  450000 |
|    107 | Adam      |      35 | California  | 5000000 |
|    108 | Macculam  |      40 | Florida     |  350000 |
|    109 | Brayan    |      32 | Alaska      |  400000 |
|    110 | Stephen   |      40 | Arizona     |  600000 |
|    111 | Alexander |      45 | California  |   70000 |
+--------+-----------+---------+-------------+---------+
```

```
mysql> SELECT MIN(income) AS Minimum_Income FROM employees;
+----------------+
| Minimum_Income |
+----------------+
|          70000 |
+----------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT MAX(income) AS "Maximum Income" FROM employees;
+----------------+
| Maximum Income |
+----------------+
|        5000000 |
+----------------+
1 row in set (0.00 sec)
```

Example

```
mysql> SELECT * FROM employees;
+--------+-----------+---------+-------------+---------+
| emp_id | emp_name  | emp_age | city        | income  |
+--------+-----------+---------+-------------+---------+
|    101 | Peter     |      32 | Newyork     |  200000 |
|    102 | Mark      |      32 | California  |  300000 |
|    103 | Donald    |      40 | Arizona     | 1000000 |
|    104 | Obama     |      35 | Florida     | 5000000 |
|    105 | Linklon   |      32 | Georgia     |  250000 |
|    106 | Kane      |      45 | Alaska      |  450000 |
|    107 | Adam      |      35 | California  | 5000000 |
|    108 | Macculam  |      40 | Florida     |  350000 |
|    109 | Brayan    |      32 | Alaska      |  400000 |
|    110 | Stephen   |      40 | Arizona     |  600000 |
|    111 | Alexander |      45 | California  |   70000 |
+--------+-----------+---------+-------------+---------+
```

```
mysql> SELECT MIN(income) AS Minimum_Income
    -> FROM employees
    -> WHERE emp_age>=32 AND emp_age<=40;
+----------------+
| Minimum_Income |
+----------------+
|         200000 |
+----------------+
```

```
mysql> SELECT MAX(income) AS "Maximum_Income"
    -> FROM employees
    -> WHERE emp_age > 35;
+----------------+
| Maximum_Income |
+----------------+
|        1000000 |
+----------------+
```

Control Functions

The control functions that allow us a degree of conditionality when returning result sets

IF() Function

The IF function returns a value **YES** when the given condition evaluates to true and returns a **NO** value when the condition evaluates to false

The IF() function returns a value if a condition is TRUE, or another value if a condition is FALSE.

Syntax

```
IF(condition, value_if_true, value_if_false)
```

Example

```
SELECT IF(200>350,'YES','NO');
```

Output:

```
NO
```

```
SELECT IF(251 = 251,' Correct','Wrong');
```

Output:

```
Correct
```

Example

studentid	firstname	lastname	class	age
1	Rinky	Ponting	12	20
2	Mark	Boucher	11	22
3	Sachin	Tendulkar	10	18
4	Peter	Fleming	10	22
5	Virat	Kohli	12	23
NULL	NULL	NULL	NULL	NULL

```
SELECT lastname,
IF(age>20,"Mature","Immature")
As Result
FROM student;
```

lastname	Result
Ponting	Immature
Boucher	Mature
Tendulkar	Immature
Fleming	Mature
Kohli	Mature

IFNULL() Function

The IFNULL function accepts two expressions, and if the first expression is **not null**, it returns the first arguments. If the first expression is **null**, it returns the second argument. This function returns either string or numeric value, depending on the context where it is used.

Syntax

We can use the IFNULL function with the following syntax:

```
IFNULL (Expression1, Expression2)
```

It returns expression1 when the expression1 is not null. Otherwise, it will return expression2.

Example

```
SELECT IFNULL("Hello", "javaTpoint");
```

Output:

```
Hello
```

```
SELECT IFNULL(NULL,5);
```

```
5
```

Example

studentid	contactname	cellphone	homephone
2	Will Smith	3214356574	NULL
3	Johnsena	NULL	4563480897
4	Peter Joe	NULL	2123157870
5	kelly Bruke	5683128765	NULL
6	Freeda Pinto	4563482354	NULL
NULL	NULL	NULL	NULL

