oracle分析函数技术详解（配上开窗函数over()） - CSDN博客

一、Oracle分析函数入门

分析函数是什么？

分析函数是Oracle专门用于解决复杂报表统计需求的功能强大的函数，它可以在数据中进行分组然后计算基于组的某种统计值，并且每一组的每一行都可以返回一个统计值。

分析函数和聚合函数的不同之处是什么？

普通的聚合函数用group by分组，每个分组返回一个统计值，而分析函数采用partition by分组，并且每组每行都可以返回一个统计值。

分析函数的形式

分析函数带有一个**开窗函数over()，包含三个分析子句:分组(partition by)，排序(order by)，窗口(rows)**，他们的使用形式如下：over(partition by xxx order by yyy rows between zzz)。

注：窗口子句在这里我只说rows方式的窗口,range方式和滑动窗口也不提

分析函数例子(在scott用户下模拟)

示例目的：显示各部门员工的工资，并附带显示该部分的最高工资。

```
--显示各部门员工的工资，并附带显示该部分的最高工资。SELECT E.DEPTNO,        E.EMPNO,        E.ENAME,
E.SAL,        LAST_VALUE(E.SAL)        OVER(PARTITION BY E.DEPTNO                ORDERBY E.SAL ROWS
--unbounded preceding and unbouned following针对当前所有记录的前一条、后一条记录，也就是表中的所有记录--
unbounded：不受控制的，无限的--preceding：在...之前--following：在...之后BETWEEN UNBOUNDED PRECEDING AND
UNBOUNDED FOLLOWING) MAX_SAL  FROM EMP E;
```

运行结果：

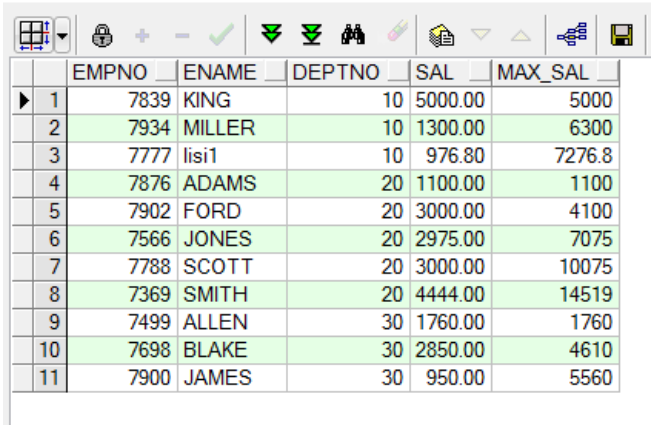
	DEPTNO	EMPNO	ENAME	SAL	MAX_SAL
1	10	7777	lisi1	976.80	5000
2	10	7934	MILLER	1300.00	5000
3	10	7839	KING	5000.00	5000
4	20	7876	ADAMS	1100.00	4444
5	20	7566	JONES	2975.00	4444
6	20	7788	SCOTT	3000.00	4444
7	20	7902	FORD	3000.00	4444
8	20	7369	SMITH	4444.00	4444
9	30	7900	JAMES	950.00	2850
10	30	7521	WARD	1250.00	2850
11	30	7654	MARTIN	1250.00	2850
12	30	7844	TURNER	1500.00	2850
13	30	7499	ALLEN	1760.00	2850
14	30	7698	BLAKE	2850.00	2850

示例目的：按照deptno分组，然后计算每组值的总和

```
SELECT EMPNO,        ENAME,        DEPTNO,        SAL,        SUM(SAL) OVER(PARTITION BY DEPTNO ORDERBY
```

ENAME) max_sal FROM SCOTT.EMP;

运行结果：

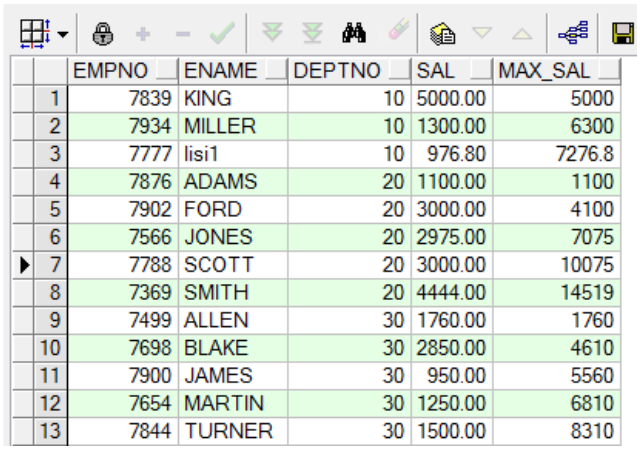


	EMPNO	ENAME	DEPTNO	SAL	MAX_SAL
1	7839	KING	10	5000.00	5000
2	7934	MILLER	10	1300.00	6300
3	7777	lisi1	10	976.80	7276.8
4	7876	ADAMS	20	1100.00	1100
5	7902	FORD	20	3000.00	4100
6	7566	JONES	20	2975.00	7075
7	7788	SCOTT	20	3000.00	10075
8	7369	SMITH	20	4444.00	14519
9	7499	ALLEN	30	1760.00	1760
10	7698	BLAKE	30	2850.00	4610
11	7900	JAMES	30	950.00	5560

示例目的：对各部门进行分组，并附带显示第一行至当前行的汇总

SELECT EMPNO, ENAME, DEPTNO, SAL, --注意ROWS BETWEEN unbounded preceding AND current row 是指第一行至当前行的汇总SUM(SAL) OVER(PARTITION BY DEPTNO ORDERBY ENAME ROWS BETWEEN UNBOUNDED PRECEDING ANDCURRENT ROW) max_sal FROM SCOTT.EMP;

运行结果：

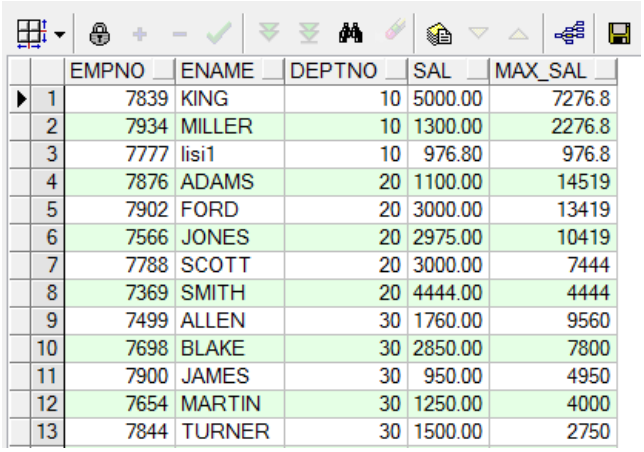


	EMPNO	ENAME	DEPTNO	SAL	MAX_SAL
1	7839	KING	10	5000.00	5000
2	7934	MILLER	10	1300.00	6300
3	7777	lisi1	10	976.80	7276.8
4	7876	ADAMS	20	1100.00	1100
5	7902	FORD	20	3000.00	4100
6	7566	JONES	20	2975.00	7075
7	7788	SCOTT	20	3000.00	10075
8	7369	SMITH	20	4444.00	14519
9	7499	ALLEN	30	1760.00	1760
10	7698	BLAKE	30	2850.00	4610
11	7900	JAMES	30	950.00	5560
12	7654	MARTIN	30	1250.00	6810
13	7844	TURNER	30	1500.00	8310

