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.

Example:

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

OVER ( partition by subjects order by mark desc )

AS 'dense\_rank' FROM result;

1. **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).

**Example:**

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

OVER ( partition by subjects order by mark desc )

AS 'rank' FROM result;

1. **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.

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

OVER ( partition by subjects order by mark )

AS 'percent\_rank' FROM result;

percent\_rank is calculated using following formula-

(rank - 1) / (rows - 1)

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 'percent\_rank'

FROM result;

1. **Note:** While using ranking function, in MySQL query the use of order by clause is must otherwise all rows are considered as peers i.e(duplicates) and all rows are assigned same rank i.e 1.

Mysql Limit clause:

Limit is used to return the number of records to return.

Select \* from PMS\_EMPLOYEE\_DETAILS limit 10;

Select \* from PMS\_EMPLOYEE\_DETAILS where limit 10;

**Mysql second Highest salary Finding:**

* 1. **Using IN clause**

SELECT MAX(salary) FROM employees

WHERE salary NOT IN ( SELECT Max(salary) FROM employees);

* 1. Using sub query and < operator instead of IN clause
  2. SELECT MAX(salary) From employees

WHERE salary < ( SELECT Max(salary) FROM employees);

3.Using Limit clause to find highest Salary:

SELECT salary

FROM (SELECT salary FROM employees ORDER BY salary DESC LIMIT 2) AS Emp ORDER BY salary LIMIT 1

(or)

SELECT \* FROM employee ORDER by salary DESC LIMIT 1,2

MySQL RANK() function :

SELECT

first\_name,

last\_name,

salary,

RANK() OVER (ORDER BY salary) salary\_rank

FROM

employees;

The ORDER BY clause sorted the rows in the result by salary. The RANK() function then is applied to each row in the result considering the order of employees by salary in descending order.

**Subqueries in MySQL:**

In MySQL, a subquery is a query within a query. You can create subqueries within your SQL statements. These subqueries can reside in the

* + - WHERE clause,
    - FROM clause,
    - SELECT clause.

**The following are the rules to use subqueries:**

* Subqueries should always use in **parentheses.**
* If the main query does not have multiple columns for subquery, then a subquery can have only one column in the SELECT command.
* We can use various comparison operators with the subquery, such as >, <, =, IN, ANY, SOME, and ALL. A multiple-row operator is very useful when the subquery returns more than one row.
* We cannot use the **ORDER BY** clause in a subquery, although it can be used inside the main query.
* If we use a subquery in a **set function**, it cannot be immediately enclosed in a set function.

**The following are the advantages of using subqueries:**

* The subqueries make the queries in a structured form that allows us to isolate each part of a statement.
* The subqueries provide alternative ways to query the data from the table; otherwise, we need to use complex joins and unions.
* The subqueries are more readable than complex join or union statements.

Subquery Syntax:

SELECT column\_list (s) FROM  table\_name

WHERE  column\_name OPERATOR

(SELECT column\_list (s)  FROM table\_name [WHERE])

**1. Using Where Clause:**

1.**Employee detail whose id matches in a subquery**:

SELECT emp\_name, city, income FROM employees

   WHERE emp\_id IN (SELECT emp\_id FROM employees);

2.**Employee detail whose salary is more than 350000**

SELECT \* FROM employees

   WHERE emp\_id IN (SELECT emp\_id FROM employees

WHERE income > 350000);

**3.maximum income**

SELECT emp\_name, city, income FROM employees

   WHERE income = (SELECT MAX(income) FROM employees);

**4. student detail who does not belong to particular City** from both tables as follows:

1.SELECT Name, City FROM student

WHERE City NOT IN ( SELECT City FROM student2 WHERE City='Bangalore');  

### 2. MySQL Subquery in the FROM Clause

SELECT Max(items), MIN(items), FLOOR(AVG(items))

FROM

   (SELECT order\_id, COUNT(order\_id) AS items FROM orders

GROUP BY order\_date) AS Student\_order\_detail;

### MySQL Correlated Subqueries:

A correlated subquery in MySQL is a subquery that depends on the outer query.

It uses the data from the outer query or contains a reference to a

query that also appears in the outer query.

MySQL evaluates it once from each row in the outer query.

