Table of Contents

Ranking functions are used to rank rows of data according to some criteria. . 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 are

- NTile
- Islands and Gaps
- Lead and Lag
- Percent_rank (optional)
- Cume_Dist (optional)

Oracle seldom develops functions that are not useful in business.

1. Ntile

Ntile works by dividing the data into percentile groupings, called buckets. Suppose you want the salaries divided into 2 groups (the top 50% and the bottom 50%). The number of buckets is an argument to NTILE

Demo 01: NTILE() based on salary

```
Select NTile(5) Over (order by salary ) as Bucket
, emp_id, dept_id, year_hired, salary
from adv_emp;
```

We have 23 rows with non null salaries and we get one group with 12 rows and one with 11.

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

Note the employees who earn 25000 ; there are 5 of these employees- three in bucket 2 and 2 in the next bucket. We could add a second sort key to decide on the group if that make business sense. Perhaps we want people with earlier hire date to be in the lower numbered group. You should not make up a distinction rule that has no business rule to support it. But if we were giving out bonuses based on the employee's bucket- we would have to deal with this appropriately.

Demo 02: Suppose we want to have a different set of buckets for each department. We can partition by the dept_id. This starts a new bucketing for each department. I also use a variable for the bucket count.

```
variable d number;
exec :d := 2;

select
  NTile (:d) Over (partition by dept_id order by salary, year_hired ) as Bucket
, dept_id, year_hired, salary, emp_id
from adv_emp
;
```

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

Demo 03: Perhaps we don't want to make buckets for small departments and we want to improve the output a bit. One CTE puts together the bucket list and the second CTE finds the department with more than 8 employees. These are joined in the main query.

```
variable NumBkts number;
exec :NumBkts := 2;
variable MinDept number;
exec :MinDept := 8;

With
Bucket_list as(
   Select
     NTile (:NumBkts) Over (partition by dept_id order by salary, year_hired )
as nt
   , emp_id, name_last, dept_id
   from adv_emp
```

```
      )
    ,
   Groups as (
      select dept_id, count(*) as deptCount
      from adv_emp
      group by dept_id
      having count(*) >= :MinDept
      )
   select ' Group:' || cast(groups.dept_id as varchar(5) )|| '-' || cast(nt as
   varchar(2)) as StudyGroup
   , emp_id, name_last
   from Groups
   join Bucket_list on Groups.dept_id = Bucket_list.dept_id
   ;
```

```
STUDYGROUP      EMP_ID NAME_LAST
--------------- ------ -------------------------
 Group:10-1     301    Green
 Group:10-1     320    Jarrett
 Group:10-1     310    Wabich
 Group:10-1     322    Wabich
 Group:10-1     305    Coltrane
 Group:10-2     311    Brubeck
 Group:10-2     319    Redman
 Group:10-2     321    Rollins
 Group:10-2     309    Beiderbecke
 Group:30-1     313    Davis
 Group:30-1     318    Shorter
 Group:30-1     307    Tatum
 Group:30-1     308    Evans
 Group:30-1     317    Wasliewski
 Group:30-2     316    Monk
 Group:30-2     306    Cohen
 Group:30-2     304    Mobley
 Group:30-2     314    Turrentine
```

2. Islands

Islands are sequences of values with no gaps. For example; these are the order id values between 400 and 510 from the order headers table. I have color coded each of the islands

```
ORD_ID
----------
       400
       401
       402
       405
       407
       408
       411
       412
       413
       414
       415
```

Demo 04: This is the code to get the first value in an island and the last value and the count.

```
with dataset as (
select order_id, order_id - dense_rank() over ( order by order_id) as grp
from oe_orderheaders
where order_id between 400 and 510
)
select MIN(order_id) as start_of_range
, MAX(order_id) as end_of_range
, COUNT(order_id) as number_in_range
from dataset
group by  grp
order by 1;
```

```
START_OF_RANGE END_OF_RANGE NUMBER_IN_RANGE
-------------- ------------ ---------------
           400          402               3
           405          405               1
           407          408               2
           411          415               5
```

