The ranking functions in MySql are used to rank each row of a partition. The ranking functions are also part of MySQL windows functions list.

* These functions are always used with **OVER()** clause.
* The ranking functions always assign rank on basis of **ORDER BY** clause.
* The rank is assigned to rows in a sequential manner.
* The assignment of rank to rows always start with 1 for every new partition.

There are 3 types of ranking functions supported in MySQL-

1. **dense\_rank():**  
   This function will assign rank to each row within a partition **without gaps**. Basically, the ranks are assigned in a consecutive manner i.e if there is a tie between values then they will be assigned the same rank, and next rank value will be one greater then the previous rank assigned.
2. **rank():**  
   This function will assign rank to each row within a partition **with gaps**. Here, ranks are assigned in a non-consecutive manner i.e if there is a tie between values then they will be assigned same rank, and next rank value will be previous rank + no of peers(duplicates).
3. **percent\_rank():**  
   It returns the percentile rank of a row within a partition that ranges from 0 to 1. It tells the percentage of partition values less than the value in the current row, excluding the highest value.

Create result table (name, subjects ,mark)

SELECT subjects, s\_name, mark, dense\_rank()

OVER ( partition by subjects order by mark desc )

AS 'dense\_rank' FROM result;

Here, table is partitioned on the basis of “subjects”.

**order by** clause is used to arrange rows of each partition in descending order by “mark”.

**dense\_rank()** is used to rank students in each subject.

**rank()** function-

SELECT subjects, s\_name, mark, rank()

OVER ( partition by subjects order by mark desc )

AS 'rank' FROM result;

**percent\_rank()** function

SELECT subjects, s\_name, mark, percent\_rank()

OVER ( partition by subjects order by mark )

AS 'percent\_rank' FROM result;

**Here,**the percent\_rank() function calculate percentile rank in ascending order by “mark” column.

percent\_rank is calculated using following formula-

(rank - 1) / (rows - 1)

**rank** is the rank of each row of the partition resulted using rank() function.

**rows** represent the no of rows in that partition.

To, clear this formula consider following query-

SELECT subjects, s\_name, mark, rank()

OVER ( partition by subjects order by mark )-1

AS 'rank-1', count(\*) over (partition by subjects)-1

AS 'total\_rows-1', percent\_rank()

OVER ( partition by subjects order by mark ) AS 'percenr\_rank'

FROM result;

================ Nullable

In MySQL, a NULL value means unknown. A NULL value is different from zero (0) or an empty string ''.

A NULL value is not equal to anything, even itself. If you compare a NULL value with another NULL value or any other value, the result is NULL because the value of each NULL value is unknown.

Generally, you use the NULL value to indicate that the data is missing, unknown, or not applicable. For example, the phone number of a potential customer may be NULL and can be added later.

When you [create a table](https://www.mysqltutorial.org/mysql-create-table/), you can specify whether a column accepts NULL values or not by using the [NOT NULL](https://www.mysqltutorial.org/mysql-not-null-constraint/) constraint.

Create leads table

**CREATE** **TABLE** leads (

**id** INT AUTO\_INCREMENT PRIMARY **KEY**,

first\_name VARCHAR(50) **NOT** NULL,

last\_name VARCHAR(50) **NOT** NULL,

**source** VARCHAR(255) **NOT** NULL,

email VARCHAR(100),

phone VARCHAR(25)

);

**INSERT** **INTO** leads(first\_name,last\_name,**source**,email,phone)

**VALUE**('John','Doe','Web Search','john.doe@acme.com',NULL);

**INSERT** **INTO** leads(first\_name,last\_name,**source**,phone)

**VALUES**

('Lily','Bush','Cold Calling','(408)-555-1234'),

('David','William','Web Search','(408)-888-6789');

The IFNULL function accepts two parameters. The IFNULL function returns the first argument if it is not NULL, otherwise, it returns the second argument.

**SELECT** **id**, first\_name, last\_name, **IFNULL**(phone, 'N/A') phone

**FROM**  leads;

The COALESCE function accepts a list of arguments and returns the first non-NULL argument. For example, you can use the COALESCE function to display the contact information of a lead based on the priority of the information in the following order: phone, email, and N/A.

**SELECT**

**id**,

first\_name,

last\_name,

**COALESCE**(phone, email, 'N/A') contact

**FROM**

leads;

The NULLIF function accepts two arguments. If the two arguments are equal, the NULLIF function returns NULL. Otherwise, it returns the first argument.

The NULLIF function is useful when you have both NULL and empty string values in a column. For example, by mistake, you insert a following row into the leads table:

**INSERT** **INTO** leads(first\_name,last\_name,**source**,email,phone)

**VALUE**('Thierry','Henry','Web Search','thierry.henry@example.com','');

Code language: SQL (Structured Query Language) (sql)

The phone is an empty string instead of NULL.

If you want to get the contact information of leads, you end up with an empty phone instead of the email as the following query:

**SELECT**

**id**,

first\_name,

last\_name,

**COALESCE**(phone, email, 'N/A') contact

**FROM**

leads;

To fix this, you use the NULLIF function to compare the phone with the empty string, if they are equal, it returns NULL, otherwise, it returns the phone number.

**SELECT**

**id**,

first\_name,

last\_name,

**COALESCE**(**NULLIF**(phone, ''), email, 'N/A') contact

**FROM**

leads;

**CREATE** **TABLE** leads (

**id** INT AUTO\_INCREMENT PRIMARY **KEY**,

first\_name VARCHAR(50) **NOT** NULL,

last\_name VARCHAR(50) **NOT** NULL,

**source** VARCHAR(255) **NOT** NULL,

email VARCHAR(100),

phone VARCHAR(25)

);

=== ifNull()

**SELECT** **id**, first\_name, last\_name, **IFNULL**(phone, 'N/A') phone

**FROM**  leads;

**--- Coalesce()**

**SELECT** **id**, first\_name, last\_name,

**COALESCE**(phone, email, 'N/A') contact

**FROM**  leads;

**SELECT** **id**, first\_name, last\_name, **COALESCE**(**NULLIF**(phone, ''), email, 'N/A') contact **FROM**  leads;

Date functions in mYsql

Curdate() -- for current date

Select date\_add(curdate(), interval 20 day);

Select adddate(‘2021-02-01’,15);

Select dayname(curdate());

==== convert

select curdate() 'Today date',

concat(dayname(curdate()),' ',convert(year(curdate()),char)) 'Format';

select datediff(date1,date2);

select date\_sub(‘date’, interval 10 day);