ANALYTICAL SQL

DECEMBER 2021

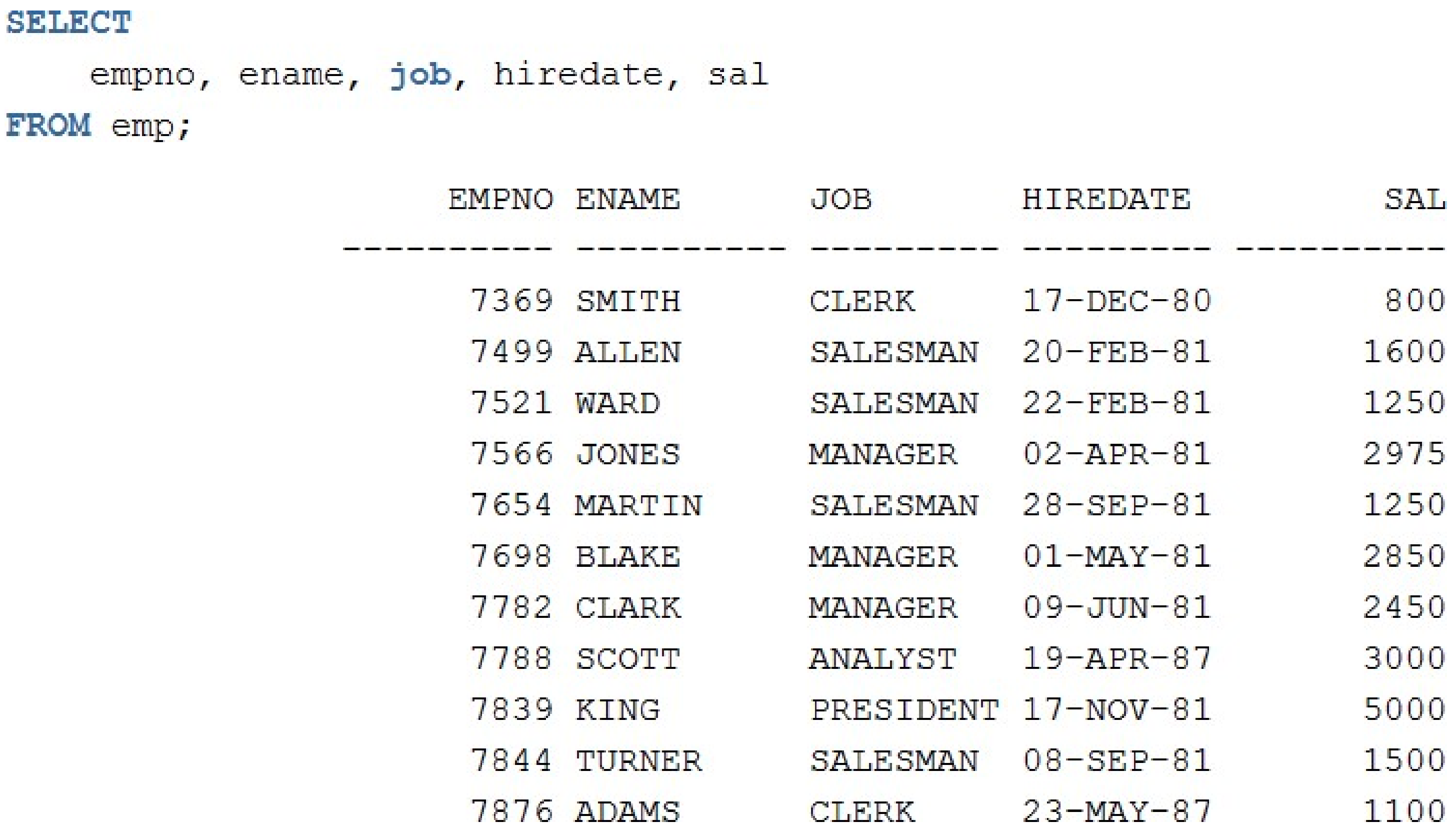
Agenda

- **Day 1**
  - Introduction to Analytical SQL
  - Analytic window functions
  - Ranking functions
  - LAB
- **Day 2**
  - Aggregate Analytical functions
  - Windowing
  - LAB
- **Day 3**
  - Pivoting operations
  - Statistical Aggregates
  - Case Study

Introduction to Analytical SQL

- Example : With Vs. Without Analytical Function
- Why Analytical Functions?
- Aggregates vs. Analytics
- Syntax & Execution Sequence

Example : With Vs. Without Analytical Function

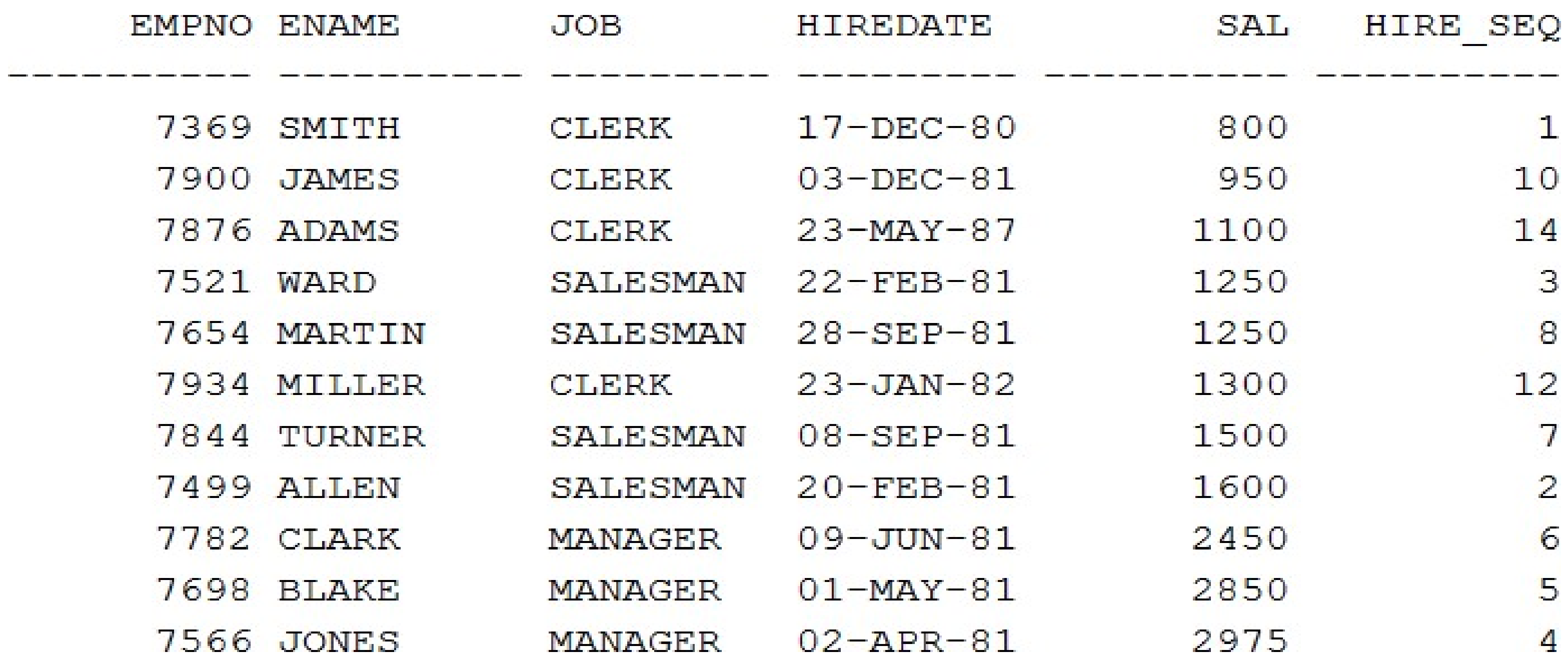


```
SELECT
    empno, ename, job, hiredate, sal
FROM emp;
```

```
     EMPNO ENAME      JOB       HIREDATE         SAL
---------- ---------- --------- --------- ----------
      7369 SMITH      CLERK     17-DEC-80        800
      7499 ALLEN      SALESMAN  20-FEB-81       1600
      7521 WARD       SALESMAN  22-FEB-81       1250
      7566 JONES      MANAGER   02-APR-81       2975
      7654 MARTIN     SALESMAN  28-SEP-81       1250
      7698 BLAKE      MANAGER   01-MAY-81       2850
      7782 CLARK      MANAGER   09-JUN-81       2450
      7788 SCOTT      ANALYST   19-APR-87       3000
      7839 KING       PRESIDENT 17-NOV-81       5000
      7844 TURNER     SALESMAN  08-SEP-81       1500
      7876 ADAMS      CLERK     23-MAY-87       1100
```

Example : With Vs. Without Analytical Function

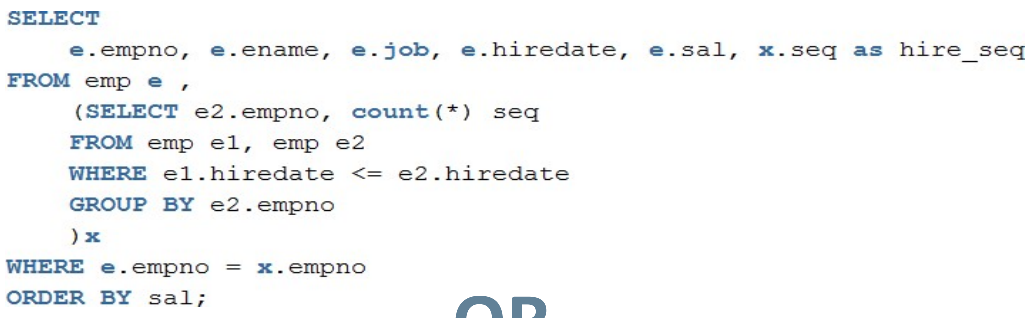
The sequence in which everyone joined the company “HIRE_SEQ” Ordered by Salary



```
     EMPNO ENAME      JOB       HIREDATE         SAL   HIRE_SEQ
---------- ---------- --------- --------- ---------- ----------
      7369 SMITH      CLERK     17-DEC-80        800          1
      7900 JAMES      CLERK     03-DEC-81        950         10
      7876 ADAMS      CLERK     23-MAY-87       1100         14
      7521 WARD       SALESMAN  22-FEB-81       1250          3
      7654 MARTIN     SALESMAN  28-SEP-81       1250          8
      7934 MILLER     CLERK     23-JAN-82       1300         12
      7844 TURNER     SALESMAN  08-SEP-81       1500          7
      7499 ALLEN      SALESMAN  20-FEB-81       1600          2
      7782 CLARK      MANAGER   09-JUN-81       2450          6
      7698 BLAKE      MANAGER   01-MAY-81       2850          5
      7566 JONES      MANAGER   02-APR-81       2975          4
```

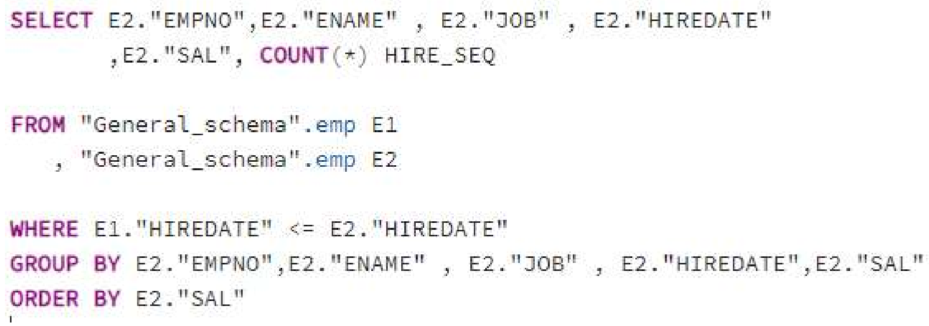
Example : With Vs. Without Analytical Function

Without Analytical Functions?