Innere Query depends on outer query. An outer query depends on inner query.

**Syntax:**

Select colname (select colname from tablename where colname = a.colname ) from tablename a;

**Example:**

**1.Using Where:**

find Highest salary using single row subquery.

select empId,salary from PMS\_EMPLOYEE\_DETAILS where salary=(select max(salary) from PMS\_EMPLOYEE\_DETAILS);

**Multiple Row Subquery:**

select ... from PMS\_EMPLOYEE\_DETAILS where deptId in (select deptid from PMS\_EMPLOYEE\_DETAILS)

SELECT emp\_name, city, income FROM employees

WHERE emp\_id IN (SELECT emp\_id FROM employees);

**2. Using From Operator:**

(i) SELECT Max(items), MIN(items), FLOOR(AVG(items))

FROM

(SELECT order\_id, COUNT(order\_id) AS items FROM orders

GROUP BY order\_date) AS Student\_order\_detail;

SELECT emp\_name, city, income

FROM employees emp WHERE income > (

SELECT AVG(income) FROM employees WHERE city = emp.city);

**(ii)Subquery with Insert:**

insert into emp select \* from pms\_employee\_details where department\_id in(select department\_id from pms\_department\_details)

select \* from emp

**(iii)Subquery with Update**

update emp set salary=salary\*0.25 where commission in(select commission from pms\_employee\_details where commission >1000)

**(iv)Subquery with Delete**

delete from emp where salary in(select salary from PMS\_EMPLOYEE\_DETAILS where salary>20000)

**with cte:**

with ctename

as

(select colname from tablename)

select \* from ctename

**cte characteristics:**

* CTE is defined using the WITH clause.
* We can define more than one CTEs in the same query.
* One CTE can be used to reference another CTE in the same WITH scope.
* CTE can be used in that query only in which it is defined.

--Using CTE

--to display the employee details along with department which has maximum no of employees

**WITH** emps\_city **AS**(

**SELECT** \* **FROM** emps **WHERE** city="Bangalore ")

**SELECT** \* **FROM** emps\_city **WHERE**

age > 25 **ORDER** **BY** age;

### Using SQL RANK() function over partition example:

The following statement finds the employees who have the second highest salary in their departments:

WITH payroll AS (

SELECT

first\_name,

last\_name,

department\_id,

salary,

RANK() OVER (

PARTITION BY department\_id

ORDER BY salary) salary\_rank

FROM

employees

)

SELECT

first\_name,

last\_name,

department\_name,

salary

FROM

payroll p

INNER JOIN departments d

ON d.department\_id = p.department\_id

WHERE

salary\_rank = 2;

**Exists and Not Exists:**

SELECT name, occupation, age FROM customer C

WHERE EXISTS (SELECT \* FROM Orders O

WHERE C.cust\_id = O.cust\_id);

This statement uses **NOT** EXISTS operator that returns the customer details who have not placed an order.

SELECT name, occupation, age FROM customer C

WHERE NOT EXISTS (SELECT \* FROM Orders O

WHERE C.cust\_id = O.cust\_id);

**MySQL Subqueries with ALL, ANY**

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

**The ANY Operator**

The ANY operator returns a boolean value as a result. It returns TRUE if ANY of the subquery values meet the condition.

ANY means that the condition will be true if the operation is true for any of the values in the range.

**ANY Syntax:**

SELECT column\_name(s)  
FROM table\_name  
WHERE column\_name

operator ANY

(SELECT column\_name  
  FROM table\_name  WHERE condition);

**Note:** The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

**The All Operator:**

The ALL operator:returns a boolean value as a result

returns TRUE if ALL of the subquery values meet the condition

is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

**ALL Syntax With SELECT**

SELECT ALL *column*\_name(s)  
FROM table\_*name*  
WHERE condition;

SELECT *column\_name(s)*  
FROM *table\_name*  
WHERE *column\_name*

*operator* ALL

(SELECT column\_name  FROM table\_name  WHERE condition);

**Example:**

SELECT cust\_id, name FROM customer WHERE

cust\_id = ANY (SELECT cust\_id FROM Orders);

SELECT cust\_id, name FROM customer WHERE

cust\_id =ALL (SELECT cust\_id FROM Orders);