Table of Contents

Ranking functions are used to rank rows of data according to some criteria. This is one of the simplest analytical techniques to understand. You probably rank many things. We might want to rank employees by salary or we might want to rank employees by salary within each department. Ranking functions have to consider ties. The ranking functions discussed here are:

- Row_Number
- Rank
- Dense_Rank

These analytic functions can be used only in the Select list or the Order By clause. Other parts of the query (join, Where, Group by, Having) are carried out before the analytic functions.

We will review RowNum to avoid confusion. RowNum is **not** a ranking function

I will generally show only a sampling of the rows for each result set. For reference, these are the rows in the table adv_emp.(the sql for these is in demo_01)

```
EMP_ID NAME_LAST                    DEPT_ID YEAR_HIRED     SALARY
------ ------------------------- ---------- ---------- ----------
301    Green                             10       2010      15000
302    Hancock                           20       2010      14000
303    Quebec                            20       2014      27000
304    Mobley                            30       2010      28000
305    Coltrane                          10       2012      27000
306    Cohen                             30       2010      28000
307    Tatum                             30       2012      13500
308    Evans                             30       2013      15000
309    Beiderbecke                       10       2014      30000
310    Wabich                            10       2012      25000
311    Brubeck                           10       2012      28000
312    Ellington                         20       2010      28000
313    Davis                             30       2012      11000
314    Turrentine                        30       2013      30000
315    Battaglia                         20       2013      12000
316    Monk                              30       2013      26000
317    Wasliewski                        30       2014      25000
318    Shorter                           30       2014      11500
319    Redman                            10       2014      30000
320    Jarrett                           10       2012      25000
321    Rollins                           10       2014      30000
322    Wabich                            10       2012      25000
323    Montgomery                        15       2012      25000

23 rows selected.
```

1. RowNum

Suppose we run the following three queries:

Demo 01: Displaying Rownum. The rows display the Rownum values in numeric order

```
select emp_id, salary, dept_id, RowNum
from adv_emp;
```

```
EMP_ID     SALARY    DEPT_ID     ROWNUM
------ ---------- ---------- ----------
301         15000         10          1
302         14000         20          2
303         27000         20          3
304         28000         30          4
305         27000         10          5
306         28000         30          6
307         13500         30          7
308         15000         30          8
309         30000         10          9
310         25000         10         10
311         28000         10         11
312         28000         20         12
313         11000         30         13
314         30000         30         14
315         12000         20         15
316         26000         30         16
317         25000         30         17
318         11500         30         18
319         30000         10         19
320         25000         10         20
321         30000         10         21
322         25000         10         22
323         25000         15         23
```

Demo 02: Now add a sorting clause. We still get the RowNum values but they do not appear in numeric order since the rows are sorted by the salary.

```
select emp_id, salary, dept_id, RowNum
from adv_emp
order by salary ;
```

```
EMP_ID     SALARY    DEPT_ID     ROWNUM
------ ---------- ---------- ----------
313         11000         30         13
318         11500         30         18
315         12000         20         15
307         13500         30          7
302         14000         20          2
301         15000         10          1
308         15000         30          8
322         25000         10         22
323         25000         15         23
320         25000         10         20
317         25000         30         17
310         25000         10         10
316         26000         30         16
305         27000         10          5
303         27000         20          3
```

```
306          28000          30          6
312          28000          20         12
311          28000          10         11
304          28000          30          4
314          30000          30         14
321          30000          10         21
319          30000          10         19
309          30000          10          9
```

The RowNum values are determined before the sort is applied- so we do not see the RowNum values in row number order. RowNum values reflect something about the physical order of the rows in secondary storage and how they are retrieved from storage and we know that we should never write code based on the physical characteristics of the rows in storage. RowNum is a pseudo column that is generated as the rows are retrieved from secondary storage and we have little control over that. RowNum is Oracle specific.

