Table of Contents

1. Ranking

A ranking shows a position in a sorted list. MySQL does not have a ranking function and we will look at a few ways to get around this problem.

Demo 01: We will start with the adv_emp table. These are the employees sorted by salary with the low salaries first. These are not yet ranked, but they are ordered

```
+--------+---------+--------+
| emp_id | dept_id | salary |
+--------+---------+--------+
|    102 |      20 |  20000 |
|    120 |      20 |  22000 |
|    115 |      30 |  24000 |
|    104 |      20 |  25000 |
|    105 |      20 |  25000 |
|    118 |      45 |  25000 |
|    119 |      45 |  25000 |
|    121 |      45 |  28000 |
|    117 |      30 |  28000 |
|    116 |      30 |  28000 |
|    108 |      45 |  28000 |
|    106 |      20 |  28000 |
|    103 |      30 |  28000 |
|    112 |      20 |  30900 |
|    114 |      45 |  32000 |
|    109 |      45 |  32000 |
|    113 |      45 |  45000 |
|    111 |      45 |  45000 |
|    110 |      30 |  45000 |
|    107 |      45 |  45000 |
|    101 |      10 |  45000 |
+--------+---------+--------+
21 rows in set (0.00 sec)
```

Suppose our company decides that they need to increase salaries for the lowest paid salaries and they can afford to increase the salaries of only 5 employees.

Demo 02: We could start with adding a Limit 5 clause.

```
+--------+---------+--------+
| emp_id | dept_id | salary |
+--------+---------+--------+
|    102 |      20 |  20000 |
|    120 |      20 |  22000 |
|    115 |      30 |  24000 |
|    104 |      20 |  25000 |
|    105 |      20 |  25000 |
+--------+---------+--------+
5 rows in set (0.00 sec)
```

But is that going to work? One of the problems with ranking is ties. We have selected employees 104 and 105 with salaries 25000 for increases but employees 118 and 119 also have salary 25000 and are rightfully rather annoyed by your query.

The company could set up business rules to handle these issues of ties (the nicest would be to give raises to everyone tied at that position). They could pick employees at random from the tied people; they could have a tie breaker such as using the hire date- that would not guarantee no ties. What we will look at is just getting the tieds rows.

For now I am going to just type in ranking numbers for the first few rows.

```
+--------+---------+--------+
| emp_id | dept_id | salary |
+--------+---------+--------+
|    102 |      20 |  20000 | # 1
|    120 |      20 |  22000 | # 2
|    115 |      30 |  24000 | # 3
```

That looks OK so far, and it would make sense that the tied rows get the same rank.

```
|    104 |      20 |  25000 | # 4
|    105 |      20 |  25000 | # 4
|    118 |      45 |  25000 | # 4
|    119 |      45 |  25000 | # 4
```

Now we will go to the next rank number

```
|    121 |      45 |  28000 | # 5
|    117 |      30 |  28000 | # 5
```

Or maybe you think that the next rank number should be #8 since this is the 8th row. In that version, no one gets rank 5, 6, 7 (since those were all tied at rank 4).

```
|    121 |      45 |  28000 | # 8
|    117 |      30 |  28000 | # 8
```

2. Various ranking schemes

2.1. Dense rank

In this result set, people with the same salary get the same rank number. The next salary gets the next rank number. This is called dense ranking since none of the rank numbers are skipped.

This uses a correlated subquery and uses one copy of the table to get the first few columns and the second to get the rank column. The logic in the subquery essentially says to count how many people have a salary less than or equal to this employee's salary and that is the value for salary_rank