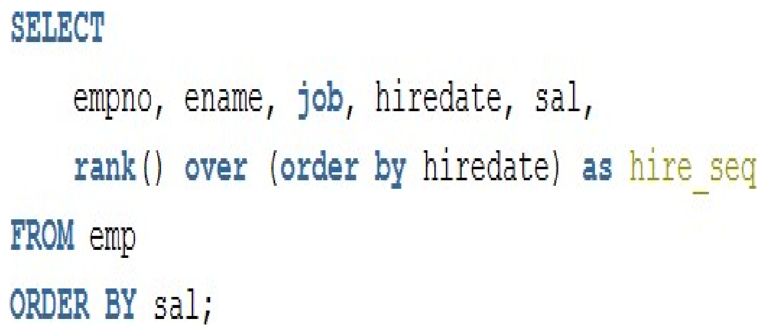
```
SELECT
    e.empno, e.ename, e.job, e.hiredate, e.sal, x.seq as hire_seq
FROM emp e ,
    (SELECT e2.empno, count(*) seq
    FROM emp e1, emp e2
    WHERE e1.hiredate <= e2.hiredate
    GROUP BY e2.empno
    )x
WHERE e.empno = x.empno
ORDER BY sal;
```

OR



```
SELECT E2."EMPNO",E2."ENAME" , E2."JOB" , E2."HIREDATE"
       ,E2."SAL", COUNT(*) HIRE_SEQ

FROM "General_schema".emp E1
   , "General_schema".emp E2

WHERE E1."HIREDATE" <= E2."HIREDATE"
GROUP BY E2."EMPNO",E2."ENAME" , E2."JOB" , E2."HIREDATE",E2."SAL"
ORDER BY E2."SAL"
```

With Analytical Functions?



```
SELECT
    empno, ename, job, hiredate, sal,
    rank() over (order by hiredate) as hire_seq
FROM emp
ORDER BY sal;
```

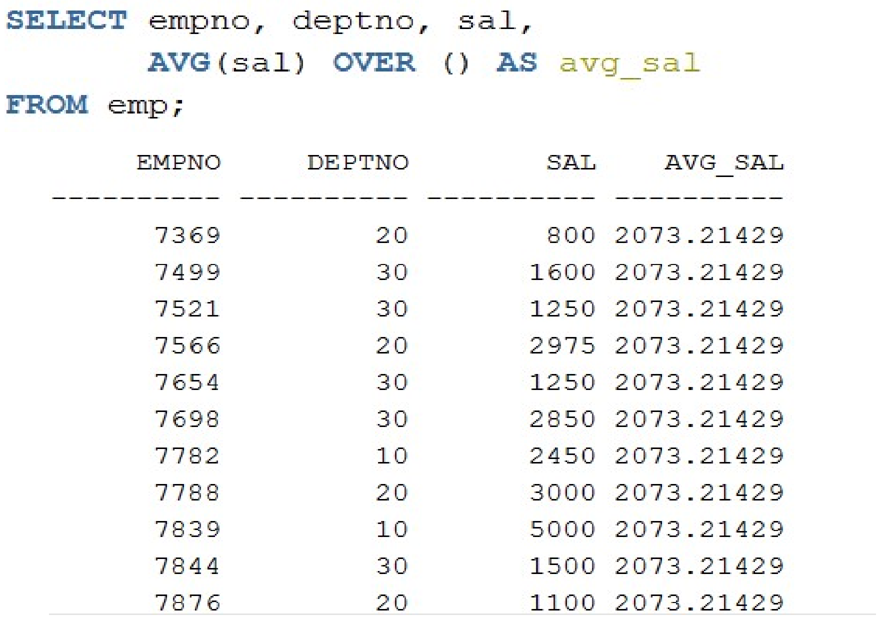
Why Analytic Functions?

- Ability to see one row from another row’s perspective
- Avoid self-join queries
- Summary data in detailed rows
- Slice and dice within the results

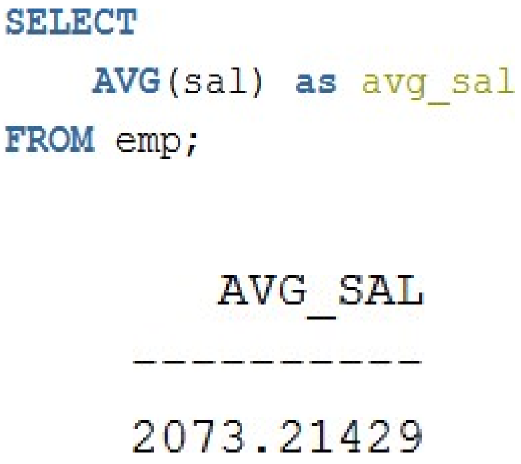
Aggregates vs. Analytics

- Aggregate functions
  - Rows are collapsed. One row per group
  - Group-BY columns must exist in the SELECT list
- Analytic functions
  - Rows are not collapsed
  - As many rows in the output as in the input
  - No restrictions on the columns in the SELECT list
  - Can appear only in the SELECT or ORDER BY clause
  - Evaluated after joins, WHERE, GROUP BY, HAVING clauses

Aggregates vs. Analytics (Example)



```
SELECT empno, deptno, sal,
       AVG(sal) OVER () AS avg_sal
FROM emp;

     EMPNO     DEPTNO        SAL    AVG_SAL
---------- ---------- ---------- ----------
      7369         20        800 2073.21429
      7499         30       1600 2073.21429
      7521         30       1250 2073.21429
      7566         20       2975 2073.21429
      7654         30       1250 2073.21429
      7698         30       2850 2073.21429
      7782         10       2450 2073.21429
      7788         20       3000 2073.21429
      7839         10       5000 2073.21429
      7844         30       1500 2073.21429
      7876         20       1100 2073.21429
```



```
SELECT
    AVG(sal) as avg_sal
FROM emp;

   AVG_SAL
----------
2073.21429
```

Simplified Syntax

```
FUNCTION(<arg>,<arg>,…)
OVER (
<partition clause>
<sorting clause>
<windowing clause>
)
```

Execution Sequence

1. Table Joins
2. WHERE clause filters
3. GROUP BY
4. HAVING
5. Analytic Functions
6. DISTINCT
7. ORDER BY

Analytical Window Functions

- OVER Clause
- PARTITION BY Clause
- ORDER BY Clause
  - LEAD/LAG Functions

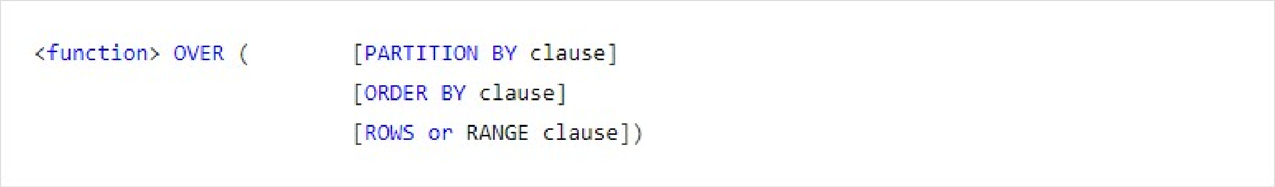
Over clause

The OVER clause is used to determine

when the function’s calculations should restart ==> (PARTITION BY)

what order they are evaluated in by that function ==> (ORDER BY)

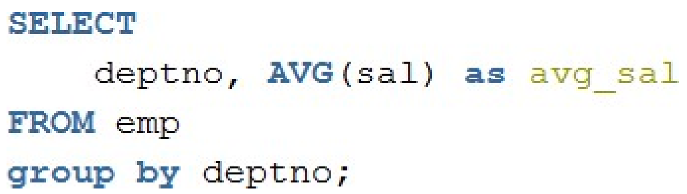
which rows from the query are applied to the function ==> (ROWS or **RANGE)**



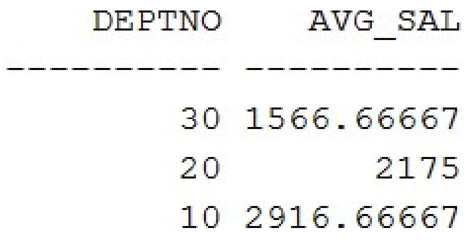
```
<function> OVER (        [PARTITION BY clause]
                         [ORDER BY clause]
                         [ROWS or RANGE clause])
```

In looking at the syntax, it appears that all of the sub-clauses are optional. In fact, each function that can use the OVER clause determines which of the sub-clauses are allowed, and which are required. Depending on the function being used, the OVER clause itself may be optional (more details about the functions will be discussed later).

Calculate the average salary per department



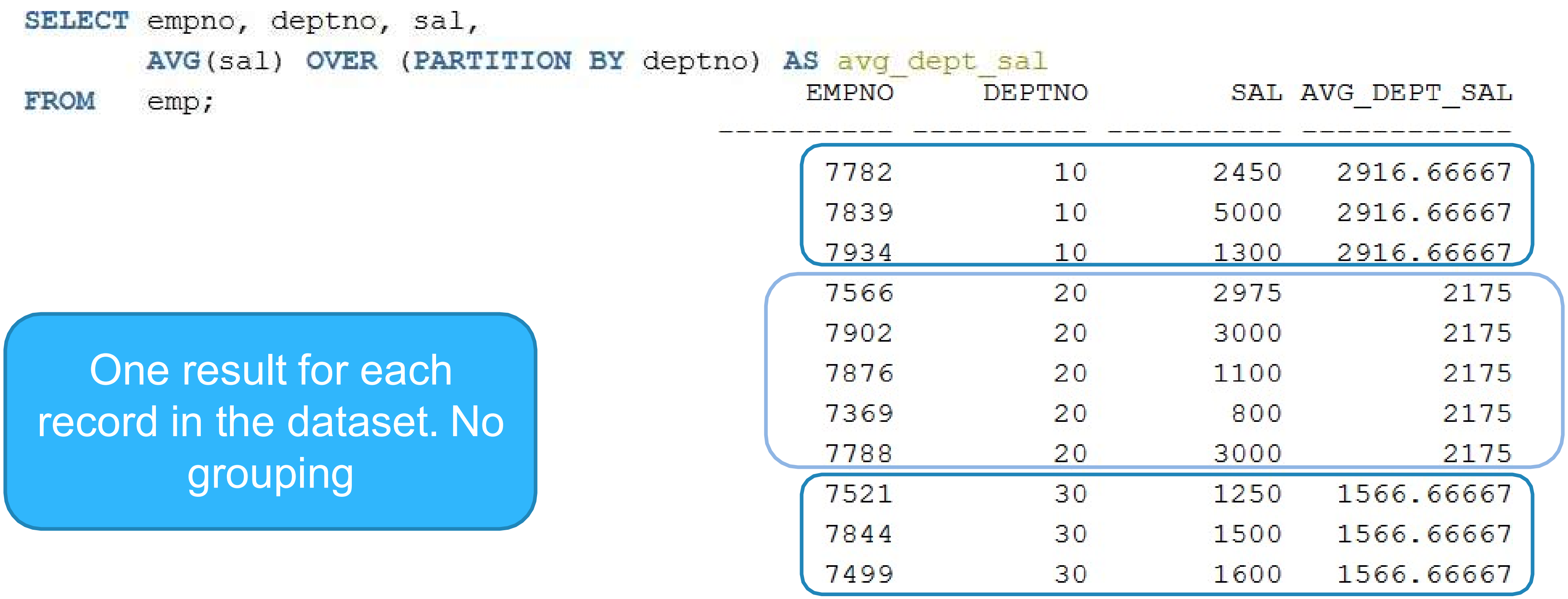
```
SELECT
    deptno, AVG(sal) as avg_sal
FROM emp
group by deptno;
```



```
    DEPTNO    AVG_SAL
---------- ----------
        30 1566.66667
        20       2175
        10 2916.66667
```