1.1. Using a subquery

As a first attempt to solve this we could try a subquery; the subquery does the sort and then delivers the sorted rows to the parent query which applies the RowNum.

Demo 03: Getting the data in a subquery in the From clause and using RowNum in the parent query

```
select emp_id, dept_id, salary, rownum
from ( select emp_id, dept_id, salary
       from adv_emp
       order by salary);
```

This does give us the rows ranked ok- except that employees can have the same salary but different row num (such as salary 25000). If you were going to give raises to the ten lowest paid employees- this would be a problem. (Think of this as an automatic raise- not a situation where a human looks at the output and notices the tie!)

```
EMP_ID    DEPT_ID     SALARY     ROWNUM
------ ---------- ---------- ----------
313            30      11000          1
318            30      11500          2
315            20      12000          3
307            30      13500          4
302            20      14000          5
301            10      15000          6
308            30      15000          7
322            10      25000          8
323            15      25000          9
320            10      25000         10
317            30      25000         11
310            10      25000         12
316            30      26000         13
305            10      27000         14
303            20      27000         15
306            30      28000         16
312            20      28000         17
311            10      28000         18
304            30      28000         19
314            30      30000         20
321            10      30000         21
319            10      30000         22
309            10      30000         23
```

There have been various ways to work around these kinds of problems and there is a need to have a uniform way of dealing with this task.

2. Row_Number

This example uses the row_number function and a simple windowing clause. That function produces a new number for each row in the result set. Notice that the Over clause is using to supply the sorting rule. The rows in a table do not have a natural ordering- we need to supply one.

You cannot use the row_number function without also supplying an Over clause. Row_Number is an ansii standard technique.

Demo 04: Using the Row_Number() function

```
Select emp_id, dept_id, salary
, row_number () Over (order by salary  ) as col_Order
from adv_emp;
```

```
EMP_I    DEPT_ID     SALARY  COL_ORDER
----- ---------- ---------- ----------
313           30      11000          1
318           30      11500          2
315           20      12000          3
307           30      13500          4
302           20      14000          5
301           10      15000          6
308           30      15000          7
322           10      25000          8
323           15      25000          9
320           10      25000         10
317           30      25000         11
310           10      25000         12
316           30      26000         13
305           10      27000         14
303           20      27000         15
306           30      28000         16
312           20      28000         17
311           10      28000         18
304           30      28000         19
314           30      30000         20
321           10      30000         21
319           10      30000         22
309           10      30000         23

23 rows selected.
```

The query gets data from the adv_emp view and returns each row; there is no Where clause to filter the rows. The row_number function supplies a number for each row returned to the result set.

The row_number function has a different syntax than we are used to for functions; it has a required Over ( ) clause attached to it. This is a "window specification"; it is also referred to as a partition by clause. This clause can contain any of a number of things- here we have an ordering. It is the order by clause in the parentheses following the keyword Over which says to supply the row_numbers in salary order.

The windowing clause lets us calculate moving averages.

We can reset aggregates or ranks when a department changes (a control break report)

We can use multiple functions within a single query.

The Over clause supports three different techniques

- Order the rows by some attribute: `Over (order by salary  )`
- Partition the rows by some attribute: `Over (partition by Dept_id order by salary   )`
- Define a moving window frame: `Over (order by day rows between 2 preceding and 1 following)`

Change the SQL to sort in descending order and the rows are ranked in descending order.

Demo 05: Row_Number with a descending sort

```
Select emp_id, dept_id, salary
, row_number () Over (order by salary desc  nulls last) as col_Order
from adv_emp;
```

Demo 06: You can specify Nulls Last or Null First in the ordering

```
Select emp_id, dept_id, salary
, row_number () Over (order by salary desc  nulls first) as col_Order
from  adv_emp;
```

Now add a regular order by clause as the last clause in the query. The final order by clause controls the order in which the rows are displayed. It does not affect the value returned by the row_number function. If you have a lot of employees and you want to find their row_number rank, sorting by the employee id can be useful.