Demo 03:

```
Select
  emp_1.emp_id
, dept_id
, emp_1.salary
, (
   Select count(distinct salary)
   From a_testbed.adv_emp as emp_2
   Where emp_2.salary <= emp_1.salary
   )as salary_rank
From a_testbed.adv_emp as emp_1
order by salary_rank;
+--------+---------+--------+-------------+
| emp_id | dept_id | salary | salary_rank |
+--------+---------+--------+-------------+
|    102 |      20 |  20000 |           1 |
|    120 |      20 |  22000 |           2 |
|    115 |      30 |  24000 |           3 |
|    104 |      20 |  25000 |           4 |
|    105 |      20 |  25000 |           4 |
|    118 |      45 |  25000 |           4 |
|    119 |      45 |  25000 |           4 |
|    121 |      45 |  28000 |           5 |
|    117 |      30 |  28000 |           5 |
|    116 |      30 |  28000 |           5 |
|    108 |      45 |  28000 |           5 |
|    106 |      20 |  28000 |           5 |
|    103 |      30 |  28000 |           5 |
|    112 |      20 |  30900 |           6 |
|    114 |      45 |  32000 |           7 |
|    109 |      45 |  32000 |           7 |
|    113 |      45 |  45000 |           8 |
|    111 |      45 |  45000 |           8 |
|    110 |      30 |  45000 |           8 |
|    107 |      45 |  45000 |           8 |
|    101 |      10 |  45000 |           8 |
+--------+---------+--------+-------------+
21 rows in set (0.00 sec)
```

2.2. Non-dense rank

There is another way to count the ranks. People with the same salary get the same rank number but rank numbers are skipped. At first the rank numbers will seem off, but what is happening is that the tied rows get the largest sequential rank for that value. In the demo output, there are 4 people tied at salary 25000 and instead of their getting ranks 4,5,6,7- they all get rank 7. This is a non-dense rank because rank numbers are skipped.

Demo 04: What is the difference in the syntax between these two queries?

```
Select
  emp_1.emp_id
, dept_id
, emp_1.salary
, (
   Select count(salary)
   From a_testbed.adv_emp as emp_2
   Where emp_2.salary <= emp_1.salary
   ) as salary_rank
From a_testbed.adv_emp as emp_1
Order by salary_rank;
+--------+---------+--------+-------------+
| emp_id | dept_id | salary | salary_rank |
+--------+---------+--------+-------------+
|    102 |      20 |  20000 |           1 |
|    120 |      20 |  22000 |           2 |
|    115 |      30 |  24000 |           3 |
|    104 |      20 |  25000 |           7 |
|    105 |      20 |  25000 |           7 |
|    118 |      45 |  25000 |           7 |
|    119 |      45 |  25000 |           7 |
|    121 |      45 |  28000 |          13 |
|    117 |      30 |  28000 |          13 |
|    116 |      30 |  28000 |          13 |
|    108 |      45 |  28000 |          13 |
|    106 |      20 |  28000 |          13 |
|    103 |      30 |  28000 |          13 |
|    112 |      20 |  30900 |          14 |
|    114 |      45 |  32000 |          16 |
|    109 |      45 |  32000 |          16 |
|    113 |      45 |  45000 |          21 |
|    111 |      45 |  45000 |          21 |
|    110 |      30 |  45000 |          21 |
|    107 |      45 |  45000 |          21 |
|    101 |      10 |  45000 |          21 |
+--------+---------+--------+-------------+
21 rows in set (0.00 sec)
```

Demo 05: If the previous output annoys you, some small modifications can report the tied rows at the smaller rank number and still skip ranks numbers.

```
Select emp_1.emp_id, dept_id, emp_1.salary
, (
   Select count(salary)
   From   a_testbed.adv_emp as emp_2
   Where  emp_2.salary < emp_1.salary
  ) +1 as salary_rank
From  a_testbed.adv_emp as emp_1
Order by salary_rank;
+--------+---------+--------+-------------+
| emp_id | dept_id | salary | salary_rank |
+--------+---------+--------+-------------+
|    102 |      20 |  20000 |           1 |
|    120 |      20 |  22000 |           2 |
|    115 |      30 |  24000 |           3 |
|    104 |      20 |  25000 |           4 |
|    105 |      20 |  25000 |           4 |
|    118 |      45 |  25000 |           4 |
|    119 |      45 |  25000 |           4 |
|    121 |      45 |  28000 |           8 |
|    117 |      30 |  28000 |           8 |
|    116 |      30 |  28000 |           8 |
|    108 |      45 |  28000 |           8 |
|    106 |      20 |  28000 |           8 |
|    103 |      30 |  28000 |           8 |
|    112 |      20 |  30900 |          14 |
|    114 |      45 |  32000 |          15 |
|    109 |      45 |  32000 |          15 |
|    113 |      45 |  45000 |          17 |
|    111 |      45 |  45000 |          17 |
|    110 |      30 |  45000 |          17 |
|    107 |      45 |  45000 |          17 |
|    101 |      10 |  45000 |          17 |
+--------+---------+--------+-------------+
21 rows in set (0.00 sec)
```

3. MySQL Approach

This is a very MySQL approach to this which uses some MySQL functions we discussed earlier. You can read more about this and some of the issues with a user variable approach that I am not including from the following site

http://rpbouman.blogspot.com/2009/09/mysql-another-ranking-trick.html

These give us rank and dense rank. What essentially happens here is that this uses group_concat to get a csv list of all of the salaries and Find_in_set to do the rank.