One record for each group

OVER (PARTITION BY..)



```
SELECT empno, deptno, sal,
       AVG(sal) OVER (PARTITION BY deptno) AS avg_dept_sal
FROM   emp;
```

```
     EMPNO     DEPTNO        SAL AVG_DEPT_SAL
---------- ---------- ---------- ------------
      7782         10       2450   2916.66667
      7839         10       5000   2916.66667
      7934         10       1300   2916.66667
      7566         20       2975         2175
      7902         20       3000         2175
      7876         20       1100         2175
      7369         20        800         2175
      7788         20       3000         2175
      7521         30       1250   1566.66667
      7844         30       1500   1566.66667
      7499         30       1600   1566.66667
```

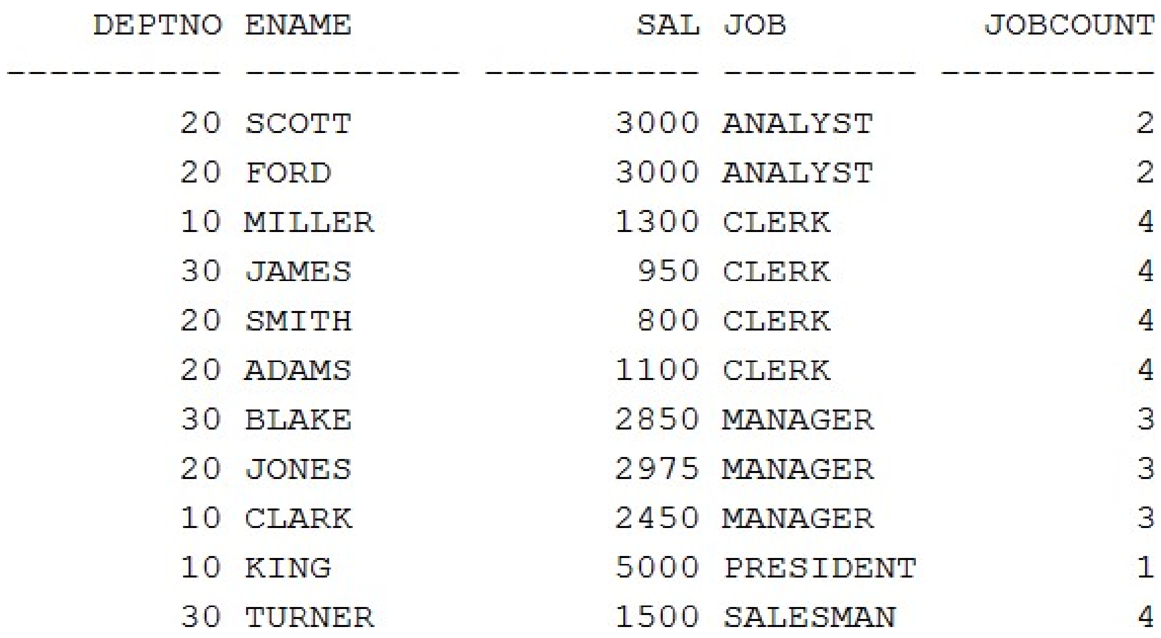
One result for each record in the dataset. No grouping

PARTITION BY Clause

- The query_partition_clause divides the result set into partitions, or groups of data.
- The operation of the analytic function is restricted to the boundary imposed by these partitions, similar to the way a GROUP BY clause affects the action of an aggregate function.
- If the query_partition_clause is omitted, the whole result set is treated as a single partition.

HANDS ON

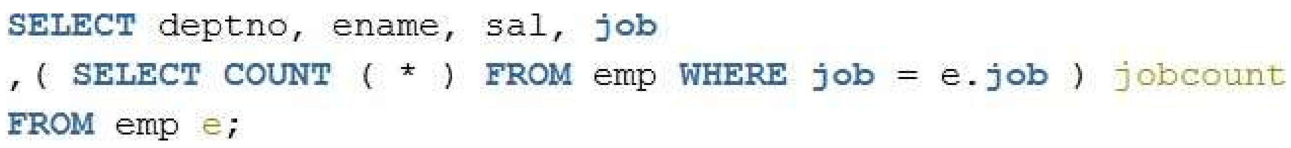
Calculate the number of employees per job



```
    DEPTNO ENAME             SAL JOB         JOBCOUNT
---------- ---------- ---------- --------- ----------
        20 SCOTT            3000 ANALYST            2
        20 FORD             3000 ANALYST            2
        10 MILLER           1300 CLERK              4
        30 JAMES             950 CLERK              4
        20 SMITH             800 CLERK              4
        20 ADAMS            1100 CLERK              4
        30 BLAKE            2850 MANAGER            3
        20 JONES            2975 MANAGER            3
        10 CLARK            2450 MANAGER            3
        10 KING             5000 PRESIDENT          1
        30 TURNER           1500 SALESMAN           4
```

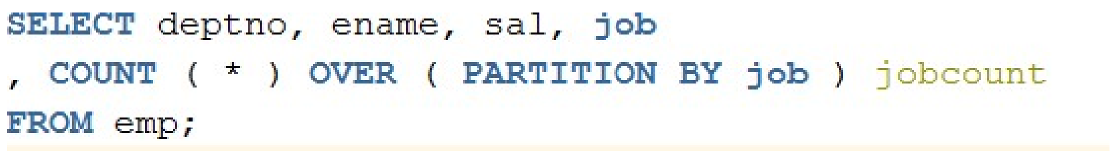
HANDS ON

- With no analytical function:



```
SELECT deptno, ename, sal, job
,( SELECT COUNT ( * ) FROM emp WHERE job = e.job ) jobcount
FROM emp e;
```

- With analytical function:



```
SELECT deptno, ename, sal, job
, COUNT ( * ) OVER ( PARTITION BY job ) jobcount
FROM emp;
```

ORDER BY Clause

- The query_order_clause controls the order that the rows are evaluated by the function

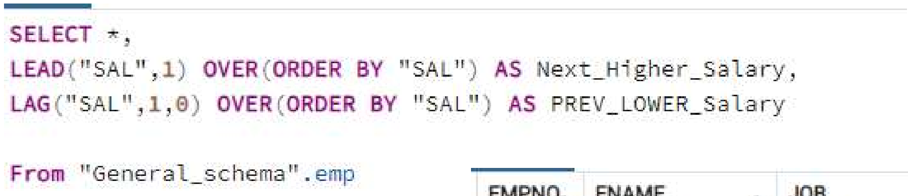
- The query_order_clause defines the logical order of the rows within each partition of the result set.

- ASC | DESC
  Specifies that the values in the specified column should be sorted in ascending or descending order. ASC is the default sort order.

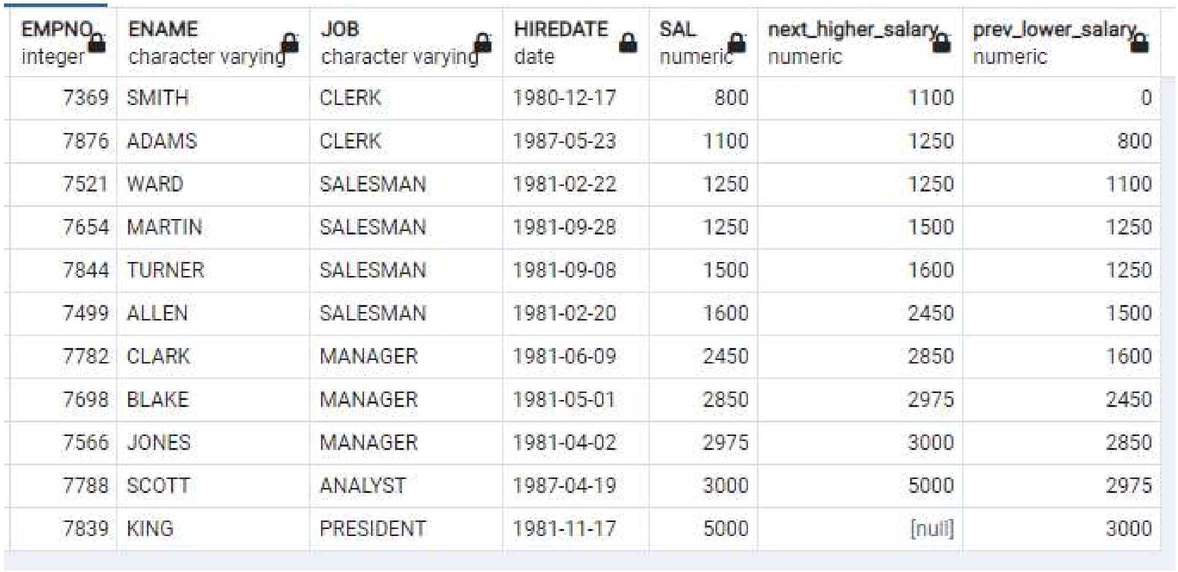
LEAD and LAG Functions

- Return the value from a field when looking one record (or more) behind/ahead using the order specified
- Syntax : LAG (field_name, <num_recs>, <default_value>) OVER (ORDER BY field_name)
- ORDER BY is required
- Optional second parameter to look more than one record

LEAD and LAG Functions (Example)



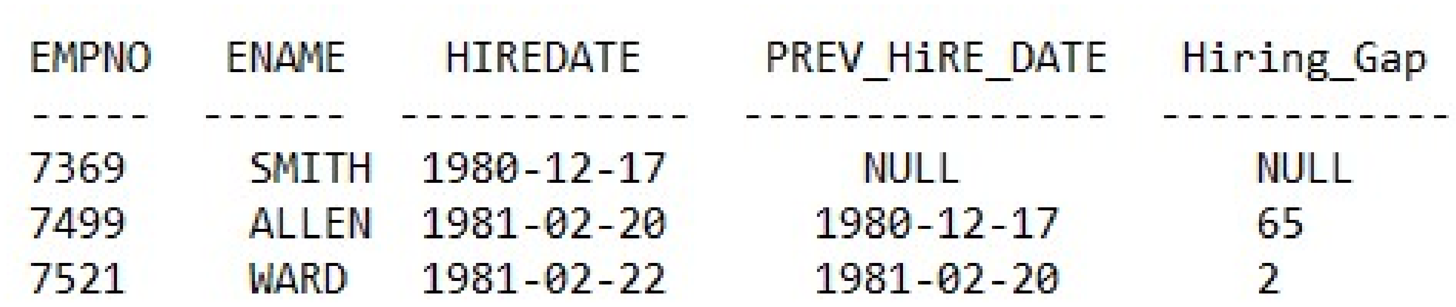
```
SELECT *,
LEAD("SAL",1) OVER(ORDER BY "SAL") AS Next_Higher_Salary,
LAG("SAL",1,0) OVER(ORDER BY "SAL") AS PREV_LOWER_Salary

From "General_schema".emp
```