How does that work? Take a look at the CTE; I have added another column for the rank. Note that for each island the difference between the ord_id and the rank is the same value. And for different islands, that column has a different value. So we can group by the difference value to put all of the rows in the island in the same group.

```
select order_id
, dense_rank() over ( order by order_id) as DRank
, order_id - dense_rank() over ( order by order_id) as Diff
from oe_orderheaders
where order_id between 400 and 510;
```

```
    ORD_ID      DRANK       DIFF
---------- ---------- ----------
       400          1        399
       401          2        399
       402          3        399
       405          4        401
       407          5        402
       408          6        402
       411          7        404
       412          8        404
       413          9        404
       414         10        404
       415         11        404
```

Demo 05: Suppose we want to find data about our customers and how consistently they place orders over months. I am going to use the books table and orders for the year 2014
For the demo I limited this to three selected customers: `224038, 227105, 272787.` First, display the order dates for those customers. I have inserted spaces between each customer.

```
select cust_id,  order_Date
from  bk_order_headers
where extract ( YEAR from order_Date) = 2015
and cust_id in (224038, 227105, 272787)
order by cust_id, order_date;
```

```
   CUST_ID ORDER_DAT
```

```
---------- ---------
      CUST_ID ORDER_DATE
---------- ----------
    224038 02-MAY-15
    224038 26-MAY-15
    227105 12-FEB-15
    227105 12-FEB-15
    227105 12-JUN-15
    227105 30-JUL-15
    227105 03-AUG-15
    227105 12-AUG-15
    227105 12-AUG-15
    227105 12-AUG-15
    227105 12-SEP-15
    227105 19-SEP-15
    227105 18-NOV-15
    227105 06-DEC-15
    227105 12-DEC-15
    227105 12-DEC-15
    227105 31-DEC-15
    272787 15-FEB-15
    272787 02-MAR-15
    272787 12-MAR-15
    272787 13-MAR-15
    272787 08-APR-15
    272787 15-JUN-15
    272787 16-JUN-15
    272787 02-JUL-15
    272787 22-SEP-15
    272787 22-SEP-15
    272787 22-SEP-15
    272787 30-SEP-15
    272787 02-NOV-15
    272787 02-NOV-15
    272787 06-NOV-15
    272787 12-NOV-15
    272787 12-NOV-15
    272787 02-DEC-15

 35 rows selected.
```

This is the query to display the islands for those customers.

```
with dataset as (
    select  cust_id
    , extract( month from order_Date) as mn
    , extract( month from order_Date) - dense_rank()
     over (partition by cust_id order by extract( month from order_Date))as grp
    from bk_order_headers
    where extract ( YEAR from order_Date) = 2015
    and cust_id in (224038, 227105, 272787)
)
select cust_id
, MIN(mn) as month_start
, MAX(mn) as month_end
from dataset
group by cust_id, grp
order by cust_id, grp;
```

```
   CUST_ID MONTH_START  MONTH_END
---------- ----------- ----------
    224038           5          5
    227105           2          2
    227105           6          9
    227105          11         12
    272787           2          4
    272787           6          7
    272787           9          9
    272787          11         12

8 rows selected.
```

If you are going to work with actual date values, then you will need to use date arithmetic function to get the difference. Work with the CTE expression to get the same difference value for each island.

3. Lead and Lag

It might be useful to see the data for one day and also the previous day's data on the same line. For this we can use the Lag function. The Lag function also uses an Over clause and the default is one value previous. There is also a Lead function that shows the next value. If there is no previous (or next) value then a Null is returned.