Demo 07: Row_Number and sorting the result

```
select emp_id, dept_id, salary
, row_number () Over (order by salary desc  ) as col_Order
from adv_emp
order by emp_id;
```

```
EMP_ID    DEPT_ID     SALARY  COL_ORDER
------ ---------- ---------- ----------
301           10      15000         17
302           20      14000         19
303           20      27000         10
304           30      28000          6
305           10      27000          9
306           30      28000          8
307           30      13500         20
308           30      15000         18
309           10      30000          3
310           10      25000         12
. . .  rows omitted.
```

But most of the time you would want to do a final sort by the same attribute as the "over" clause to emphasize the ranking order. Note that we have not solved the problem with the ties yet.

Demo 08: You can include more than one sort key in the window specification. The row_number values still go from 1 to 23- one for each row.

```
select emp_id, dept_id, salary
, row_number () Over (order by dept_id, salary desc  ) as col_Order
from adv_emp
order by dept_id, salary desc;
```

```
EMP_I    DEPT_ID     SALARY  COL_ORDER
----- ---------- ---------- ----------
309           10      30000          1
321           10      30000          2
319           10      30000          3
311           10      28000          4
305           10      27000          5
320           10      25000          6
322           10      25000          7
310           10      25000          8
301           10      15000          9
323           15      25000         10
312           20      28000         11
303           20      27000         12
302           20      14000         13
315           20      12000         14
314           30      30000         15
304           30      28000         16
306           30      28000         17
316           30      26000         18
317           30      25000         19
308           30      15000         20
307           30      13500         21
318           30      11500         22
313           30      11000         23

23 rows selected.
```

2.1. Partition By

There is another option you can use which does a partition by an attribute.

Demo 09: Row number with a partition. Here we are partitioning the data by the department id; for each new department the row_number restarts at 1

```
select emp_id, dept_id, salary
, row_number () Over (partition by dept_id order by salary ) as col_Order
from adv_emp
order by dept_id, salary ;
```

```
EMP_I    DEPT_ID     SALARY  COL_ORDER
----- ---------- ---------- ----------
301           10      15000          1
320           10      25000          2
310           10      25000          3
322           10      25000          4
305           10      27000          5
311           10      28000          6
319           10      30000          7
321           10      30000          8
309           10      30000          9
323           15      25000          1
315           20      12000          1
302           20      14000          2
303           20      27000          3
312           20      28000          4
```

```
313          30      11000          1
318          30      11500          2
307          30      13500          3
308          30      15000          4
317          30      25000          5
316          30      26000          6
306          30      28000          7
304          30      28000          8
314          30      30000          9

23 rows selected.
```

3. Rank & Dense Rank

Change the query to use the dense_rank function and then the rank function instead of the row_number function. The windowing clause is ordering by salary. The Rank functions determines the rank of data relative to a group of values. The windowing clause is not changed.

Demo 10: Using the DenseRank() function

```
Select emp_id, dept_id, salary
, dense_rank () Over (order by salary desc nulls last  ) as col_Order
from adv_emp;
```

Note the values for the rank column go 1,1,1,1 , 2,2,2,2, 3,3, 4,5,5 because tied rows get the same number, but dense_rank does not skip numbers for ties.

```
EMP_I    DEPT_ID     SALARY  COL_ORDER
----- ---------- ---------- ----------
314          30      30000          1
321          10      30000          1
309          10      30000          1
319          10      30000          1
311          10      28000          2
304          30      28000          2
312          20      28000          2
306          30      28000          2
305          10      27000          3
303          20      27000          3
316          30      26000          4
310          10      25000          5
322          10      25000          5
323          15      25000          5
320          10      25000          5
317          30      25000          5
301          10      15000          6
308          30      15000          6
302          20      14000          7
307          30      13500          8
315          20      12000          9
318          30      11500         10
313          30      11000         11

23 rows selected.
```