EMPNO integer	ENAME character varying	JOB character varying	HIREDATE date	SAL numeric	next_higher_salary numeric	prev_lower_salary numeric
7369	SMITH	CLERK	1980-12-17	800	1100	0
7876	ADAMS	CLERK	1987-05-23	1100	1250	800
7521	WARD	SALESMAN	1981-02-22	1250	1250	1100
7654	MARTIN	SALESMAN	1981-09-28	1250	1500	1250
7844	TURNER	SALESMAN	1981-09-08	1500	1600	1250
7499	ALLEN	SALESMAN	1981-02-20	1600	2450	1500
7782	CLARK	MANAGER	1981-06-09	2450	2850	1600
7698	BLAKE	MANAGER	1981-05-01	2850	2975	2450
7566	JONES	MANAGER	1981-04-02	2975	3000	2850
7788	SCOTT	ANALYST	1987-04-19	3000	5000	2975
7839	KING	PRESIDENT	1981-11-17	5000	[null]	3000

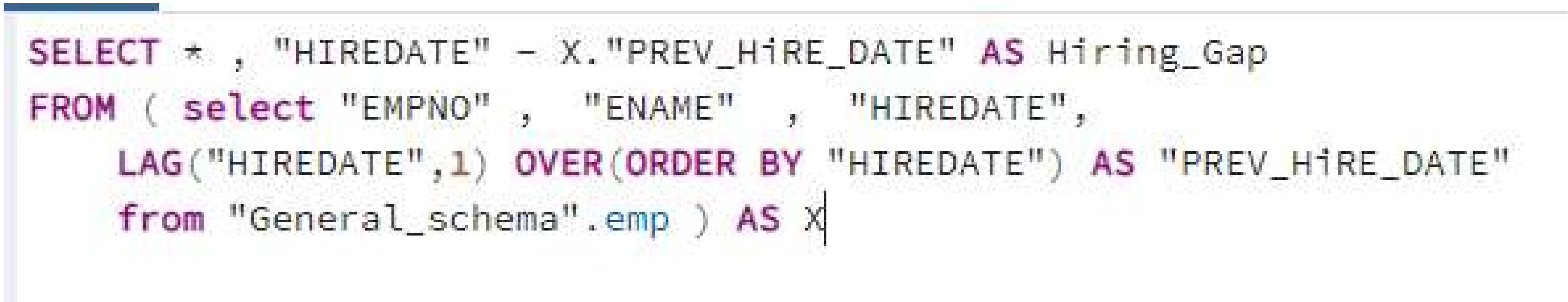
HANDS ON

Get the previous hire date for each employee the calculate the hiring gap in days



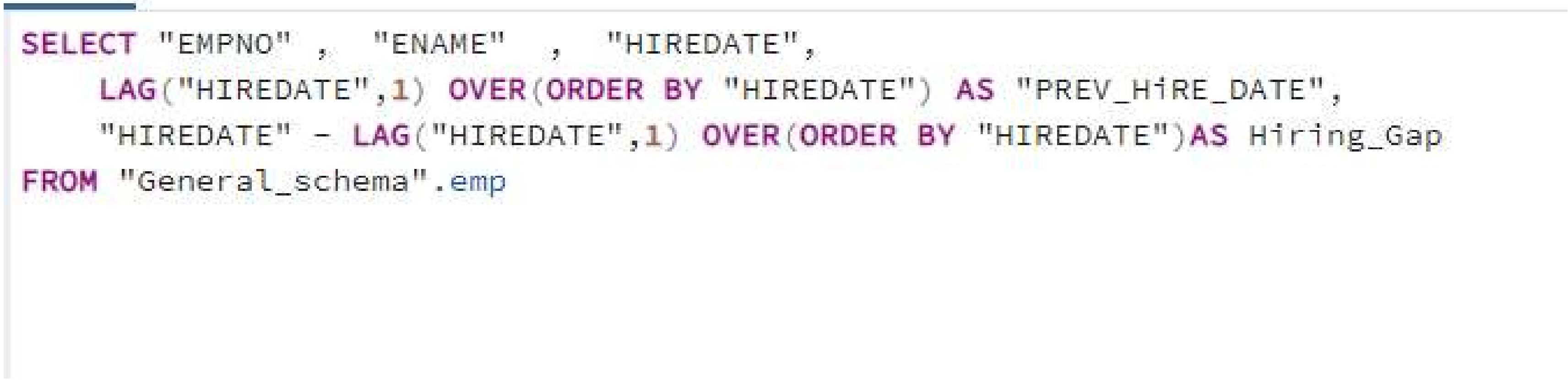
```
EMPNO   ENAME    HIREDATE       PREV_HiRE_DATE   Hiring_Gap
-----  ------  ------------   ---------------  ------------
7369     SMITH  1980-12-17          NULL             NULL
7499     ALLEN  1981-02-20       1980-12-17          65
7521     WARD   1981-02-22       1981-02-20          2
```

HANDS ON



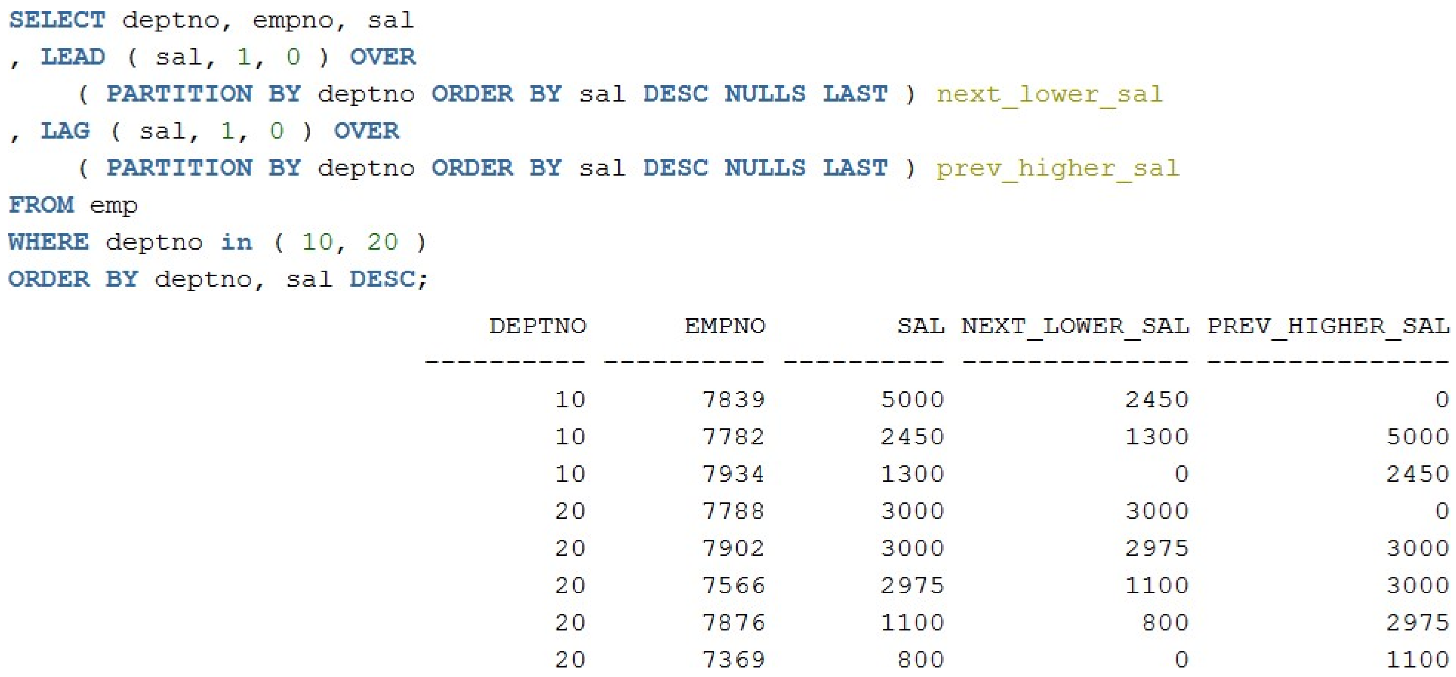
```
SELECT * , "HIREDATE" - X."PREV_HiRE_DATE" AS Hiring_Gap
FROM ( select "EMPNO" ,  "ENAME"  ,  "HIREDATE",
    LAG("HIREDATE",1) OVER(ORDER BY "HIREDATE") AS "PREV_HiRE_DATE"
    from "General_schema".emp ) AS X
```

OR



```
SELECT "EMPNO" ,  "ENAME"  ,  "HIREDATE",
    LAG("HIREDATE",1) OVER(ORDER BY "HIREDATE") AS "PREV_HiRE_DATE",
    "HIREDATE" - LAG("HIREDATE",1) OVER(ORDER BY "HIREDATE")AS Hiring_Gap
FROM "General_schema".emp
```

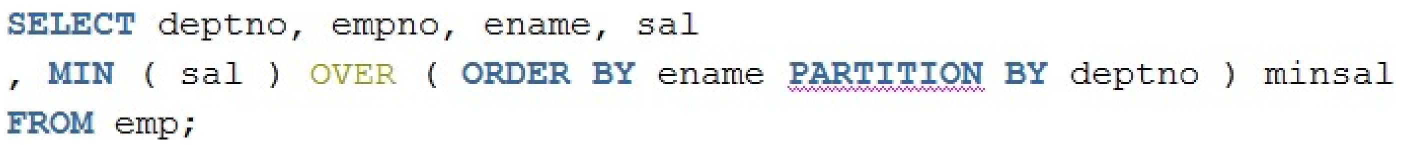
PARTITION BY & ORDER BY



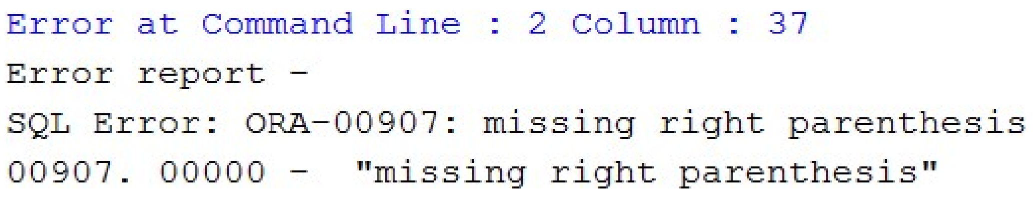
```
SELECT deptno, empno, sal
, LEAD ( sal, 1, 0 ) OVER
    ( PARTITION BY deptno ORDER BY sal DESC NULLS LAST ) next_lower_sal
, LAG ( sal, 1, 0 ) OVER
    ( PARTITION BY deptno ORDER BY sal DESC NULLS LAST ) prev_higher_sal
FROM emp
WHERE deptno in ( 10, 20 )
ORDER BY deptno, sal DESC;
```

```
    DEPTNO      EMPNO        SAL NEXT_LOWER_SAL PREV_HIGHER_SAL
---------- ---------- ---------- -------------- ---------------
        10       7839       5000           2450               0
        10       7782       2450           1300            5000
        10       7934       1300              0            2450
        20       7788       3000           3000               0
        20       7902       3000           2975            3000
        20       7566       2975           1100            3000
        20       7876       1100            800            2975
        20       7369        800              0            1100
```