First we can look at what group_concat gives us.

Demo 06: This uses group_concat which concatenates all the salaries separated by commas.

```
Select
Group_concat(salary order by salary)  as salarylist
From a_testbed.adv_emp \G
*************************** 1. row ***************************
salarylist:
20000,22000,24000,25000,25000,25000,25000,28000,28000,28000,28000,28000,28000,3
0900,32000,32000,45000,45000,45000,45000,45000
1 row in set (0.00 sec)
```

Demo 07: Add Distinct to get only one copy of each salary value

```
Select
Group_concat(distinct salary order by salary)  as salarylist
From  a_testbed.adv_emp \G
*************************** 1. row ***************************
salarylist: 20000,22000,24000,25000,28000,30900,32000,45000
1 row in set (0.00 sec)
```

Now look at that string of salary values. What is the position of salary 25000 in that list?

```
(1)20000,(2)22000,(3)24000,(4)25000,(5)28000,(6)30900,(7)32000,(8)45000
```

It is in postion 4 and that is its rank. The Find_in_set function can do that search for us.

```
Select find_in_set( 25000, '20000,22000,24000,25000,28000,30900,32000,45000');
```

(Functions are your friends- particulalry when you nest them.)

Demo 08: Now use Find_in_set to pick out the position of a salary in that list, giving the rank

```
Select emp_id, salary
, find_in_set(
    salary
  , (
     Select group_concat( distinct salary order by salary )
     From  a_testbed.adv_emp
    )
    ) as salary_rank
From a_testbed.adv_emp
Order by salary_rank;
+--------+--------+-------------+
| emp_id | salary | salary_rank |
+--------+--------+-------------+
|    102 |  20000 |           1 |
|    120 |  22000 |           2 |
|    115 |  24000 |           3 |
|    104 |  25000 |           4 |
|    105 |  25000 |           4 |
|    118 |  25000 |           4 |
|    119 |  25000 |           4 |
|    121 |  28000 |           5 |
|    117 |  28000 |           5 |
|    116 |  28000 |           5 |
|    108 |  28000 |           5 |
|    106 |  28000 |           5 |
|    103 |  28000 |           5 |
|    112 |  30900 |           6 |
|    114 |  32000 |           7 |
|    109 |  32000 |           7 |
|    113 |  45000 |           8 |
|    111 |  45000 |           8 |
|    110 |  45000 |           8 |
|    107 |  45000 |           8 |
|    101 |  45000 |           8 |
+--------+--------+-------------+
21 rows in set (0.00 sec)
```

Demo 09: What if I skip Distinct? What happens to the Rank column?

```
Select emp_id, salary
, find_in_set(
    salary
  , (
     Select group_concat(salary order by salary )
     From  a_testbed.adv_emp
    )
    ) as salary_rank
From a_testbed.adv_emp
Order by salary_rank;
+--------+--------+-------------+
| emp_id | salary | salary_rank |
+--------+--------+-------------+
|    102 |  20000 |           1 |
|    120 |  22000 |           2 |
|    115 |  24000 |           3 |
|    104 |  25000 |           4 |
|    105 |  25000 |           4 |
|    118 |  25000 |           4 |
|    119 |  25000 |           4 |
|    121 |  28000 |           8 |
|    117 |  28000 |           8 |
|    116 |  28000 |           8 |
|    108 |  28000 |           8 |
|    106 |  28000 |           8 |
|    103 |  28000 |           8 |
|    112 |  30900 |          14 |
|    114 |  32000 |          15 |
|    109 |  32000 |          15 |
|    113 |  45000 |          17 |
|    111 |  45000 |          17 |
|    110 |  45000 |          17 |
|    107 |  45000 |          17 |
|    101 |  45000 |          17 |
+--------+--------+-------------+
21 rows in set (0.00 sec)
```