示例目标：当前行至最后一行的汇总

SELECT EMPNO, ENAME, DEPTNO, SAL, --注意ROWS BETWEEN current row AND unbounded following 指当前行到最后一行的汇总SUM(SAL) OVER(PARTITION BY DEPTNO ORDERBY ENAME ROWS BETWEENCURRENT ROW AND UNBOUNDED FOLLOWING) max_sal FROM SCOTT.EMP;

运行结果：

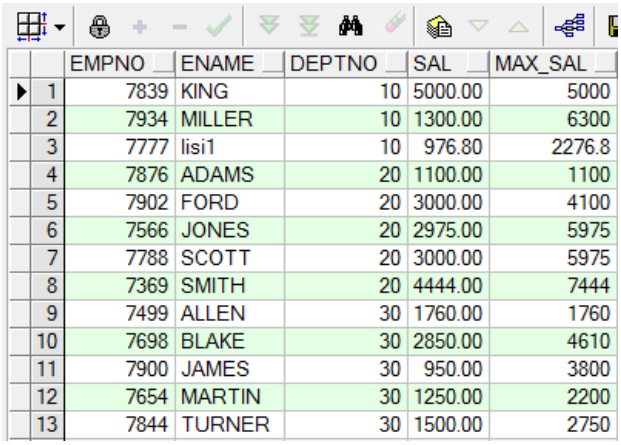


	EMPNO	ENAME	DEPTNO	SAL	MAX_SAL
1	7839	KING	10	5000.00	7276.8
2	7934	MILLER	10	1300.00	2276.8
3	7777	lisi1	10	976.80	976.8
4	7876	ADAMS	20	1100.00	14519
5	7902	FORD	20	3000.00	13419
6	7566	JONES	20	2975.00	10419
7	7788	SCOTT	20	3000.00	7444
8	7369	SMITH	20	4444.00	4444
9	7499	ALLEN	30	1760.00	9560
10	7698	BLAKE	30	2850.00	7800
11	7900	JAMES	30	950.00	4950
12	7654	MARTIN	30	1250.00	4000
13	7844	TURNER	30	1500.00	2750

示例目标：当前行的上一行(rownum-1)到当前行的汇总

```
SELECT EMPNO,        ENAME,        DEPTNO,        SAL,        --注意ROWS BETWEEN 1 preceding AND current row 是指当前行的上一行(rownum-1)到当前行的汇总 SUM(SAL) OVER(PARTITION BY DEPTNO                    ORDERBY ENAME ROWS                    BETWEEN1 PRECEDING ANDCURRENT ROW) max_sal  FROM SCOTT.EMP;
```

运行结果：

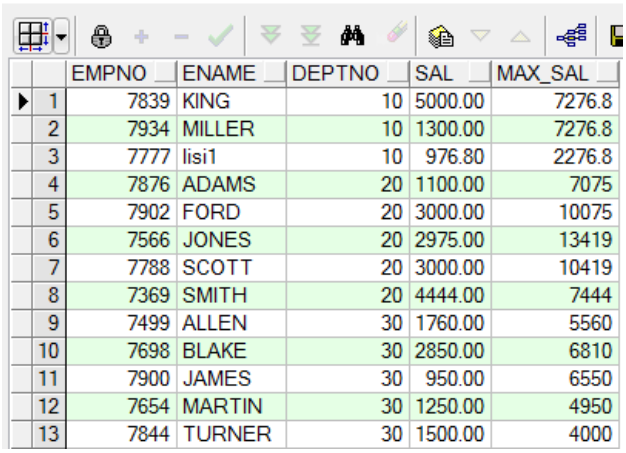


	EMPNO	ENAME	DEPTNO	SAL	MAX_SAL
1	7839	KING	10	5000.00	5000
2	7934	MILLER	10	1300.00	6300
3	7777	lisi1	10	976.80	2276.8
4	7876	ADAMS	20	1100.00	1100
5	7902	FORD	20	3000.00	4100
6	7566	JONES	20	2975.00	5975
7	7788	SCOTT	20	3000.00	5975
8	7369	SMITH	20	4444.00	7444
9	7499	ALLEN	30	1760.00	1760
10	7698	BLAKE	30	2850.00	4610
11	7900	JAMES	30	950.00	3800
12	7654	MARTIN	30	1250.00	2200
13	7844	TURNER	30	1500.00	2750

示例目标：　　当前行的上一行(rownum-1)到当前行的下辆行(rownum+2)的汇总

```
SELECT EMPNO,        ENAME,        DEPTNO,        SAL,        --注意ROWS BETWEEN 1 preceding AND 1 following 是指当前行的上一行(rownum-1)到当前行的下辆行(rownum+2)的汇总SUM(SAL) OVER(PARTITION BY DEPTNO ORDERBY ENAME                    ROWS BETWEEN1 PRECEDING AND2 FOLLOWING) max_sal  FROM SCOTT.EMP;
```

运行结果：



	EMPNO	ENAME	DEPTNO	SAL	MAX_SAL
1	7839	KING	10	5000.00	7276.8
2	7934	MILLER	10	1300.00	7276.8
3	7777	lisi1	10	976.80	2276.8
4	7876	ADAMS	20	1100.00	7075
5	7902	FORD	20	3000.00	10075
6	7566	JONES	20	2975.00	13419
7	7788	SCOTT	20	3000.00	10419
8	7369	SMITH	20	4444.00	7444
9	7499	ALLEN	30	1760.00	5560
10	7698	BLAKE	30	2850.00	6810
11	7900	JAMES	30	950.00	6550
12	7654	MARTIN	30	1250.00	4950
13	7844	TURNER	30	1500.00	4000

二、理解over()函数

1.1、两个order by的执行时机

分析函数（以及与其配合的开窗函数over（））是在整个sql查询结束后(sql语句中的order by的执行比较特殊)再进行的操作，也就是说sql语句中的order by也会影响分析函数的执行结果：

a）两者一致：如果sql语句中的order by满足与分析函数配合的开窗函数over（）分析时要求的排序，即sql语句中的order by子句里的内容和开窗函数over（）中的order by子句里的内容一样，

那么sql语句中的排序将先执行,分析函数在分析时就不必再排序；

b）两者不一致：如果sql语句中的order by不满足与分析函数配合的开窗函数over（）分析时要求的排序，即sql语句中的order by子句里的内容和开窗函数over（）中的order by子句里的内容不一样，