Order Of Items In Analytic Clause



```
SELECT deptno, empno, ename, sal
, MIN ( sal ) OVER ( ORDER BY ename PARTITION BY deptno ) minsal
FROM emp;
```



```
Error at Command Line : 2 Column : 37
Error report -
SQL Error: ORA-00907: missing right parenthesis
00907. 00000 -  "missing right parenthesis"
```

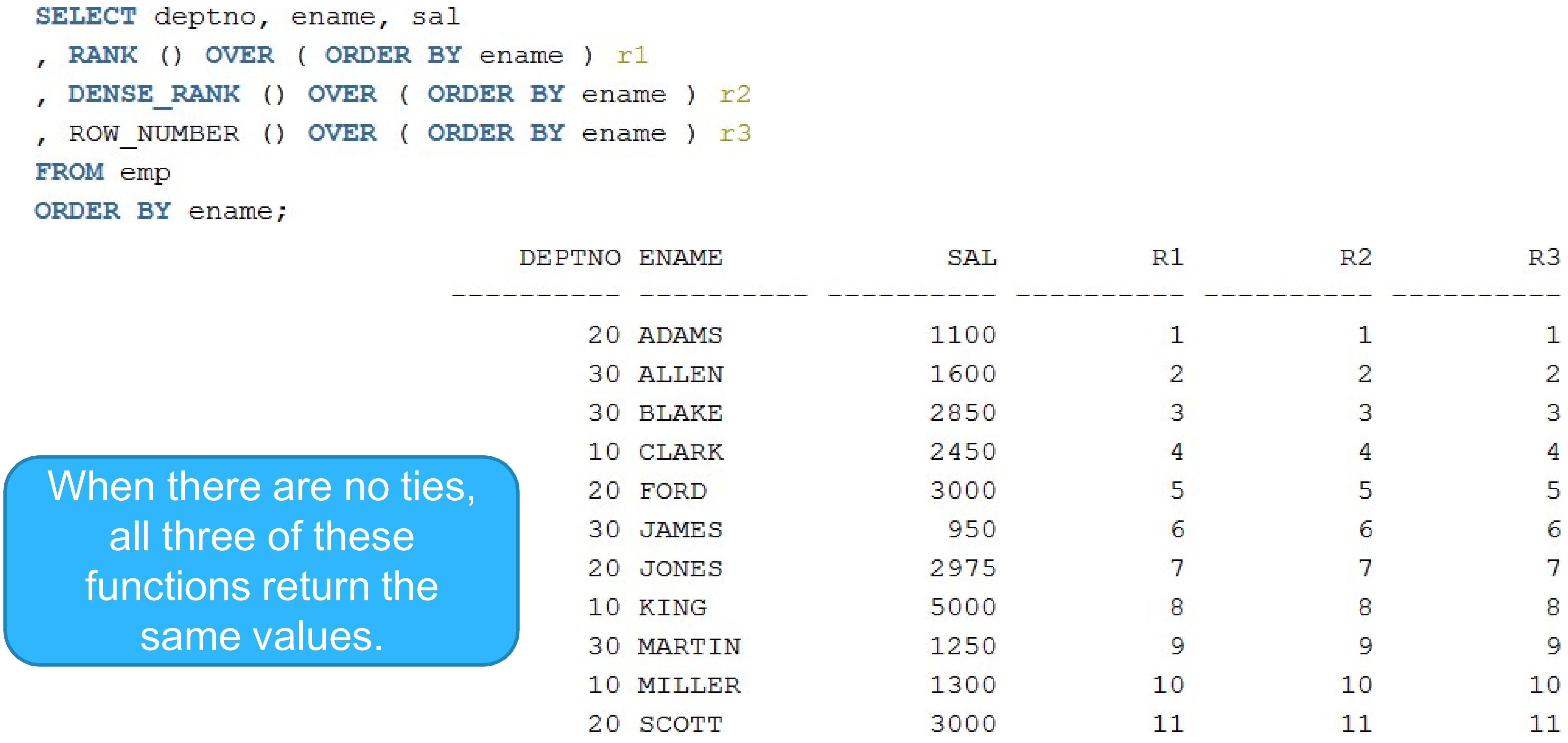
Components must be in correct order

Ranking Functions

- RANK , DENSE_RANK and ROW_NUMBER Function
- FIRST_VALUE and LAST_VALUE Function
- PERCENT_RANK Function
- NTILE Function
- CUME_DIST Function

Ranking Functions

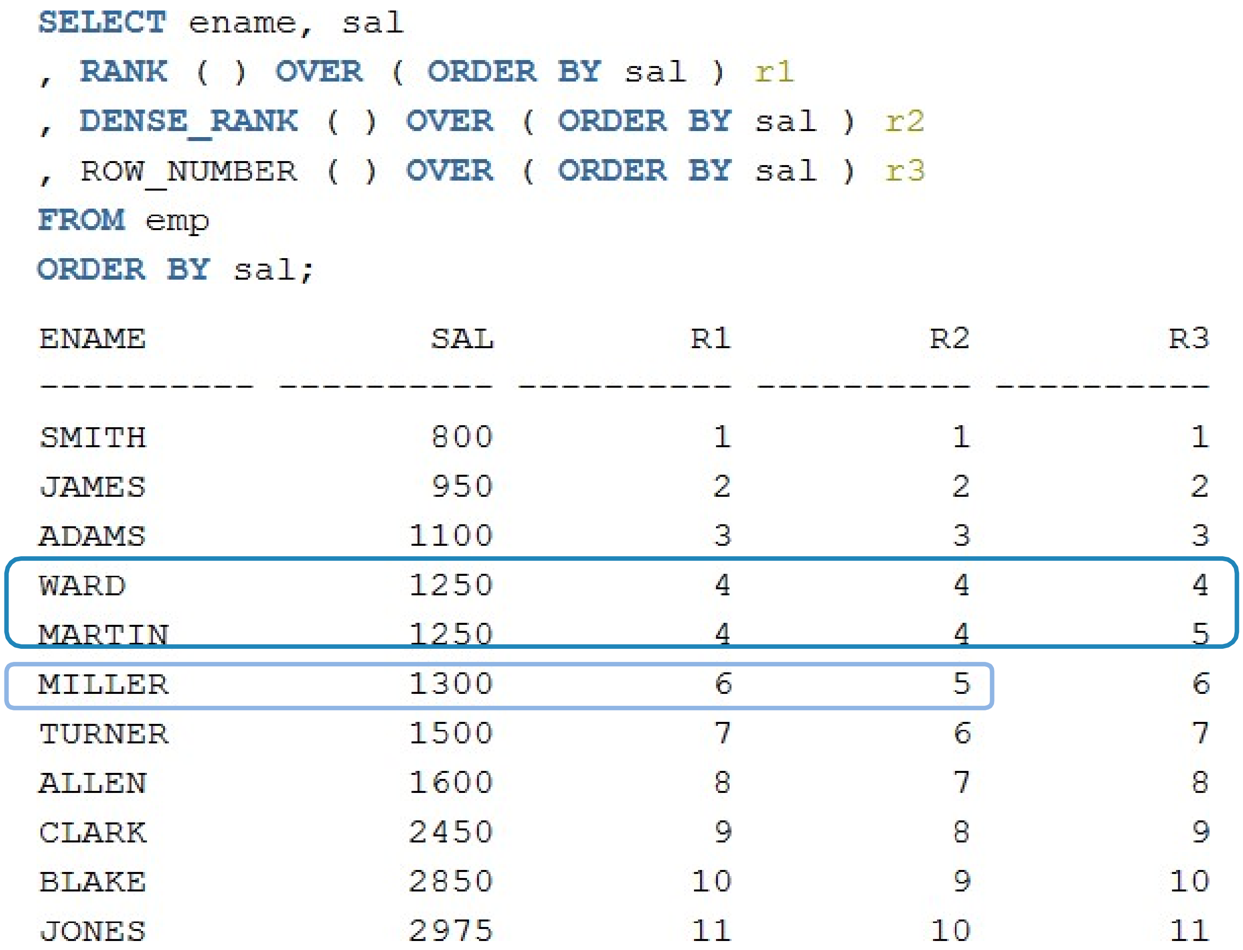
- Where does this record fall, when the records are placed in a certain order?
- Ordering (Ranking) functions:
  - RANK
  - DENSE_RANK
  - ROW_NUMBER
- Syntax:
  - RANK ( ) OVER (ORDER BY field_name)
- ORDER BY expression is mandatory for Ranking function
- All three functions return a number (Rank)
- Difference between functions is how they handle ties



```
SELECT deptno, ename, sal
, RANK () OVER ( ORDER BY ename ) r1
, DENSE_RANK () OVER ( ORDER BY ename ) r2
, ROW_NUMBER () OVER ( ORDER BY ename ) r3
FROM emp
ORDER BY ename;
```

```
    DEPTNO ENAME             SAL         R1         R2         R3
---------- ---------- ---------- ---------- ---------- ----------
        20 ADAMS            1100          1          1          1
        30 ALLEN            1600          2          2          2
        30 BLAKE            2850          3          3          3
        10 CLARK            2450          4          4          4
        20 FORD             3000          5          5          5
        30 JAMES             950          6          6          6
        20 JONES            2975          7          7          7
        10 KING             5000          8          8          8
        30 MARTIN           1250          9          9          9
        10 MILLER           1300         10         10         10
        20 SCOTT            3000         11         11         11
```

When there are no ties, all three of these functions return the same values.



```
SELECT ename, sal
, RANK ( ) OVER ( ORDER BY sal ) r1
, DENSE_RANK ( ) OVER ( ORDER BY sal ) r2
, ROW_NUMBER ( ) OVER ( ORDER BY sal ) r3
FROM emp
ORDER BY sal;

ENAME             SAL         R1         R2         R3
---------- ---------- ---------- ---------- ----------
SMITH             800          1          1          1
JAMES             950          2          2          2
ADAMS            1100          3          3          3
WARD             1250          4          4          4
MARTIN           1250          4          4          5
MILLER           1300          6          5          6
TURNER           1500          7          6          7
ALLEN            1600          8          7          8
CLARK            2450          9          8          9
BLAKE            2850         10          9         10
JONES            2975         11         10         11
```