```
SELECT
    contactname, IFNULL(cellphone, homephone) phone
FROM
    student_contact;
```

Output:

When the above MySQL statement runs successfully, it will give the following output.

contactname	phone
Will Smith	3214356574
Johnsena	4563480897
Peter Joe	2123157870
kelly Bruke	5683128765
Freeda Pinto	4563482354

NULLIF()

The NULLIF function accepts two expressions, and if the first expression is equal to the second expression, it returns the **NULL**. Otherwise, it returns the first expression.

Syntax

We can use the NULLIF function with the following syntax:

```
NULLIF (Expression1, Expression2)
```

It returns Null when expression1 is equal to expression2. Otherwise, it will return expression1.

Example

```
SELECT NULLIF("javaTpoint", "javaTpoint");
```

In the above function, the MySQL statement checks the first expression is equal to the second expression or not. If both expressions are the same, it returns NULL. Otherwise, it will return the first expression.

Output:

```
NULL
```

```
SELECT NULLIF("Hello", "404");
```

The following MySQL statement compares both expressions. If expression1 = expression2, it returns NULL. Otherwise, it will return expression1.

Output:

```
Hello
```

CASE Expression

Syntax

```
CASE value
    WHEN [compare_value] THEN result
    [WHEN [compare_value] THEN result ...]
    [ELSE result]
END
```

It returns the result when the first **compare_value** comparison becomes true. Otherwise, it will return the else clause.

Example

	studentid	firstname	lastname	class	age
▶	1	Ricky	Ponting	CS	20
	2	Mark	Boucher	EE	22
	3	Michael	Clark	CS	18
	4	Peter	Fleming	CS	22
	5	Virat	Kohli	EC	23
*	NULL	NULL	NULL	NULL	NULL

```
SELECT studentid, firstname,
 CASE class
    WHEN 'CS' THEN 'Computer Science'
    WHEN 'EC' THEN 'Electronics and Communication'
    ELSE 'Electrical Engineering'
END AS department from students;
```

studentid	firstname	department
1	Ricky	Computer Science
2	Mark	Electrical Engineering
3	Michael	Computer Science
4	Peter	Computer Science
5	Virat	Electronics and Communication

String Function

- String values can be explained as “bits of text”. The string functions allow us to manipulate these values before they are displayed.

- Adding text to an existing value
- Changing part of a string
- Extracting text from a string
- Finding a piece of text in string

CONCAT()

- Adding text to an existing value

There are two simple ways to add more text to an existing value - either at the start or end of the text. Placing the text at either end is best achieved with the CONCAT() function.

Syntax:

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

Thus we can take an existing value (say string2) and place a new value (string1) at the beginning to get string1string2. To see this in action let's retrieve the title of The Beatles 'The White Album'

```
mysql> SELECT CONCAT(cds.title," By The Beatles")
    -> FROM cds WHERE cdID='20';
+-------------------------------------+
| CONCAT(cds.title," By The Beatles") |
+-------------------------------------+
| The White Album By The Beatles      |
+-------------------------------------+
1 row in set (0.00 sec)
```

REPLACE

- Changing Part of a String

As well as add text we can replace it or overwrite it completely. To replace an instance of text within a string we can use the REPLACE() function.

Syntax:

```
REPLACE(whole_string,to_be_replaced,replacement)
```

Therefore if we wanted to replace the word 'White' with the word 'Black' in the cds.title:

```
mysql> SELECT REPLACE(cds.title,'White','Black')
    -> FROM cds WHERE cdID='20';
+-----------------------------------+
| REPLACE(cds.title,'White','Black') |
+-----------------------------------+
| The Black Album
+-----------------------------------+
1 row in set (0.02 sec)
```

Insert()

Another Function we can use to add text is the INSERT() function that overwrites any text in the string from a start point for a certain length.

Syntax:

```
INSERT(string,start_position,length,newstring)
```

In this case the crucial bits of information are the position to start (how many characters from the begriming) and the length. So again to replace 'White' (which starts at character 5 in the string) with 'Black' in the title we need to start at position 5 for a length of 5.