那么sql语句中的排序将最后在分析函数分析结束后执行排序。

1.2、开窗函数over（）分析函数中的分组/排序/窗口

开窗函数over（）**分析函数**包含三个分析子句：分组子句(partition by)， 排序子句(order by)， 窗口子句(rows)

窗口就是分析函数分析时要处理的数据范围，就拿sum来说,它是sum窗口中的记录而不是整个分组中的记录，因此我们在想得到某个栏位的累计值时，我们需要把窗口指定到该分组中的第一行数据到当前行，如果你指定该窗口从该分组中的第一行到最后一行,那么该组中的每一个sum值都会一样，即整个组的总和。

窗口子句在这里我只说rows方式的窗口，range方式和滑动窗口也不提。

窗口子句中我们经常用到指定第一行，当前行，最后一行这样的三个属性：

第一行是 unbounded preceding，

当前行是 current row，

最后一行是 unbounded following，

注释：

当开窗函数over（）出现分组(partition by)子句时，

unbounded preceding即第一行是指表中一个分组里的第一行， unbounded following即最后一行是指表中一个分组里的最后一行；

当开窗函数over（）省略了分组(partition by)子句时，

unbounded preceding即第一行是指表中的第一行， unbounded following即最后一行是指表中的最后一行。

窗口子句不能单独出现，必须有order by子句时才能出现，

例如：

last_value(sal) over(partition by deptno orderby sal rows between unbounded preceding and unbounded following)

以上示例指定窗口为整个分组。而出现order by子句的时候，不一定要有窗口子句，但效果会很不一样，此时的窗口默认是当前组的第一行到当前行！

如果省略分组,则把全部记录当成一个组。

a) 如果存在order by则默认窗口是unbounded preceding and current row --当前组的第一行到当前行

b) 如果这时省略order by则窗口默认为unbounded preceding and unbounded following --整个组

而无论是否省略分组子句，如下结论都是成立的：

1、窗口子句不能单独出现，必须有order by子句时才能出现。

2、当省略窗口子句时：

a) 如果存在order by则默认的窗口是unbounded preceding and current row --当前组的第一行到当前行，即在当前组中，第一行到当前行

b) 如果同时省略order by则默认的窗口是unbounded preceding and unbounded following --整个组

所以，

lag(sal) over(order by sal) 解释

over（order by salary）表示意义如下：

首先，我们要知道由于省略分组子句，所以当前组的范围为整个表的数据行，

然后，在当前组（此时为整个表的数据行）这个范围里执行排序（即order by salary），

最后，我们知道分析函数lag(sal)在当前组（此时为整个表的数据行）这个范围里的窗口范围为当前组的第一行到当前行，即分析函数lag(sal)在这个窗口范围执行。

参见：

Oracle的LAG和LEAD分析函数

Oracle分析函数

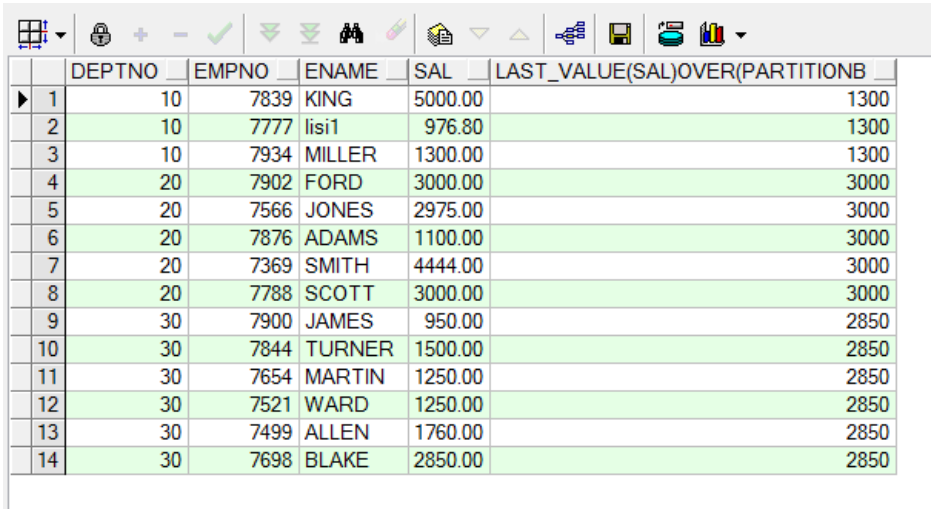
ROW_NUMBER()|RANK()|LAG()使用详解

1.3、帮助理解over()的实例

例1：关注点：sql无排序，over()排序子句省略

```
SELECT DEPTNO, EMPNO, ENAME, SAL,        LAST_VALUE(SAL) OVER(PARTITION BY DEPTNO)FROM EMP;
```

运行结果：

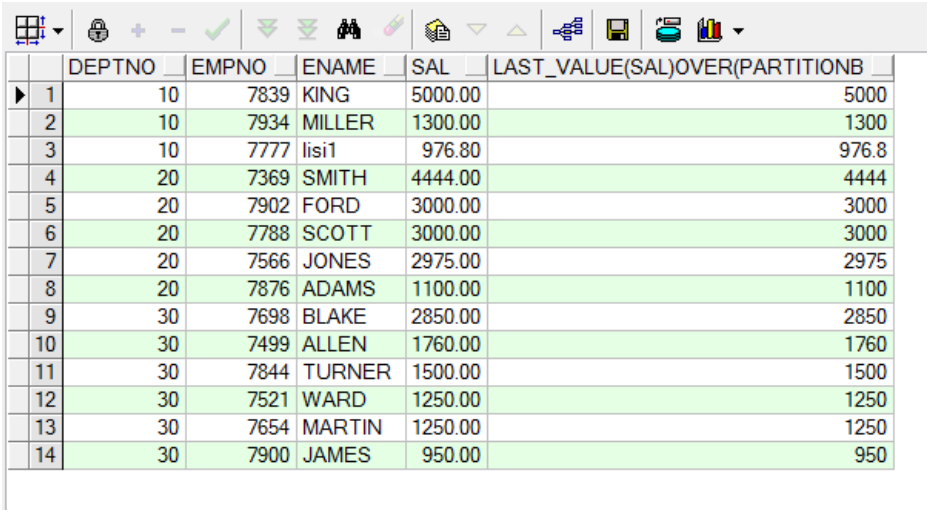


	DEPTNO	EMPNO	ENAME	SAL	LAST_VALUE(SAL)OVER(PARTITIONB
1	10	7839	KING	5000.00	1300
2	10	7777	lisi1	976.80	1300
3	10	7934	MILLER	1300.00	1300
4	20	7902	FORD	3000.00	3000
5	20	7566	JONES	2975.00	3000
6	20	7876	ADAMS	1100.00	3000
7	20	7369	SMITH	4444.00	3000
8	20	7788	SCOTT	3000.00	3000
9	30	7900	JAMES	950.00	2850
10	30	7844	TURNER	1500.00	2850
11	30	7654	MARTIN	1250.00	2850
12	30	7521	WARD	1250.00	2850
13	30	7499	ALLEN	1760.00	2850
14	30	7698	BLAKE	2850.00	2850