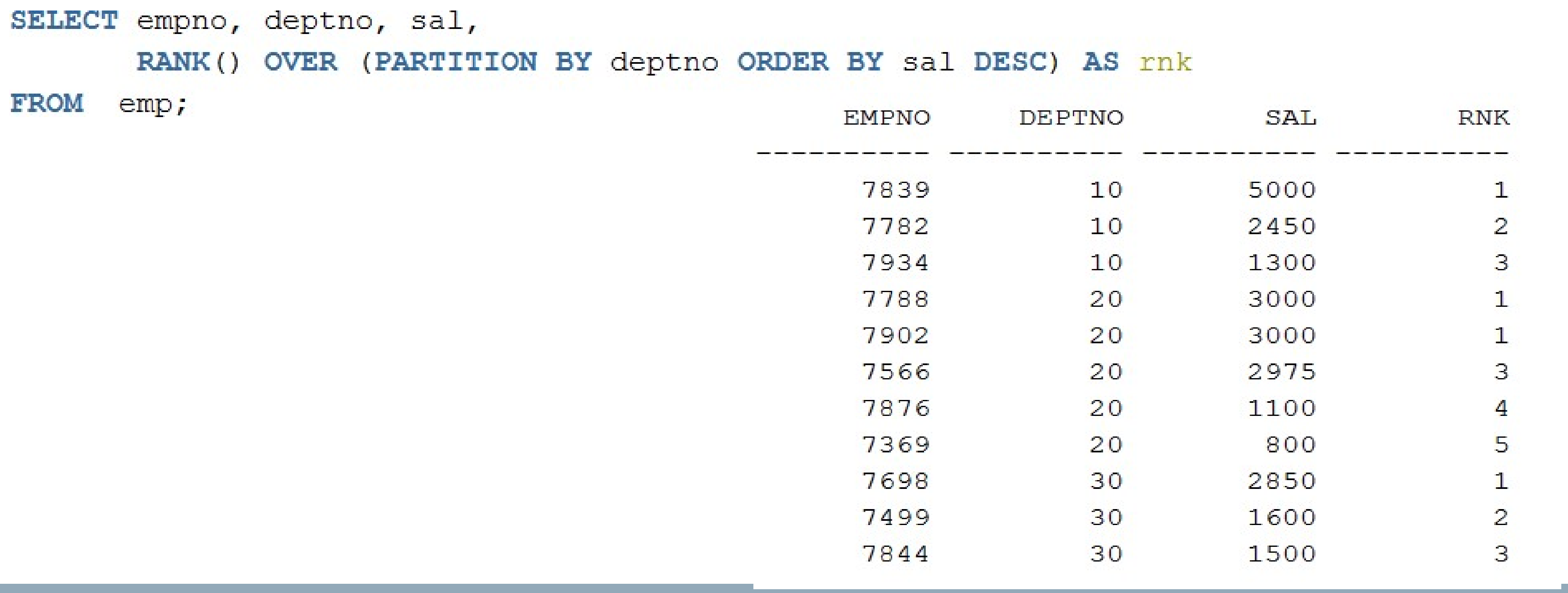
RANK and DENSE_RANK will assign the same number to multiple records with the same sort value

The difference is in how each one handles the record which follows

ROW_NUMBER assigns a unique number to each record. The highest value assigned by ROW_NUMBER will be equal to COUNT(*)

Find out the highest salary per department

- Step 1:

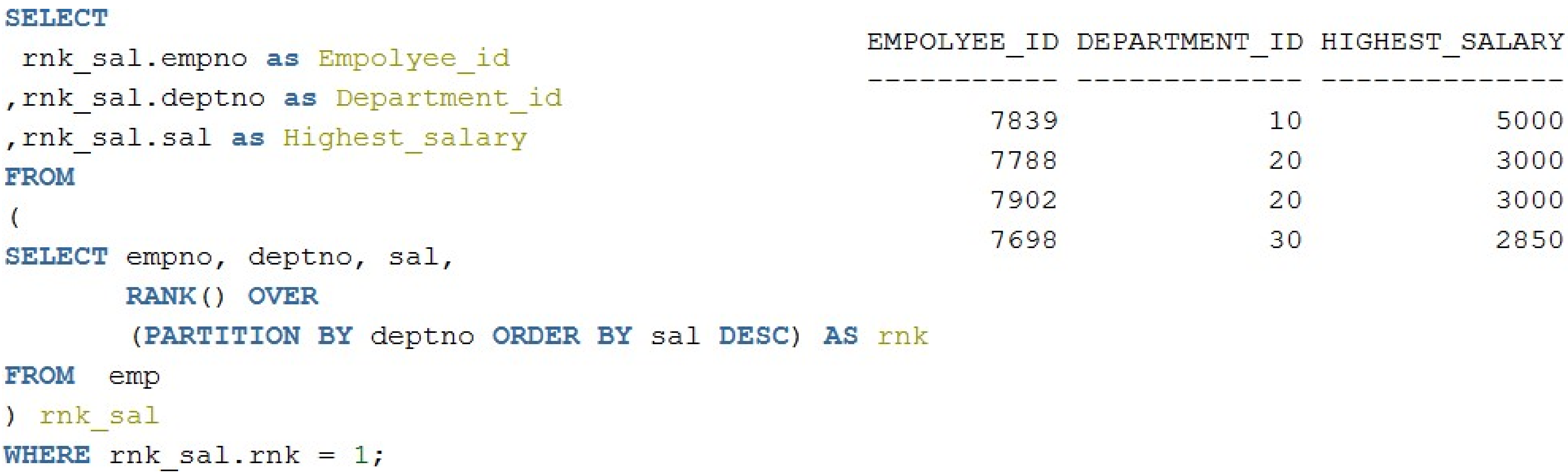


```
SELECT empno, deptno, sal,
       RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS rnk
FROM  emp;
```

```
     EMPNO     DEPTNO        SAL        RNK
---------- ---------- ---------- ----------
      7839         10       5000          1
      7782         10       2450          2
      7934         10       1300          3
      7788         20       3000          1
      7902         20       3000          1
      7566         20       2975          3
      7876         20       1100          4
      7369         20        800          5
      7698         30       2850          1
      7499         30       1600          2
      7844         30       1500          3
```

Find out the highest salary per department

- Step 2:

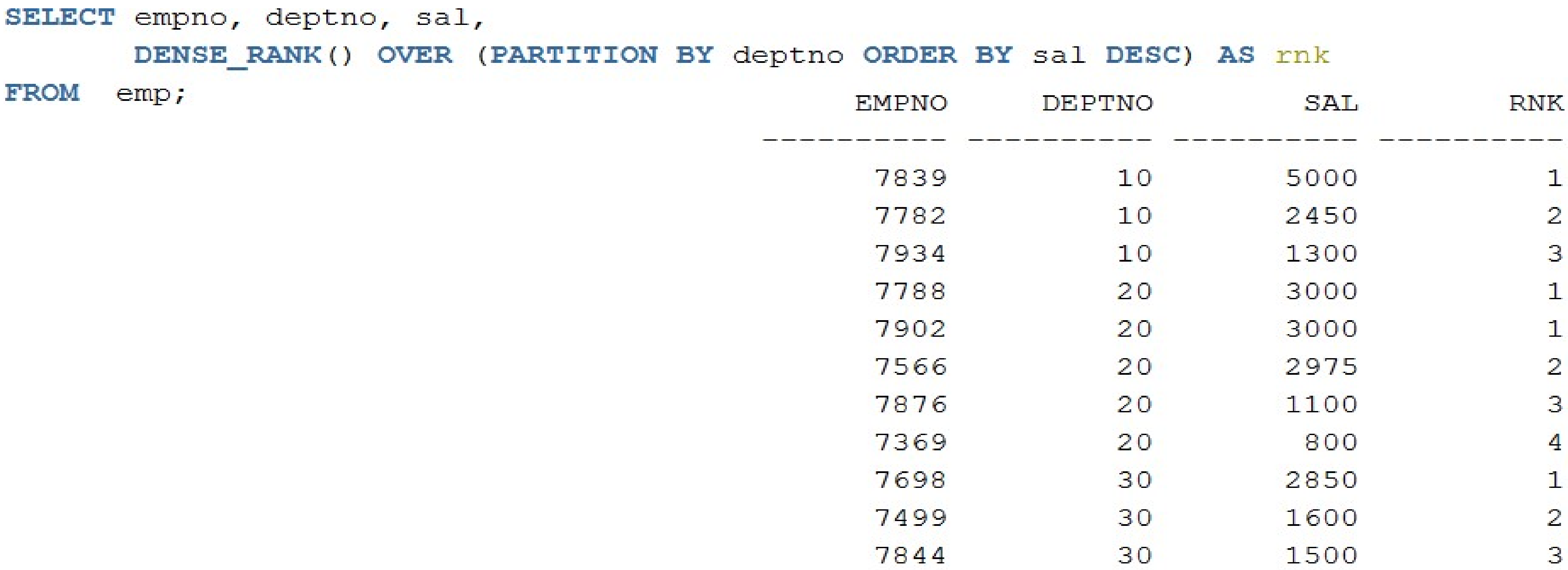


```
SELECT
 rnk_sal.empno as Empolyee_id
,rnk_sal.deptno as Department_id
,rnk_sal.sal as Highest_salary
FROM
(
SELECT empno, deptno, sal,
       RANK() OVER
       (PARTITION BY deptno ORDER BY sal DESC) AS rnk
FROM  emp
) rnk_sal
WHERE rnk_sal.rnk = 1;
```

```
EMPOLYEE_ID DEPARTMENT_ID HIGHEST_SALARY
----------- ------------- --------------
       7839            10           5000
       7788            20           3000
       7902            20           3000
       7698            30           2850
```

Find out the highest salary per department

- Step 1:

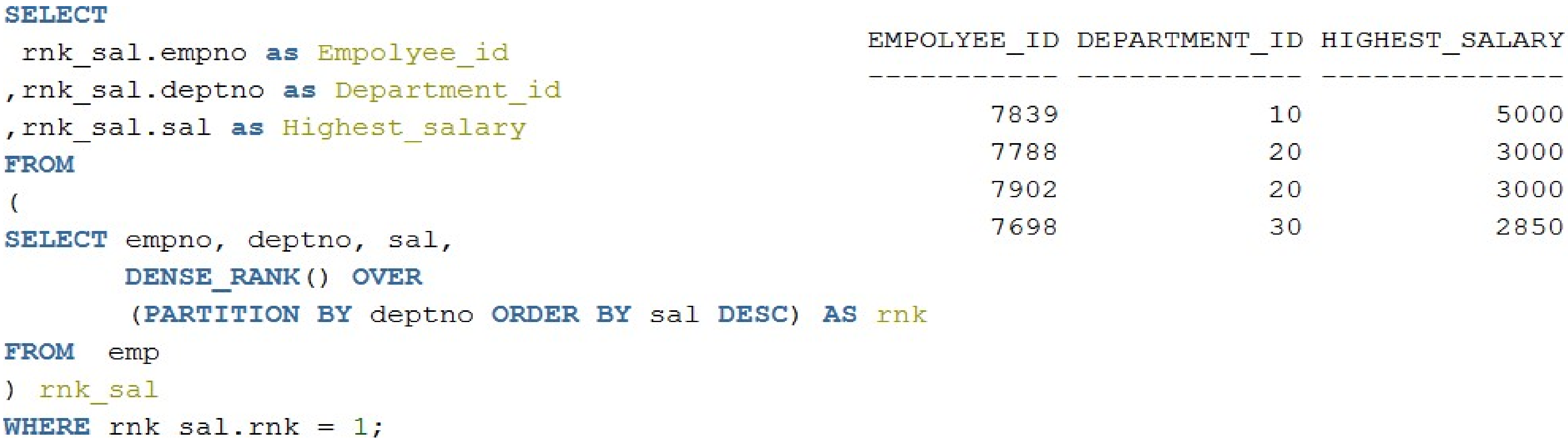


```
SELECT empno, deptno, sal,
       DENSE_RANK() OVER (PARTITION BY deptno ORDER BY sal DESC) AS rnk
FROM  emp;
```

```
     EMPNO     DEPTNO        SAL        RNK
---------- ---------- ---------- ----------
      7839         10       5000          1
      7782         10       2450          2
      7934         10       1300          3
      7788         20       3000          1
      7902         20       3000          1
      7566         20       2975          2
      7876         20       1100          3
      7369         20        800          4
      7698         30       2850          1
      7499         30       1600          2
      7844         30       1500          3
```