```
mysql> SELECT INSERT(cds.title,5,5,'Black')
    -> FROM cds WHERE cdID='20';
+-----------------------------------+
| INSERT(cds.title,5,5,'Black')     |
+-----------------------------------+
| The Black Album                   |
+-----------------------------------+
1 row in set (0.01 sec)
```

LEFT(), RIGHT(), and MID()

- Extracting text from a string

As well as adding text to a string we can also use functions to extract specific data from a string. To begin with lets look at three LEFT(), RIGHT() and MID().

- ***Syntax:***
  - LEFT(string,length)
  - RIGHT(string,length)
  - MID(string,start_position,length)

The first two, LEFT() and RIGHT(), are fairly straight forward. You specify the string and the length of the string to keep, relative to either the left or right depending on which function you are using. So to keep the words 'The' (which occupies 3 characters on the left) and 'Album' (5 characters on the right) we would specify:

Example

```
mysql> SELECT LEFT(cds.title,3), RIGHT(cds.title,5)
    -> FROM cds WHERE cdID='20';
+-------------------+---------------------+
| LEFT(cds.title,3) | RIGHT(cds.title,5)  |
+-------------------+---------------------+
| The               | Album               |
+-------------------+---------------------+
1 row in set (0.00 sec)
```

The MID() function is only slightly complex. You still specify the length, but also a starting position. So to keep the work 'White', you would start at position 5 and have a length of 5.

```
mysql> SELECT MID(cds.title,5,5)
    -> FROM cds WHERE cdID='20';
+--------------------+
| MID(cds.title,5,5) |
+--------------------+
| White              |
+--------------------+
```

SUBSTRING

There is also another extraction function that is worth mentioning; SUBSTRING().

Syntax:
SUBSTRING(string,position)

This returns all of the string after the position. Thus to return 'White Album' you would start at '5'.

```
mysql> SELECT SUBSTRING(cds.title,5)
    -> FROM cds WHERE cdID='20';
+-------------------------+
| SUBSTRING(cds.title,5)  |
+-------------------------+
| White Album             |
+-------------------------+
1 row in set (0.00 sec)
```

LOCATE()

- Finding a piece of text in a string

In some of the string functions we have seen so far it has been necessary to provide a starting position as part of the function This position can be found using the LOCATE() function specifying the text to find (substring) as well as the string to search in.

Syntax:

```
LOCATE(substring,string)
```

So to find the location of 'White':

```
mysql> SELECT LOCATE('White',cds.title)
    -> FROM cds WHERE cdID='20';
+-----------------------------+
| LOCATE('White',cds.title)
+-----------------------------+
|                           5
+-----------------------------+
1 row in set (0.06 sec)
```

LENGTH()

It is also possible to automatically calculate the length of a piece of text using LENGTH().

Syntax:

```
LENGTH(string)
```

So with the word 'White'.

```
mysql> SELECT LENGTH('White');
+------------------+
| LENGTH('White') |
+------------------+
|                5 |
+------------------+
1 row in set (0.03 sec)
```

LCASE() and UCASE()

- Transforming strings

The final group of string functions this workshop will look at are those that transform the string in some way. The first two change the case of the string to either uppercase - UCASE() - or to lowercase - LCASE().

Syntax:

```
LCASE(string)
UCASE(string)
```

As you can imagine the usage of these are fairly straightforward.

```
mysql> SELECT LCASE(cds.title), UCASE(cds.title)
    -> FROM cds WHERE cdID='20';
+------------------ +------------------+
| LCASE(cds.title)   | UCASE(cds.title)   |
+------------------  +------------------+
| the white album    | THE WHITE ALBUM    |
+------------------ +------------------+
1 row in set (0.01 sec)
```

REVERSE

Syntax:

```
REVERSE(string)
```

This rather obviously reverses the order of the letters. For example the alphabet.