例2：关注点：sql无排序，over()排序子句有，窗口省略

```
SELECT DEPTNO,        EMPNO,        ENAME,        SAL,        LAST_VALUE(SAL) OVER(PARTITION BY DEPTNO
ORDERBY SAL DESC)  FROM EMP;
```

运行结果：

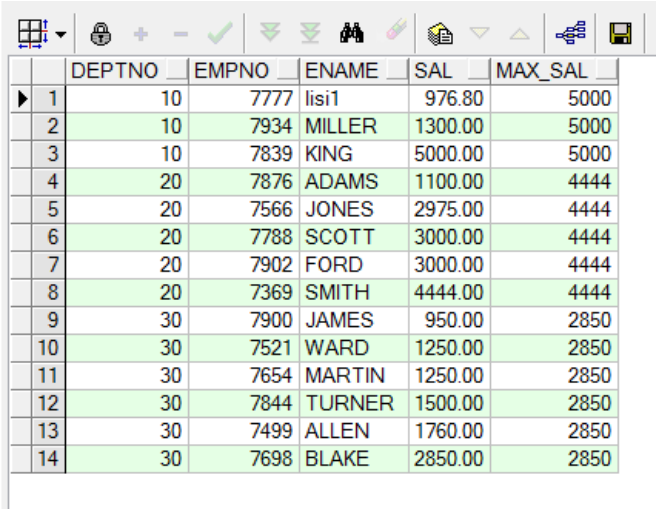


	DEPTNO	EMPNO	ENAME	SAL	LAST_VALUE(SAL)OVER(PARTITIONB
1	10	7839	KING	5000.00	5000
2	10	7934	MILLER	1300.00	1300
3	10	7777	lisi1	976.80	976.8
4	20	7369	SMITH	4444.00	4444
5	20	7902	FORD	3000.00	3000
6	20	7788	SCOTT	3000.00	3000
7	20	7566	JONES	2975.00	2975
8	20	7876	ADAMS	1100.00	1100
9	30	7698	BLAKE	2850.00	2850
10	30	7499	ALLEN	1760.00	1760
11	30	7844	TURNER	1500.00	1500
12	30	7521	WARD	1250.00	1250
13	30	7654	MARTIN	1250.00	1250
14	30	7900	JAMES	950.00	950

例3：关注点：sql无排序，over()排序子句有，窗口也有，窗口特意强调全组数据

```
SELECT DEPTNO,        EMPNO,        ENAME,        SAL,        LAST_VALUE(SAL)        OVER(PARTITION BY DEPTNO
ORDERBY SAL              ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) MAX_SAL  FROM EMP;
```

运行结果：

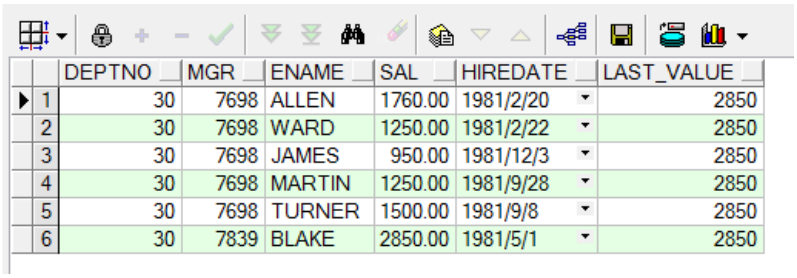


	DEPTNO	EMPNO	ENAME	SAL	MAX_SAL
1	10	7777	lisi1	976.80	5000
2	10	7934	MILLER	1300.00	5000
3	10	7839	KING	5000.00	5000
4	20	7876	ADAMS	1100.00	4444
5	20	7566	JONES	2975.00	4444
6	20	7788	SCOTT	3000.00	4444
7	20	7902	FORD	3000.00	4444
8	20	7369	SMITH	4444.00	4444
9	30	7900	JAMES	950.00	2850
10	30	7521	WARD	1250.00	2850
11	30	7654	MARTIN	1250.00	2850
12	30	7844	TURNER	1500.00	2850
13	30	7499	ALLEN	1760.00	2850
14	30	7698	BLAKE	2850.00	2850

例4：关注点：sql有排序（正序），over()排序子句无，先做sql排序再进行分析函数运算

```
SELECT DEPTNO,        MGR,        ENAME,        SAL,        HIREDATE,        LAST_VALUE(SAL) OVER(PARTITION BY DEPTNO) LAST_VALUE  FROM EMP WHERE DEPTNO =30ORDERBY DEPTNO, MGR;
```

运行结果：

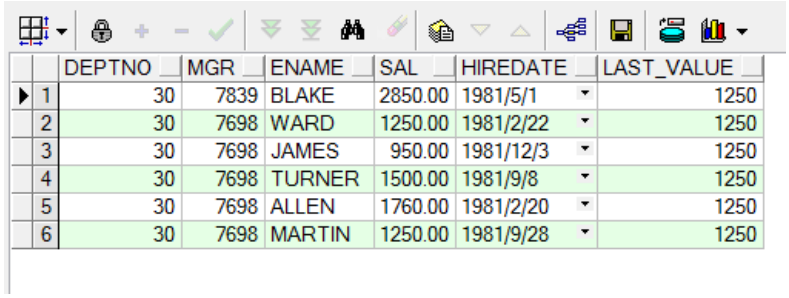


	DEPTNO	MGR	ENAME	SAL	HIREDATE	LAST_VALUE
1	30	7698	ALLEN	1760.00	1981/2/20	2850
2	30	7698	WARD	1250.00	1981/2/22	2850
3	30	7698	JAMES	950.00	1981/12/3	2850
4	30	7698	MARTIN	1250.00	1981/9/28	2850
5	30	7698	TURNER	1500.00	1981/9/8	2850
6	30	7839	BLAKE	2850.00	1981/5/1	2850

例5：关注点：sql有排序（倒序），over()排序子句无，先做sql排序再进行分析函数运算

```
SELECT DEPTNO,        MGR,        ENAME,        SAL,        HIREDATE,        LAST_VALUE(SAL) OVER(PARTITION BY DEPTNO) LAST_VALUE  FROM EMP WHERE DEPTNO =30ORDERBY DEPTNO, MGR DESC;
```