Find out the highest salary per department

- Step 2:

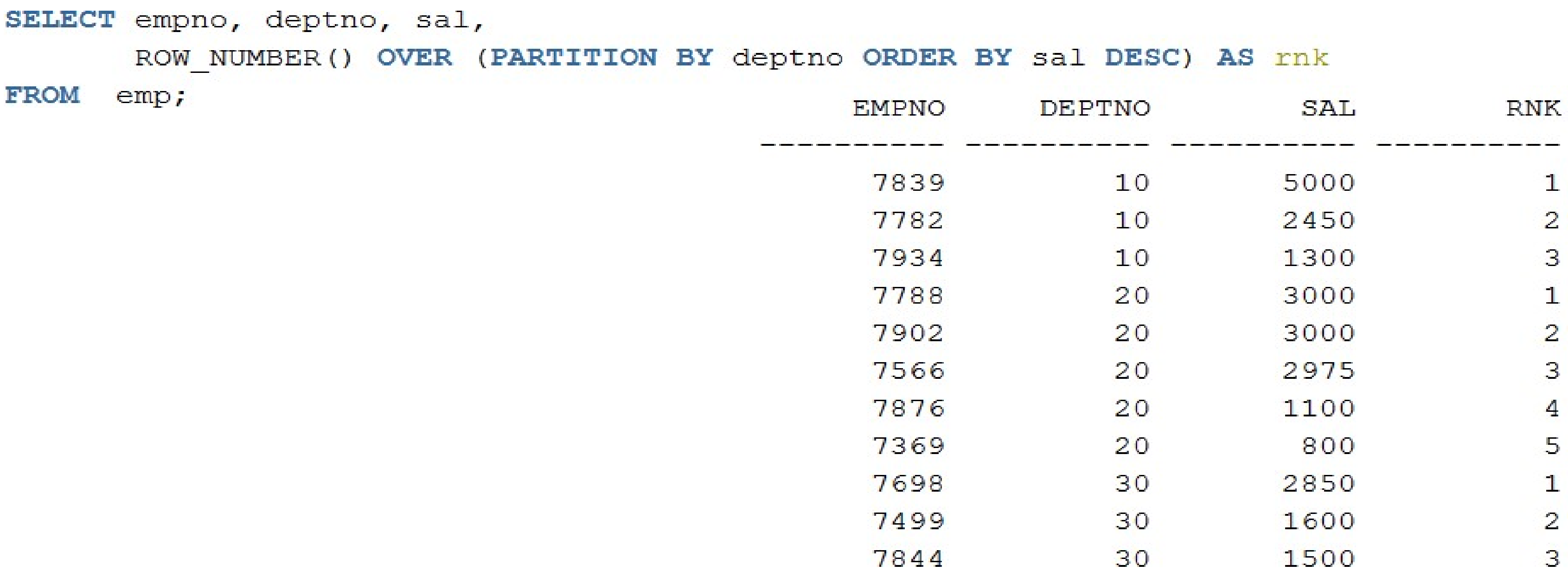


```
SELECT
 rnk_sal.empno as Empolyee_id
,rnk_sal.deptno as Department_id
,rnk_sal.sal as Highest_salary
FROM
(
SELECT empno, deptno, sal,
       DENSE_RANK() OVER
       (PARTITION BY deptno ORDER BY sal DESC) AS rnk
FROM  emp
) rnk_sal
WHERE rnk sal.rnk = 1;
```

```
EMPOLYEE_ID DEPARTMENT_ID HIGHEST_SALARY
----------- ------------- --------------
       7839            10           5000
       7788            20           3000
       7902            20           3000
       7698            30           2850
```

Find out the highest salary per department

- Step 1:

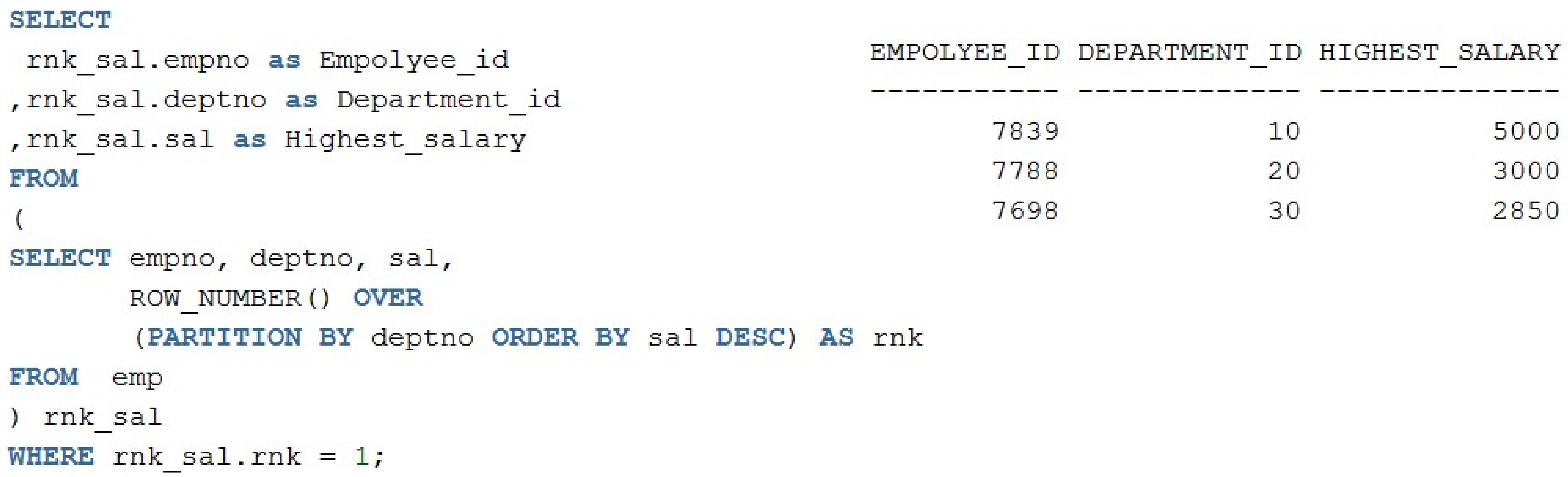


```
SELECT empno, deptno, sal,
       ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sal DESC) AS rnk
FROM   emp;
```

```
     EMPNO     DEPTNO        SAL        RNK
---------- ---------- ---------- ----------
      7839         10       5000          1
      7782         10       2450          2
      7934         10       1300          3
      7788         20       3000          1
      7902         20       3000          2
      7566         20       2975          3
      7876         20       1100          4
      7369         20        800          5
      7698         30       2850          1
      7499         30       1600          2
      7844         30       1500          3
```

Find out the highest salary per department

- Step 2:

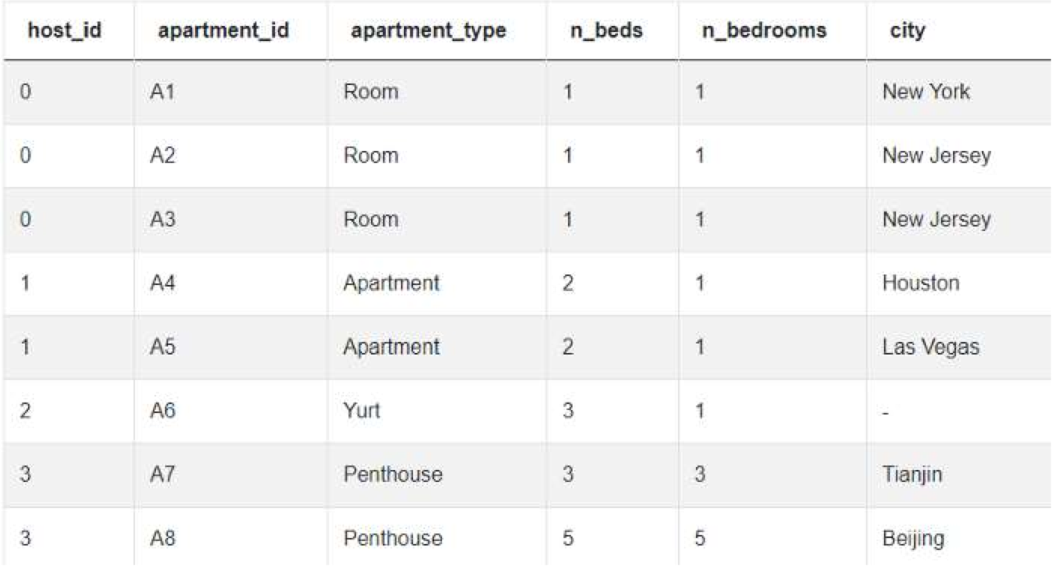


```
SELECT
 rnk_sal.empno as Empolyee_id
,rnk_sal.deptno as Department_id
,rnk_sal.sal as Highest_salary
FROM
(
SELECT empno, deptno, sal,
       ROW_NUMBER() OVER
       (PARTITION BY deptno ORDER BY sal DESC) AS rnk
FROM  emp
) rnk_sal
WHERE rnk_sal.rnk = 1;
```

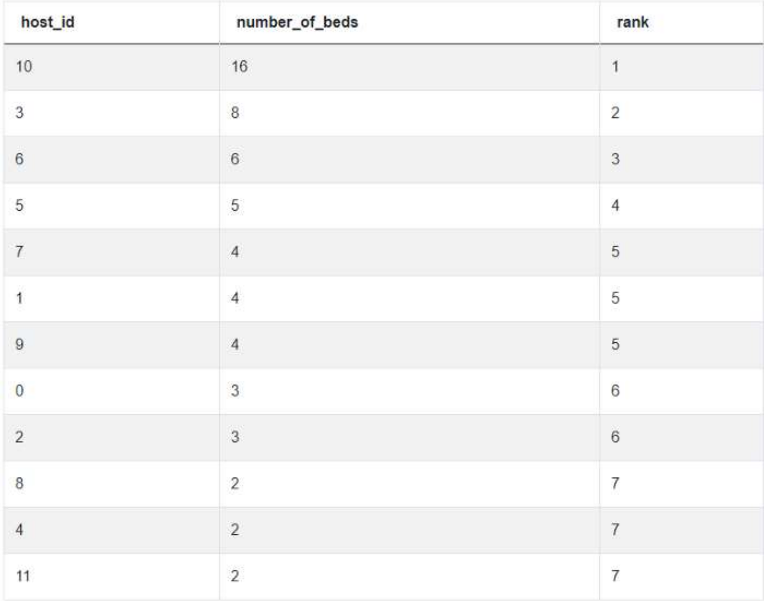
```
EMPOLYEE_ID DEPARTMENT_ID HIGHEST_SALARY
----------- ------------- --------------
       7839            10           5000
       7788            20           3000
       7698            30           2850
```

HANDS ON

rank each host based on the number of beds they have listed on our website. The host with the most beds should be ranked first (rank = 1)