Demo 11: Using the Rank() function

```
Select emp_id, dept_id, salary
, rank () Over (order by salary desc nulls last ) as col_Order
from adv_emp;
```

Note the values for the rank column go 1,1,1,1 5,5,5,5 ,. . . because there are four rows tied for first place at salary 30000; then it skips 2,3,4, etc The next set of ties all get value 5 and then the count skips to 9.

```
EMP_I    DEPT_ID     SALARY  COL_ORDER
----- ---------- ---------- ----------
314           30      30000          1
321           10      30000          1
309           10      30000          1
319           10      30000          1
311           10      28000          5
304           30      28000          5
312           20      28000          5
306           30      28000          5
305           10      27000          9
303           20      27000          9
316           30      26000         11
310           10      25000         12
322           10      25000         12
323           15      25000         12
320           10      25000         12
317           30      25000         12
301           10      15000         17
308           30      15000         17
302           20      14000         19
307           30      13500         20
315           20      12000         21
318           30      11500         22
313           30      11000         23
```

You can imagine that people could not agree on which was the "right" way to handle ranks with ties- so they gave us both ways. You need to use the version that makes the most sense to the business situation.

4. Top N

Now we want to get the top six employees in terms of salary. The first problem is to find out what the user wants to do if there are ties. We will assume that the user wants all ties at the last selected position- so we might return 6 rows or more than 6 if there are ties at the last position.

We might try putting the column alias in a Where clause- but that is not valid syntax. Neither can we put the rank or dense_rank function in the Where clause. These functions are allowed in the Select list.

So we go back to a subquery that produces the ranks and a parent query that gets everyone of rank 6 or less. Note that we are using rank, not dense_rank. How many rows would you get with dense_rank?

Demo 12: Top 6 salaries

```
select emp_Id, dept_id, salary, col_Order
from (
      select emp_Id, dept_id, salary
```

```
        , rank() over (order by salary desc nulls last  ) as col_Order
        from adv_emp
     ) tbl
  where col_Order  <= 6;
```

```
 EMP_ID    DEPT_ID     SALARY  COL_ORDER
------ ---------- ---------- ----------
309           10      30000          1
321           10      30000          1
319           10      30000          1
314           30      30000          1
304           30      28000          5
312           20      28000          5
311           10      28000          5
306           30      28000          5

8 rows selected.
```

Demo 13: You can also use the With clause instead of a subquery. I have changed the sort to ascending to get the bottom six salaries

```
  with rankings as (
        select emp_Id, dept_id, salary
        , rank() over (order by salary asc nulls last  ) as col_Order
        from adv_emp
  )
  select emp_Id, dept_id, salary, col_Order
  from rankings
  where col_Order  <= 6;
```

```
EMP_ID    DEPT_ID     SALARY  COL_ORDER
------ ---------- ---------- ----------
313           30      11000          1
318           30      11500          2
315           20      12000          3
307           30      13500          4
302           20      14000          5
301           10      15000          6
308           30      15000          6

7 rows selected.
```

Although we think of an SQL statement as being done "all-at-once" there is an ordering in which the various parts of a select query is processed. If we have a query with a Join and an Order by, the From clause is the first part done- so the Joins are done first, then the Where clause is applied, then the Select and finally the Order By. If we have an analytical function added to the query, it is processed after any Join, Where, Group By or Having clause. It is evaluated just before the final Order By clause.

If we run the following query, only the rows for Dept 30 and 20 will be given a rank.

Demo 14: Rank() within selected departments

```
  select emp_Id, Dept_id, salary
  , rank() Over (order by salary desc  ) as col_Order
  from adv_emp
  where dept_id IN (30, 20);
```

5. Using a grouping

In this query we group by the department ID. We want to show the average salary and we rank over the avg(salary). I truncated the averages to make the output easier to read.