运行结果：



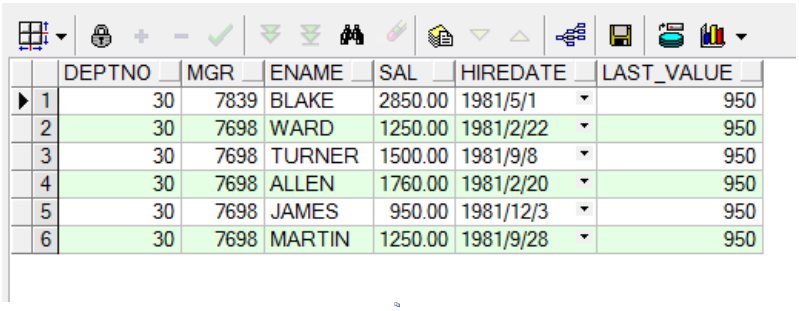
	DEPTNO	MGR	ENAME	SAL	HIREDATE	LAST_VALUE
1	30	7839	BLAKE	2850.00	1981/5/1	1250
2	30	7698	WARD	1250.00	1981/2/22	1250
3	30	7698	JAMES	950.00	1981/12/3	1250
4	30	7698	TURNER	1500.00	1981/9/8	1250
5	30	7698	ALLEN	1760.00	1981/2/20	1250
6	30	7698	MARTIN	1250.00	1981/9/28	1250

例6：关注点：sql有排序（倒序），over()排序子句有，窗口子句无，此时的运算是：sql先选数据但是不排序，而后排序子句先排序并进行分析函数处理（窗口默认为第一行到当前行），最后再进行sql排序

```
SELECT DEPTNO,        MGR,        ENAME,        SAL,        HIREDATE,        MIN(SAL) OVER(PARTITION BY DEPTNO
```

```
ORDERBY SAL ASC) LAST_VALUE  FROM EMP WHERE DEPTNO =30ORDERBY DEPTNO, MGR DESC;
```

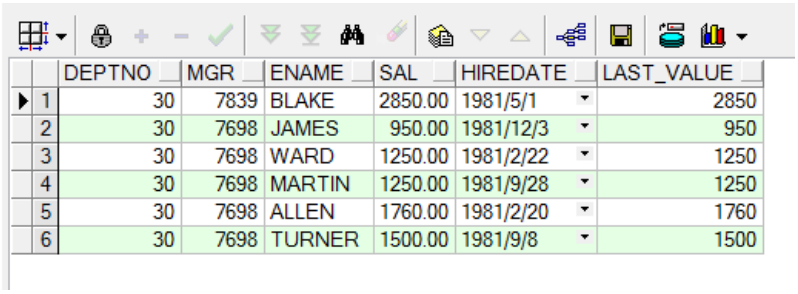
运行结果：



	DEPTNO	MGR	ENAME	SAL	HIREDATE	LAST_VALUE
1	30	7839	BLAKE	2850.00	1981/5/1	950
2	30	7698	WARD	1250.00	1981/2/22	950
3	30	7698	TURNER	1500.00	1981/9/8	950
4	30	7698	ALLEN	1760.00	1981/2/20	950
5	30	7698	JAMES	950.00	1981/12/3	950
6	30	7698	MARTIN	1250.00	1981/9/28	950

```
SELECT DEPTNO,        MGR,        ENAME,        SAL,        HIREDATE,        MIN(SAL) OVER(PARTITION BY DEPTNO
ORDERBY SAL DESC) LAST_VALUE  FROM EMP WHERE DEPTNO =30ORDERBY DEPTNO, MGR DESC;
```

运行结果：



	DEPTNO	MGR	ENAME	SAL	HIREDATE	LAST_VALUE
1	30	7839	BLAKE	2850.00	1981/5/1	2850
2	30	7698	JAMES	950.00	1981/12/3	950
3	30	7698	WARD	1250.00	1981/2/22	1250
4	30	7698	MARTIN	1250.00	1981/9/28	1250
5	30	7698	ALLEN	1760.00	1981/2/20	1760
6	30	7698	TURNER	1500.00	1981/9/8	1500

三、常见分析函数详解

为了方便进行实践，特将演示表和数据罗列如下：

一、创建表

```
createtable t(    bill_month varchar2(12) ,    area_code number,    net_type varchar(2),    local_fare
number );
```

二、插入数据

```
insertinto t values('200405',5761,'G', 7393344.04); insertinto t values('200405',5761,'J',
5667089.85); insertinto t values('200405',5762,'G', 6315075.96); insertinto t
values('200405',5762,'J', 6328716.15); insertinto t values('200405',5763,'G', 8861742.59); insertinto
t values('200405',5763,'J', 7788036.32); insertinto t values('200405',5764,'G', 6028670.45);
insertinto t values('200405',5764,'J', 6459121.49); insertinto t values('200405',5765,'G',
13156065.77); insertinto t values('200405',5765,'J', 11901671.70); insertinto t
values('200406',5761,'G', 7614587.96); insertinto t values('200406',5761,'J', 5704343.05); insertinto
t values('200406',5762,'G', 6556992.60); insertinto t values('200406',5762,'J', 6238068.05);
insertinto t values('200406',5763,'G', 9130055.46); insertinto t values('200406',5763,'J',
7990460.25); insertinto t values('200406',5764,'G', 6387706.01); insertinto t
values('200406',5764,'J', 6907481.66); insertinto t values('200406',5765,'G', 13562968.81); insertinto
t values('200406',5765,'J', 12495492.50); insertinto t values('200407',5761,'G', 7987050.65);
insertinto t values('200407',5761,'J', 5723215.28); insertinto t values('200407',5762,'G',
6833096.68); insertinto t values('200407',5762,'J', 6391201.44); insertinto t
values('200407',5763,'G', 9410815.91); insertinto t values('200407',5763,'J', 8076677.41); insertinto
t values('200407',5764,'G', 6456433.23); insertinto t values('200407',5764,'J', 6987660.53);
insertinto t values('200407',5765,'G', 14000101.20); insertinto t values('200407',5765,'J',
12301780.20); insertinto t values('200408',5761,'G', 8085170.84); insertinto t
```

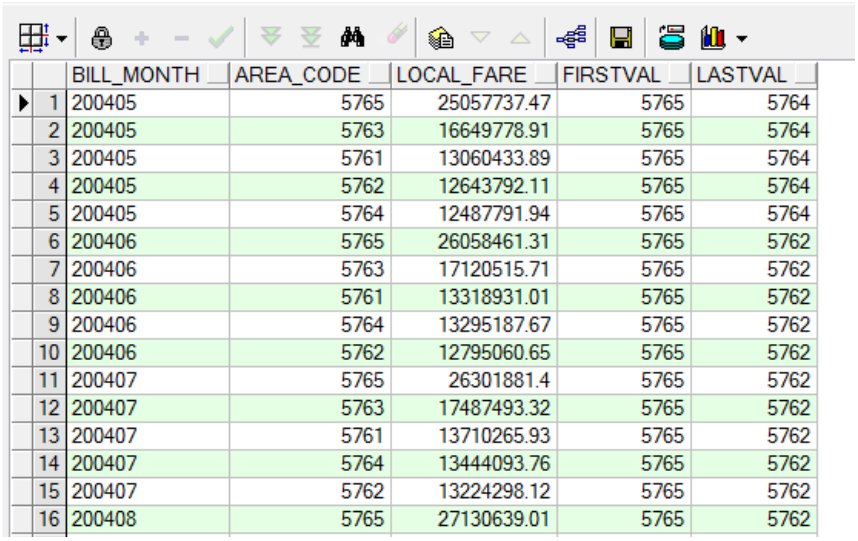
```
values('200408',5761,'J', 6050611.37); insertinto t values('200408',5762,'G', 6854584.22); insertinto t values('200408',5762,'J', 6521884.50); insertinto t values('200408',5763,'G', 9468707.65); insertinto t values('200408',5763,'J', 8460049.43); insertinto t values('200408',5764,'G', 6587559.23); insertinto t values('200408',5764,'J', 7342135.86); insertinto t values('200408',5765,'G', 14450586.63); insertinto t values('200408',5765,'J', 12680052.38); commit;
```