```
mysql> SELECT
  REVERSE('abcdefghijklmnopqrstuvwxyz');
+------------------------------------+
| REVERSE('abcdefghijklmnopqrstuvwxyz') |
+------------------------------------+
| zyxwvutsrqponmlkjihgfedcba         |
+------------------------------------+
1 row in set (0.00 sec)
```

Numeric Functions

- Before talking about the specific numeric functions, it is probably worth mentioning that MySQL can perform simple math functions using mathematical operators.

Operator	Function
+	Add
-	Subtract
*	Multiply
/	Divide

Examples:

```
mysql> SELECT 6+3;
+-----+
| 6+3 |
+-----+
|   9 |
+-----+
1 row in set (0.00 sec)
```

FLOOR()

This reduces any number containing decimals to the lowest whole number.

Syntax:

```
SELECT FLOOR(number)
```

Example:

```
mysql> SELECT FLOOR(4.84);
+--------------+
| FLOOR(4.84)  |
+--------------+
|            4 |
+--------------+
1 row in set (0.00 sec)
```

CEILING()

Raises a number containing decimals to the highest whole number.

Syntax:

```
SELECT CEILING(number)
```

Example:

```
mysql> SELECT CEILING(4.84);
+-----------------+
| CEILING(4.84)   |
+-----------------+
|               5 |
+-----------------+
1 row in set (0.01 sec)
```

ROUND()

This function, as you may have guessed, rounds the figures up or down to the nearest whole number (or to a specified number of decimal places).

Syntax:

```
ROUND(number,[Decimal Places])
```

'Decimal Places' is optional and omitting it will mean that the figure is rounded to a whole number.

```
mysql> SELECT ROUND(14.537,2);
+------------------+
| ROUND(14.537,2)  |
+------------------+
|            14.54 |
+------------------+
1 row in set (0.00 sec)
```

TRUNCATE

This function, rather than rounding, simply shortens the number to a required decimal place.

Syntax:

```
TRUNCATE(number,places)
```

Example:

```
mysql> SELECT TRUNCATE(14.537,2);
+----------------------+
| TRUNCATE(14.537,2)   |
+----------------------+
|                14.53 |
+----------------------+
1 row in set (0.00 sec)
```

Date and Time Functions NOW(), CURTIME() and CURDATE()

```
There are three functions that you can use to
  get the current date and time. NOW() - which
  gets both date and time, CURDATE() which works
  with only the date and CURTIME() for the time.
mysql> SELECT NOW(), CURTIME(), CURDATE();
+---------------------  +-----------
  +-------------+
| NOW()                | CURTIME()      |
  CURDATE()   |
+---------------------  +-----------
  +-------------+
| 2003-06-02 19:44:51   | 19:44:51       | 2003-
  06-02 |
+---------------------  +-----------
  +-------------+
1 row in set (0.01 sec)
```

DATE_ADD() and DATE_SUB()

- There are two functions that allow you to add and subtract time to a date. These are DATE_ADD() and DATE_SUB().
- Syntax:
  - DATE_ADD(date,INTERVAL expr type)
  - DATE_SUB(date,INTERVAL expr type)
- The date - is a standard DATE or DATETIME value, next come the command INTERVAL followed by the time period (expr) and finally what type period it is (Month, Day, Year etc). Therefore to work out the date 60 days in the future:

Example

```
The date - is a standard DATE or DATETIME value, next come the
   command INTERVAL followed by the time period (expr) and finally
   what type period it is (Month, Day, Year etc). Therefore to
   work out the date 60 days in the future:
mysql> SELECT DATE_ADD(CURDATE(), INTERVAL 60 DAY);
+----------------------------------------+
| DATE_ADD(CURDATE(), INTERVAL 60 DAY)   |
+----------------------------------------+
| 2003-08-01                             |
+----------------------------------------+
1 row in set (0.00 sec)

Or 6 months in the past:
mysql> SELECT DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
+----------------------------------------+
| DATE_SUB(CURDATE(), INTERVAL 6 MONTH)  |
+----------------------------------------+
| 2002-12-02                             |
+----------------------------------------+
1 row in set (0.00 sec)
```

User Defined Function

- MYSQL also supports user defined functions that extend MYSQL.
- User define functions are functions that you can create using a programming language such as C, C++ etc., and then add them to MYSQL server.
- Once added, they can be used just like any other function.

Thank you