Demo 15: Rank() and Grouping

```
Select dept_id
, trunc(avg(salary)) as "AvgSalary"
, rank() Over (order by avg(salary) desc  ) as col_Order
from adv_emp
group by dept_id
order by dept_id
;
```

```
   DEPT_ID  AvgSalary  COL_ORDER
---------- ---------- ----------
        10      26111          1
        15      25000          2
        20      20250          4
        30      20888          3
```

We do not have to show the avg salary in order to use it in the rank function. Maybe we want to keep the salary amounts secret.

Demo 16: Rank() and Grouping

```
select dept_id
, rank() Over (order by avg(salary) desc  ) as col_Order
from  adv_emp
group by dept_id
;
```

```
   DEPT_ID   COL_ORDER
---------- ----------
        10          1
        15          2
        30          3
        20          4
```

We can use more than one of the ranking functions in a single query.

Demo 17: Using two Rank() expressions

```
Select dept_id
, Min(salary) as MinSalary
, rank() Over (order by min(salary)   ) as MinSalaryRank
, max(salary) as MaxSalary
, rank() Over (order by max(salary)   ) as MaxSalaryRank
from adv_emp
group by dept_id
order by dept_id
;
```

```
   DEPT_ID  MINSALARY MINSALARYRANK  MAXSALARY MAXSALARYRANK
---------- ---------- ------------- ---------- -------------
        10      15000             3      30000             3
        15      25000             4      25000             1
        20      12000             2      28000             2
        30      11000             1      30000             3
```

6. Partition By

The Partition by clause with a ranking function creates groups and restarts the ranking numbers for each group. Here we rank dept 10 first and then dept 15 and then dept 20 and then dept 30.

Demo 18: Rank() with a Partition

```
Select dept_id, emp_id, year_hired, salary
, rank () Over (partition by Dept_id order by salary   ) as col_Order
from adv_emp;
```

```
  DEPT_ID EMP_I YEAR_HIRED     SALARY  COL_ORDER
---------- ----- ---------- ---------- ----------
        10 301         2010      15000          1
        10 320         2012      25000          2
        10 310         2012      25000          2
        10 322         2012      25000          2
        10 305         2012      27000          5
        10 311         2012      28000          6
        10 319         2014      30000          7
        10 321         2014      30000          7
        10 309         2014      30000          7
        15 323         2012      25000          1
        20 315         2013      12000          1
        20 302         2010      14000          2
        20 303         2014      27000          3
        20 312         2010      28000          4
        30 313         2012      11000          1
        30 318         2014      11500          2
        30 307         2012      13500          3
        30 308         2013      15000          4
        30 317         2014      25000          5
        30 316         2013      26000          6
        30 306         2010      28000          7
        30 304         2010      28000          7
        30 314         2013      30000          9

23 rows selected.
```

You can have more than one partition attribute. The following restarts the numbering for each department and for each year within the department.

Demo 19: Partition by two attributes

```
select dept_id, year_hired, emp_id, salary
, rank()Over(partition by Dept_id, year_hired order by salary ) as col_Order
from adv_emp;
```

```
  DEPT_ID YEAR_HIRED EMP_I     SALARY  COL_ORDER
---------- ---------- ----- ---------- ----------
        10       2010 301        15000          1
        10       2012 320        25000          1
        10       2012 322        25000          1
        10       2012 310        25000          1
        10       2012 305        27000          4
        10       2012 311        28000          5
        10       2014 319        30000          1
        10       2014 309        30000          1
        10       2014 321        30000          1
        15       2012 323        25000          1
        20       2010 302        14000          1
        20       2010 312        28000          2
        20       2013 315        12000          1
        20       2014 303        27000          1
        30       2010 304        28000          1
        30       2010 306        28000          1
        30       2012 313        11000          1
        30       2012 307        13500          2
        30       2013 308        15000          1
        30       2013 316        26000          2
        30       2013 314        30000          3
        30       2014 318        11500          1
        30       2014 317        25000          2

23 rows selected.
```