三、first_value()与last_value()：求最值对应的其他属性

问题、取出每月通话费最高和最低的两个地区。

```
SELECT BILL_MONTH, AREA_CODE, SUM(LOCAL_FARE) LOCAL_FARE, FIRST_VALUE(AREA_CODE) OVER(PARTITION BY BILL_MONTH ORDERBYSUM(LOCAL_FARE) DESC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) FIRSTVAL, LAST_VALUE(AREA_CODE) OVER(PARTITION BY BILL_MONTH ORDERBYSUM(LOCAL_FARE) DESC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) LASTVAL FROM T GROUPBY BILL_MONTH, AREA_CODE ORDERBY BILL_MONTH
```

运行结果：



	BILL_MONTH	AREA_CODE	LOCAL_FARE	FIRSTVAL	LASTVAL
1	200405	5765	25057737.47	5765	5764
2	200405	5763	16649778.91	5765	5764
3	200405	5761	13060433.89	5765	5764
4	200405	5762	12643792.11	5765	5764
5	200405	5764	12487791.94	5765	5764
6	200406	5765	26058461.31	5765	5762
7	200406	5763	17120515.71	5765	5762
8	200406	5761	13318931.01	5765	5762
9	200406	5764	13295187.67	5765	5762
10	200406	5762	12795060.65	5765	5762
11	200407	5765	26301881.4	5765	5762
12	200407	5763	17487493.32	5765	5762
13	200407	5761	13710265.93	5765	5762
14	200407	5764	13444093.76	5765	5762
15	200407	5762	13224298.12	5765	5762
16	200408	5765	27130639.01	5765	5762

四、rank(),dense_rank()与row_number()：求排序

rank，dense_rank，row_number函数为每条记录产生一个从1开始至n的自然数，n的值可能小于等于记录的总数。这3个函数的唯一区别在于当碰到相同数据时的排名策略。

①row_number：

row_number函数返回一个唯一的值，当碰到相同数据时，排名按照记录集中记录的顺序依次递增。

②dense_rank：

dense_rank函数返回一个唯一的值，当碰到相同数据时，此时所有相同数据的排名都是一样的。

③rank：

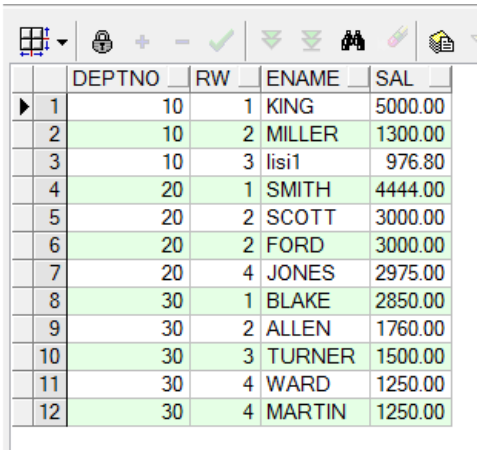
rank函数返回一个唯一的值，当碰到相同的数据时，此时所有相同数据的排名是一样的，同时会在最后一条相同记录和下一条不同记录的排名之间空出排名。

演示数据在Oracle自带的scott用户下：

1、rank()值相同时排名相同，其后排名跳跃不连续

```
SELECT*FROM (SELECT DEPTNO,                RANK() OVER(PARTITION BY DEPTNO ORDERBY SAL DESC) RW,
ENAME,                SAL          FROM SCOTT.EMP)  WHERE RW <=4;
```

运行结果：

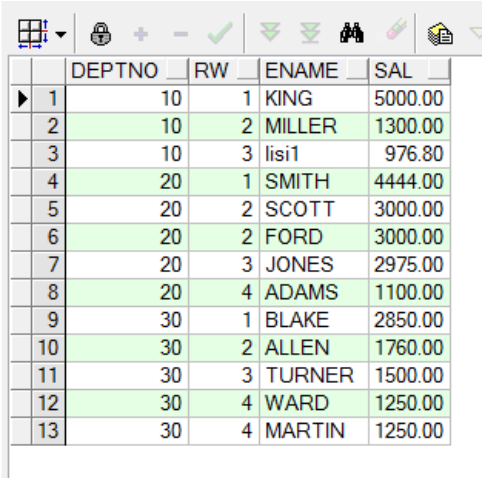


	DEPTNO	RW	ENAME	SAL
1	10	1	KING	5000.00
2	10	2	MILLER	1300.00
3	10	3	lisi1	976.80
4	20	1	SMITH	4444.00
5	20	2	SCOTT	3000.00
6	20	2	FORD	3000.00
7	20	4	JONES	2975.00
8	30	1	BLAKE	2850.00
9	30	2	ALLEN	1760.00
10	30	3	TURNER	1500.00
11	30	4	WARD	1250.00
12	30	4	MARTIN	1250.00

2、dense_rank()值相同时排名相同，其后排名连续不跳跃

```
SELECT*FROM (SELECT DEPTNO,                DENSE_RANK() OVER(PARTITION BY DEPTNO ORDERBY SAL DESC) RW,
ENAME,                SAL          FROM SCOTT.EMP)  WHERE RW <=4;
```

运行结果：

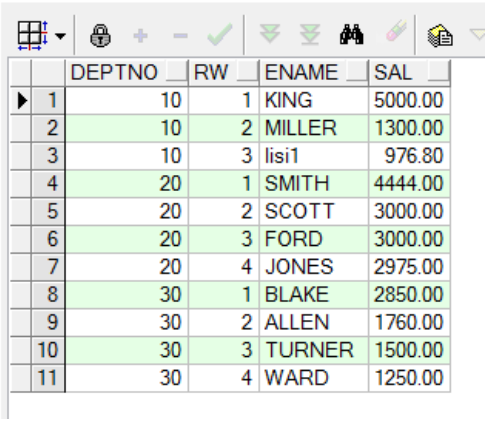


	DEPTNO	RW	ENAME	SAL
1	10	1	KING	5000.00
2	10	2	MILLER	1300.00
3	10	3	lisi1	976.80
4	20	1	SMITH	4444.00
5	20	2	SCOTT	3000.00
6	20	2	FORD	3000.00
7	20	3	JONES	2975.00
8	20	4	ADAMS	1100.00
9	30	1	BLAKE	2850.00
10	30	2	ALLEN	1760.00
11	30	3	TURNER	1500.00
12	30	4	WARD	1250.00
13	30	4	MARTIN	1250.00