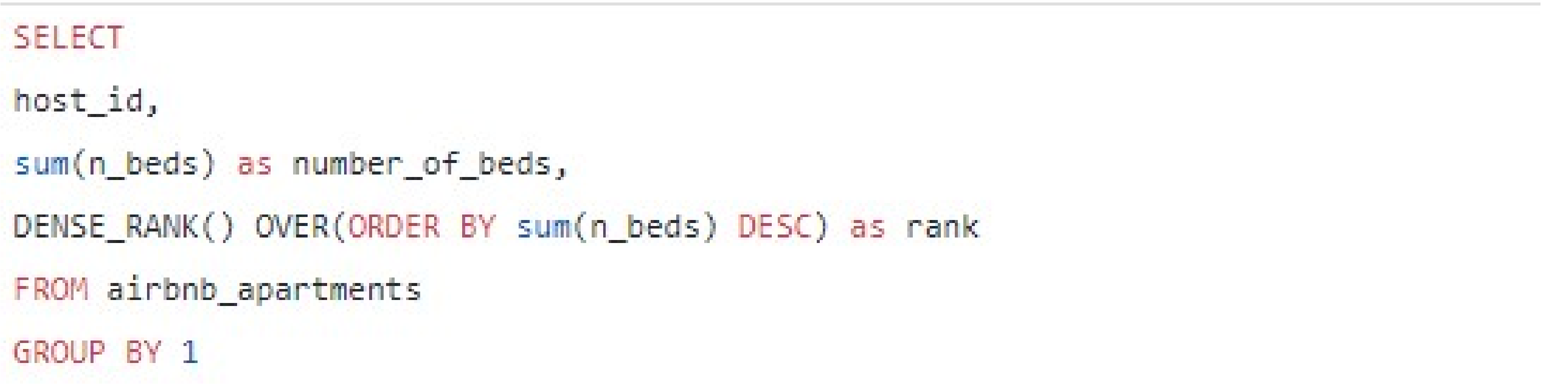
host_id	apartment_id	apartment_type	n_beds	n_bedrooms	city
0	A1	Room	1	1	New York
0	A2	Room	1	1	New Jersey
0	A3	Room	1	1	New Jersey
1	A4	Apartment	2	1	Houston
1	A5	Apartment	2	1	Las Vegas
2	A6	Yurt	3	1	-
3	A7	Penthouse	3	3	Tianjin
3	A8	Penthouse	5	5	Beijing



host_id	number_of_beds	rank
10	16	1
3	8	2
6	6	3
5	5	4
7	4	5
1	4	5
9	4	5
0	3	6
2	3	6
8	2	7
4	2	7
11	2	7

HANDS ON

rank each host based on the number of beds they have listed on our website. The host with the most beds should be ranked first (rank = 1)



```sql
SELECT
host_id,
sum(n_beds) as number_of_beds,
DENSE_RANK() OVER(ORDER BY sum(n_beds) DESC) as rank
FROM airbnb_apartments
GROUP BY 1
```

FIRST_VALUE and LAST_VALUE

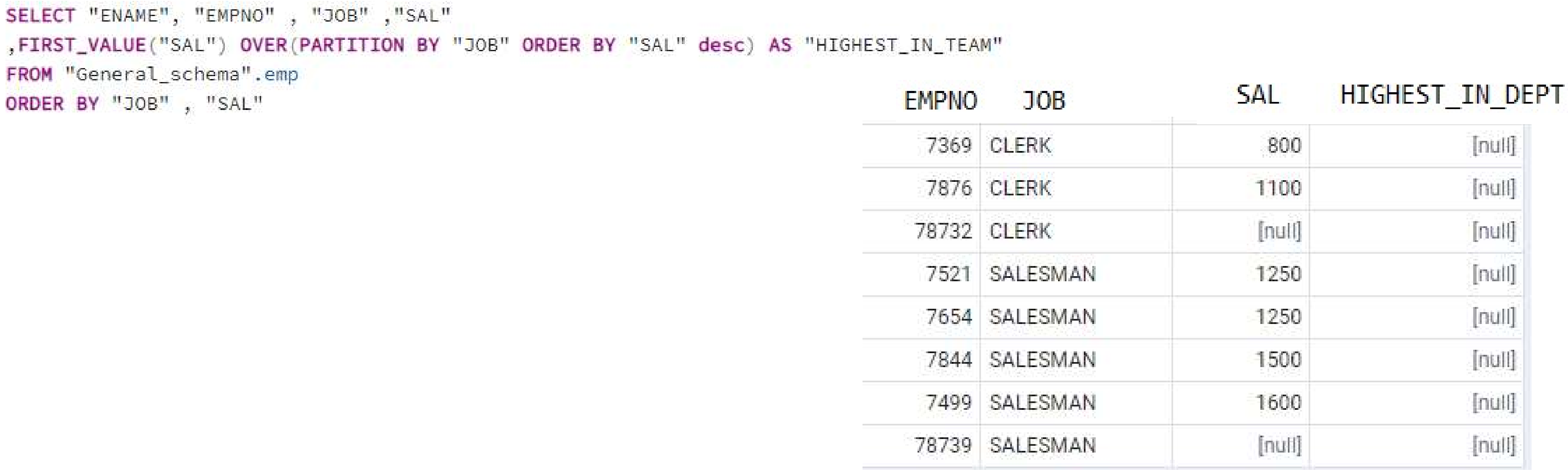
- FIRST_VALUE / LAST_VALUE { (expr [{RESPECT | IGNORE} NULLS]) } OVER (analytic_clause)
- Allows you to return the first / last result from an ordered set.
- The "{RESPECT | IGNORE} NULLS" clause indicates if NULLs are considered when determining results.

```
SELECT empno, deptno, sal,
       FIRST_VALUE( sal IGNORE NULLS )
         OVER (PARTITION BY deptno ORDER BY sal) AS lowest_in_dept
FROM   emp
ORDER BY deptno, sal;
```

```
     EMPNO     DEPTNO        SAL LOWEST_IN_DEPT
---------- ---------- ---------- --------------
      7934         10       1300           1300
      7782         10       2450           1300
      7839         10       5000           1300
      7369         20        800            800
      7876         20       1100            800
      7566         20       2975            800
      7788         20       3000            800
      7902         20       3000            800
      7900         30        950            950
      7654         30       1250            950
      7521         30       1250            950
```

FIRST_VALUE and LAST_VALUE (IGNORE NULLS)

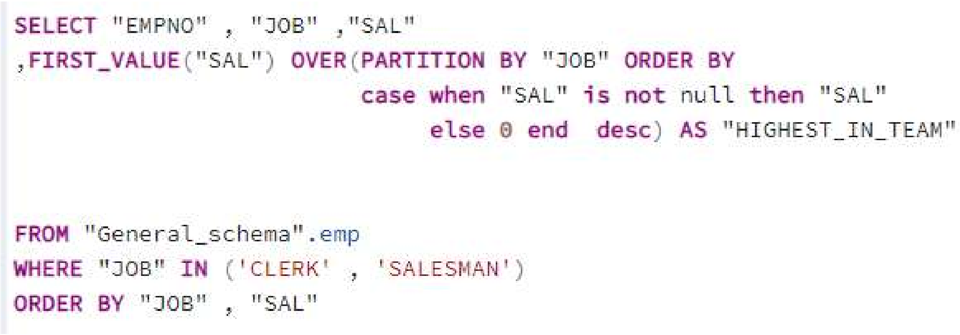
- Though **LAST_VALUE** and **FIRST_VALUE** are quite handy on many occasions, one of the main limitations for the functions have been the absence of **IGNORE_NULLS** support, as found in many other RDBMS.



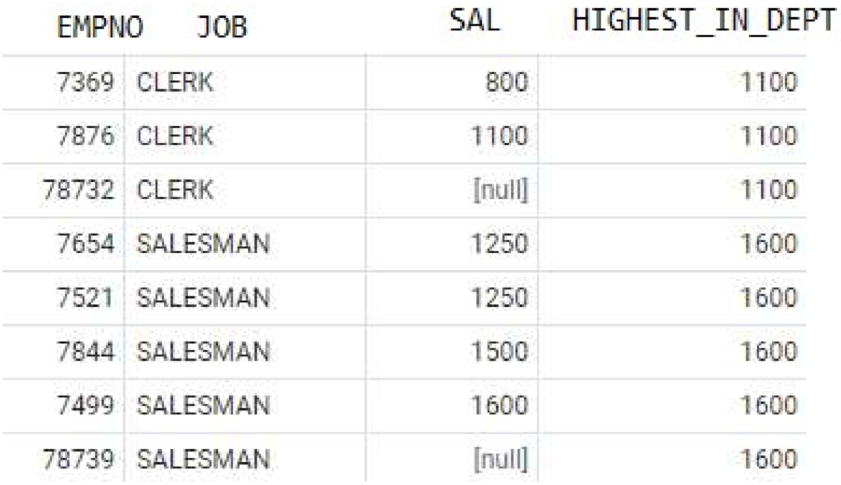
```
SELECT "ENAME", "EMPNO" , "JOB" ,"SAL"
,FIRST_VALUE("SAL") OVER(PARTITION BY "JOB" ORDER BY "SAL" desc) AS "HIGHEST_IN_TEAM"
FROM "General_schema".emp
ORDER BY "JOB" , "SAL"
```

EMPNO	JOB	SAL	HIGHEST_IN_DEPT
7369	CLERK	800	[null]
7876	CLERK	1100	[null]
78732	CLERK	[null]	[null]
7521	SALESMAN	1250	[null]
7654	SALESMAN	1250	[null]
7844	SALESMAN	1500	[null]
7499	SALESMAN	1600	[null]
78739	SALESMAN	[null]	[null]

FIRST_VALUE and LAST_VALUE (IGNORE NULLS)



```
SELECT "EMPNO" , "JOB" ,"SAL"
,FIRST_VALUE("SAL") OVER(PARTITION BY "JOB" ORDER BY
                         case when "SAL" is not null then "SAL"
                              else 0 end  desc) AS "HIGHEST_IN_TEAM"


FROM "General_schema".emp
WHERE "JOB" IN ('CLERK' , 'SALESMAN')
ORDER BY "JOB" , "SAL"
```



EMPNO	JOB	SAL	HIGHEST_IN_DEPT
7369	CLERK	800	1100
7876	CLERK	1100	1100
78732	CLERK	[null]	1100
7654	SALESMAN	1250	1600
7521	SALESMAN	1250	1600
7844	SALESMAN	1500	1600
7499	SALESMAN	1600	1600
78739	SALESMAN	[null]	1600

Date of next order(LAB)

- From orders table in the order entry schema (OE), Find the date of next order for each customer.
- There are almost 5 different ways to calculate the date of the next order, get as much as you can!

```sql
SELECT customer_id
, TRUNC ( order_date ) AS order_date
, order_total
, LEAD ( TRUNC(order_date) ) OVER
        ( PARTITION BY customer_id ORDER BY order_date ) AS next_order_date_LEAD
, LAG ( TRUNC(order_date) ) OVER
        ( PARTITION BY customer_id ORDER BY order_date DESC )  AS next_order_date_LAG
, MAX ( TRUNC(order_date) ) OVER
        ( PARTITION BY customer_id ORDER BY order_date
            ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING )  AS next_order_date_MAX
, MIN ( TRUNC(order_date) ) OVER
        ( PARTITION BY customer_id ORDER BY order_date
            ROWS BETWEEN 1 FOLLOWING AND 1 FOLLOWING )  AS next_order_date_MIN
, MIN ( TRUNC(order_date) ) OVER
        ( PARTITION BY customer_id ORDER BY order_date
            ROWS BETWEEN 1 FOLLOWING AND UNBOUNDED FOLLOWING )  AS next_order_date_MIN2
FROM orders
ORDER BY 1, 2;
```