3.1. Group_concat Limit

Our tables are small so the result returned by group_concat is short. But there is a limit to the string returned by group_concat- 1024

(Manual): The result is truncated to the maximum length that is given by the group_concat_max_len system variable, which has a default value of 1024. The value can be set higher, although the effective maximum length of the return value is constrained by the value of max_allowed_packet. The syntax to change the value of group_concat_max_len at runtime is as follows, where val is an unsigned integer:

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

3.2. Getting the top 5

Looking back at the first output result set, which rows should we return for the top 5 (or in our case bottom 5 salaries). Which of the following two return sets do you want?

This gets the top5 different salary values and returns the rows with those salary values. So we get all ties at each of the top 5 salary values.

Demo 10: Try figuring out the code- it is in the demo

```
+--------+--------+----------+
| emp_id | salary | sal_rank |
+--------+--------+----------+
|    102 |  20000 |        1 |
|    120 |  22000 |        2 |
|    115 |  24000 |        3 |
|    118 |  25000 |        4 |
|    119 |  25000 |        4 |
|    104 |  25000 |        4 |
|    105 |  25000 |        4 |
|    117 |  28000 |        5 |
|    103 |  28000 |        5 |
|    121 |  28000 |        5 |
|    106 |  28000 |        5 |
|    108 |  28000 |        5 |
|    116 |  28000 |        5 |
+--------+--------+----------+
13 rows in set (0.00 sec)
```

This gets the top 5 salaries with all of the ties at the last position

Demo 11: Try figuring out the code- it is in the demo

```
+--------+--------+----------+
| emp_id | salary | sal_rank |
+--------+--------+----------+
|    102 |  20000 |        1 |
|    120 |  22000 |        2 |
|    115 |  24000 |        3 |
|    118 |  25000 |        4 |
|    119 |  25000 |        4 |
|    104 |  25000 |        4 |
|    105 |  25000 |        4 |
+--------+--------+----------+
7 rows in set (0.00 sec)
```

Demo 12: You can also do this logic with a cross join instead of another level of subquery

```
Select *
From (
   Select emp_id, salary
   , find_in_set( salary,  all_salaries) as sal_rank
   From a_testbed.adv_emp
   Cross join (
      Select group_concat( salary order by salary ) as all_salaries
      From a_testbed.adv_emp
      ) SalGrouped
   ) dataSet
Where sal_rank <=5
Order By sal_rank;
```

Demo 13: This uses a group by dept_id to get the top 3 people from each department.

```
select dept_id, salary, salary_rank
from (
     select e.dept_id
     , e.salary
     , find_in_set(e.salary, dptgrp.sal_list)  as salary_rank
     from a_testbed.adv_emp as e,
     (
      select dept_id
      , group_concat(salary order by salary ) sal_list
      from a_testbed.adv_emp
      group by dept_id
      ) as dptgrp
    where e.dept_id = dptgrp.dept_id
    ) as z
 where salary_rank <= 3
 order by dept_id, salary_rank;
+---------+--------+-------------+
| dept_id | salary | salary_rank |
+---------+--------+-------------+
|      10 |  45000 |           1 |
|      20 |  20000 |           1 |
|      20 |  22000 |           2 |
|      20 |  25000 |           3 |
|      20 |  25000 |           3 |
|      30 |  24000 |           1 |
|      30 |  28000 |           2 |
|      30 |  28000 |           2 |
|      30 |  28000 |           2 |
|      45 |  25000 |           1 |
|      45 |  25000 |           1 |
|      45 |  28000 |           3 |
|      45 |  28000 |           3 |
+---------+--------+-------------+
13 rows in set (0.00 sec)
```

Demo 14: Join version of the previous demo

```
Select dept_id, salary, salary_rank
From (
   Select e.emp_id, e.salary, e.dept_id
   , find_in_set( salary,  all_salaries) as salary_rank
   From a_testbed.adv_emp e
   Join (
      Select dept_id, group_concat( salary order by salary ) as all_salaries
      From a_testbed.adv_emp
      Group by dept_id
      ) dptgrp  on e.dept_id = dptgrp.dept_id
   ) dataSet
Where salary_rank <= 3
Order by dept_id, salary_rank;
```