3、row_number()值相同时排名不相等，其后排名连续不跳跃

```
SELECT*FROM (SELECT DEPTNO,                ROW_NUMBER() OVER(PARTITION BY DEPTNO ORDERBY SAL DESC) RW,
ENAME,                SAL          FROM SCOTT.EMP)  WHERE RW <=4;
```

运行结果：



	DEPTNO	RW	ENAME	SAL
1	10	1	KING	5000.00
2	10	2	MILLER	1300.00
3	10	3	lisi1	976.80
4	20	1	SMITH	4444.00
5	20	2	SCOTT	3000.00
6	20	3	FORD	3000.00
7	20	4	JONES	2975.00
8	30	1	BLAKE	2850.00
9	30	2	ALLEN	1760.00
10	30	3	TURNER	1500.00
11	30	4	WARD	1250.00

五、lag()与lead()：求之前或之后的第N行

lag和lead函数可以在一次查询中取出同一字段的前n行的数据和后n行的值。这种操作可以使用对相同表的表连接来实现，不过使用lag和lead有更高的效率。

lag(arg1,arg2,arg3)

第一个参数是列名，

第二个参数是偏移的offset，

第三个参数是超出记录窗口时的默认值。

举例如下：

```
SQL> select *  from kkk;

        ID NAME
---------- --------------------
         1 1name
         2 2name
         3 3name
         4 4name
         5 5name

SQL> select id,name,lag(name,1,0) over(order by id) from kkk;

        ID NAME                 LAG(NAME,1,0)OVER(ORDERBYID)
---------- -------------------- ----------------------------
```

```
         1 1name                0

         2 2name                1name

         3 3name                2name

         4 4name                3name

         5 5name                4name

SQL> select id,name,lead(name,1,0) over(order by id) from kkk;

        ID NAME                LEAD(NAME,1,0)OVER(ORDERBYID)

---------- -------------------- ----------------------------

         1 1name                2name

         2 2name                3name

         3 3name                4name

         4 4name                5name

         5 5name                0

SQL> select id,name,lead(name,2,0) over(order by id) from kkk;

        ID NAME                LEAD(NAME,2,0)OVER(ORDERBYID)

---------- -------------------- ----------------------------

         1 1name                3name

         2 2name                4name

         3 3name                5name

         4 4name                0

         5 5name                0

SQL> select id,name,lead(name,1,'linjiqin') over(order by id) from kkk;

        ID NAME                LEAD(NAME,1,'ALSDFJLASDJFSAF')

---------- -------------------- ------------------------------
```

1 1name 2name

2 2name 3name

3 3name 4name

4 4name 5name

5 5name linjiqin

六、rollup()与cube()：排列组合分组

1)、group by rollup(a, b, c)：

首先会对(a、b、c)进行group by，

然后再对(a、b)进行group by，

其后再对(a)进行group by，

最后对全表进行汇总操作。

2)、group by cube(a, b, c)：

则首先会对(a、b、c)进行group by，

然后依次是(a、b)，(a、c)，(a)，(b、c)，(b)，(c)，

最后对全表进行汇总操作。

1、生成演示数据：

```
Connected to Oracle Database 10g Enterprise Edition Release 10.2.0.1.0

Connected as ds_trade


SQL> conn system/oracle as sysdba

Connected to Oracle Database 10g Enterprise Edition Release 10.2.0.3.0

Connected as SYS


SQL> create table scott.t as select * from dba_indexes;
```

Table created

SQL> connect scott/oracle

Connected to Oracle Database 10g Enterprise Edition Release 10.2.0.3.0

Connected as scott

SQL>

2、普通group by体验

sql> select owner, index_type, status, count(*) from t where owner like 'SY%' group by owner, index_type, status;

OWNER	INDEX_TYPE	STATUS	COUNT(*)
SYS	CLUSTER	VALID	10
SYS	FUNCTION-BASED NORMAL	VALID	13
SYSTEM	NORMAL	N/A	32
SYSMAN	FUNCTION-BASED NORMAL	VALID	1
SYS	NORMAL	VALID	630
SYSTEM	NORMAL	VALID	147
SYS	LOB	N/A	1
SYSMAN	NORMAL	VALID	368
SYSTEM	LOB	VALID	25
SYS	NORMAL	N/A	24
SYS	IOT - TOP	VALID	33
SYS	LOB	VALID	99
SYSMAN	LOB	VALID	28
SYSMAN	IOT - TOP	VALID	27

14 rows selected

SQL>

3、group by rollup(A,B,C)

GROUP BY ROLLUP(A, B, C)：

首先会对(A、B、C)进行GROUP BY，

然后再对(A、B)进行GROUP BY，

其后再对(A)进行GROUP BY，

最后对全表进行汇总操作。

sql> select owner, index_type, status, count(*) from t where owner like 'SY%' group by ROLLUP(owner, index_type, status);

OWNER	INDEX_TYPE	STATUS	COUNT(*)
SYS	LOB	N/A	1
SYS	LOB	VALID	99
SYS	LOB		100
SYS	NORMAL	N/A	24
SYS	NORMAL	VALID	630
SYS	NORMAL		654
SYS	CLUSTER	VALID	10
SYS	CLUSTER		10
SYS	IOT - TOP	VALID	33
SYS	IOT - TOP		33
SYS	FUNCTION-BASED NORMAL	VALID	13
SYS	FUNCTION-BASED NORMAL		13
SYS			810
SYSMAN	LOB	VALID	28
SYSMAN	LOB		28
SYSMAN	NORMAL	VALID	368
SYSMAN	NORMAL		368
SYSMAN	IOT - TOP	VALID	27
SYSMAN	IOT - TOP		27
SYSMAN	FUNCTION-BASED NORMAL	VALID	1

OWNER	INDEX_TYPE	STATUS	COUNT(*)
SYSMAN	FUNCTION-BASED NORMAL		1
SYSMAN			424
SYSTEM	LOB	VALID	25
SYSTEM	LOB		25
SYSTEM	NORMAL	N/A	32

4、group by cube(A,B,C)