Demo 06: Lag(attrb) and Lead(attrb)

```
select sales_day
, sales
, Lag (sales) Over (order by sales_day) as PrevDay
, Lead(sales) Over (order by sales_day) as NextDay
from adv_sales
order by sales_day;
```

```
       DAY      SALES    PREVDAY    NEXTDAY
---------- ---------- ---------- ----------
25-APR-15         400                   400
26-APR-15         400        400        400
27-APR-15         400        400        300
28-APR-15         300        400        900
29-APR-15         900        300        580
30-APR-15         580        900        425
01-MAY-15         425        580         10
02-MAY-15          10        425        325
03-MAY-15         325         10        500
04-MAY-15         500        325        550
. . .  rows omitted
```

You can specify how many days Lag you want. The nulls here show where the missing data would be.

Demo 07: Lag(attr, n)

```
select  sales_day
, sales
, Lag (sales, 3) Over (order by sales_day) as "3_DaysAgo"
, Lag (sales, 2) Over (order by sales_day) as "2_DaysAgo"
, Lag (sales, 1) Over (order by sales_day) as "1_DayAgo"
, Lag (sales, 0) Over (order by sales_day) as "ThisDay"
from adv_sales
order by sales_day;
```

```
       DAY      SALES  3_DaysAgo  2_DaysAgo   1_DayAgo    ThisDay
---------- ---------- ---------- ---------- ---------- ----------
25-APR-15         400                                         400
26-APR-15         400                              400        400
27-APR-15         400                   400        400        400
28-APR-15         300        400        400        400        300
29-APR-15         900        400        400        300        900
30-APR-15         580        400        300        900        580
01-MAY-15         425        300        900        580        425
02-MAY-15          10        900        580        425         10
03-MAY-15         325        580        425         10        325
04-MAY-15         500        425         10        325        500
05-MAY-15         550         10        325        500        550
. . .
```

There is a third argument to supply a value for the nulls that occur as a return from the Lag and Lead. This value is not used for a null that occurs in the table data.

4. Gaps

The missing values between the islands are called gaps. You can use Lead to find the gaps. This is using the order id demo from above.

Demo 08:

```
   with dataset as (
      select order_id as TheCurrentID
      , lead(order_id) over ( order by order_id) as    TheNextID
      from oe_orderheaders
      where order_id between 400 and 520
   )
   select TheCurrentID + 1  as startOfGap
   , TheNextID - 1 as EndOfGap
   from dataset
   where  TheNextID - TheCurrentID  >1;
```

```
STARTOFGAP   ENDOFGAP
---------- ----------
       403        404
       406        406
       409        410
       416        518

4 rows selected.
```

5. Percent_Rank and Cume_Dist

These are two more ranking functions

Demo 09: The row filters are to reduce the output to make this easier to understand

```
   Select emp_id, salary
   , Rank() Over (order by salary   ) as rank
   , round(PERCENT_Rank() Over (order by salary), 2 ) as Percentrank
   , round(CUME_DIST() Over (order by salary), 2 )  as CumeDist
   from adv_emp
   where salary is not null
   and dept_id in (30)
   ;
```

```
EMP_ID SALARY       RANK PERCENTRANK   CUMEDIST
------ ------ ---------- ----------- ----------
313     11000          1           0        .11
318     11500          2         .13        .22
307     13500          3         .25        .33
308     15000          4         .38        .44
317     25000          5          .5        .56
316     26000          6         .63        .67
306     28000          7         .75        .89
304     28000          7         .75        .89
314     30000          9           1          1
```

These two functions are similar in that the ranking values increase as the rank increases.

The Percent_rank starts at 0 for the first row and then calculates its value as

```
          (rank -1) / (number_of_rows -1 )
```

The lowest ranked item is 0% and the highest is 100%. If there are ties for the highest position then we do not achieve 100%. If we have ties for the lowest position, then those tied rows are all 0%

Cume_Dist returns a value between 0 and 1 which is the number of rows ranked lower than or equal to the current row, including the current row, divided by the number of rows in the partition. The demo above does not do a partition so the number of rows is based on the table.

I know these can be pretty intimidating (and look a lot like statistics) but I hope you can see that these are important techniques that businesses need to use. And if you take these slowly, you can work them out.