GROUP BY CUBE(A, B, C)：

则首先会对(A、B、C)进行GROUP BY，

然后依次是(A、B)，(A、C)，(A)，(B、C)，(B)，(C)，

最后对全表进行汇总操作。

sql> select owner, index_type, status, count(*) from t where owner like 'SY%' group by cube(owner, index_type, status);

```
OWNER                          INDEX_TYPE                  STATUS     COUNT(*)
------------------------------ --------------------------- -------- ----------
                                                                          1438
                                                           N/A              57
                                                           VALID          1381
                               LOB                                         153
                               LOB                         N/A               1
                               LOB                         VALID           152
                               NORMAL                                      1201
                               NORMAL                      N/A               56
                               NORMAL                      VALID          1145
                               CLUSTER                                       10
                               CLUSTER                     VALID             10
                               IOT - TOP                                     60
                               IOT - TOP                   VALID             60
                               FUNCTION-BASED NORMAL                         14
                               FUNCTION-BASED NORMAL       VALID             14
SYS                                                                        810
SYS                                                        N/A              25
SYS                                                        VALID           785
SYS                            LOB                                         100
SYS                            LOB                         N/A               1

OWNER                          INDEX_TYPE                  STATUS     COUNT(*)
------------------------------ --------------------------- -------- ----------
SYS                            LOB                         VALID             99
SYS                            NORMAL                                       654
SYS                            NORMAL                      N/A               24
SYS                            NORMAL                      VALID            630
```

七、max(),min(),sun()与avg()：求移动的最值总和与平均值

问题：计算出各个地区连续3个月的通话费用的平均数(移动平均值)

```
SELECT AREA_CODE, BILL_MONTH, LOCAL_FARE, SUM(LOCAL_FARE) OVER(PARTITION BY AREA_CODE ORDERBY TO_NUMBER(BILL_MONTH) RANGE BETWEEN1 PRECEDING AND1 FOLLOWING) "3month_sum", AVG(LOCAL_FARE) OVER(PARTITION BY AREA_CODE ORDERBY TO_NUMBER(BILL_MONTH) RANGE BETWEEN1 PRECEDING AND1 FOLLOWING) "3month_avg", MAX(LOCAL_FARE) OVER(PARTITION BY AREA_CODE ORDERBY TO_NUMBER(BILL_MONTH) RANGE BETWEEN1 PRECEDING AND1 FOLLOWING) "3month_max", MIN(LOCAL_FARE) OVER(PARTITION BY AREA_CODE ORDERBY TO_NUMBER(BILL_MONTH) RANGE BETWEEN1 PRECEDING AND1 FOLLOWING) "3month_min" FROM (SELECT T.AREA_CODE, T.BILL_MONTH, SUM(T.LOCAL_FARE) LOCAL_FARE FROM T GROUPBY T.AREA_CODE, T.BILL_MONTH)
```

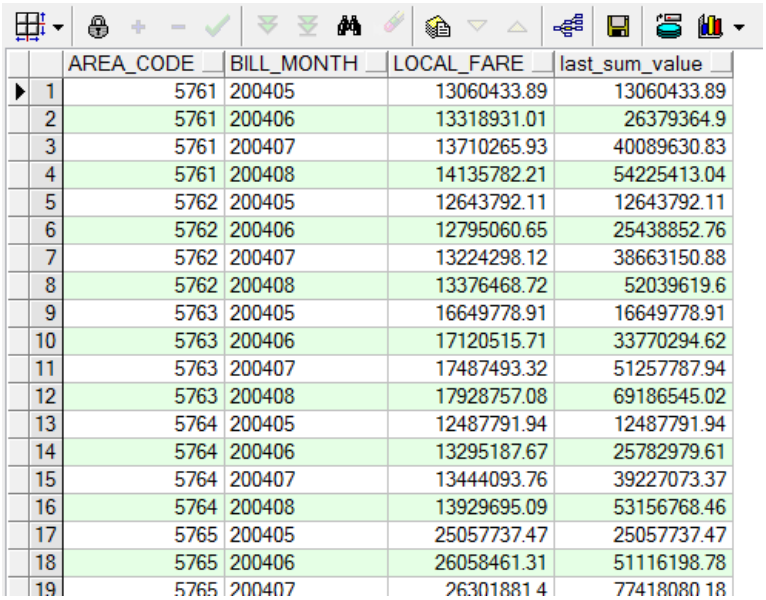
运行结果：

	AREA_CODE	BILL_MONTH	LOCAL_FARE	3month_sum	3month_avg	3month_max	3month_min
1	5761	200405	13060433.89	26379364.9	13189682.45	13318931.01	13060433.89
2	5761	200406	13318931.01	40089630.83	13363210.2766667	13710265.93	13060433.89
3	5761	200407	13710265.93	41164979.15	13721659.7166667	14135782.21	13318931.01
4	5761	200408	14135782.21	27846048.14	13923024.07	14135782.21	13710265.93
5	5762	200405	12643792.11	25438852.76	12719426.38	12795060.65	12643792.11
6	5762	200406	12795060.65	38663150.88	12887716.96	13224298.12	12643792.11
7	5762	200407	13224298.12	39395827.49	13131942.4966667	13376468.72	12795060.65
8	5762	200408	13376468.72	26600766.84	13300383.42	13376468.72	13224298.12
9	5763	200405	16649778.91	33770294.62	16885147.31	17120515.71	16649778.91
10	5763	200406	17120515.71	51257787.94	17085929.3133333	17487493.32	16649778.91
11	5763	200407	17487493.32	52536766.11	17512255.37	17928757.08	17120515.71
12	5763	200408	17928757.08	35416250.4	17708125.2	17928757.08	17487493.32
13	5764	200405	12487791.94	25782979.61	12891489.805	13295187.67	12487791.94
14	5764	200406	13295187.67	39227073.37	13075691.1233333	13444093.76	12487791.94
15	5764	200407	13444093.76	40668976.52	13556325.5066667	13929695.09	13295187.67
16	5764	200408	13929695.09	27373788.85	13686894.425	13929695.09	13444093.76
17	5765	200405	25057737.47	51116198.78	25558099.39	26058461.31	25057737.47
18	5765	200406	26058461.31	77418080.18	25806026.7266667	26301881.4	25057737.47
19	5765	200407	26301881.4	79490981.72	26496993.9066667	27130639.01	26058461.31
20	5765	200408	27130639.01	53432520.41	26716260.205	27130639.01	26301881.4

问题：求各地区按月份累加的通话费

```
SELECT AREA_CODE,        BILL_MONTH,        LOCAL_FARE,        SUM(LOCAL_FARE) OVER(PARTITION BY AREA_CODE
ORDERBY BILL_MONTH ASC) "last_sum_value"   FROM (SELECT T.AREA_CODE, T.BILL_MONTH, SUM(T.LOCAL_FARE)
LOCAL_FARE            FROM T          GROUPBY T.AREA_CODE, T.BILL_MONTH)  ORDERBY AREA_CODE, BILL_MONTH
```

运行结果：



	AREA_CODE	BILL_MONTH	LOCAL_FARE	last_sum_value
1	5761	200405	13060433.89	13060433.89
2	5761	200406	13318931.01	26379364.9
3	5761	200407	13710265.93	40089630.83
4	5761	200408	14135782.21	54225413.04
5	5762	200405	12643792.11	12643792.11
6	5762	200406	12795060.65	25438852.76
7	5762	200407	13224298.12	38663150.88
8	5762	200408	13376468.72	52039619.6
9	5763	200405	16649778.91	16649778.91
10	5763	200406	17120515.71	33770294.62
11	5763	200407	17487493.32	51257787.94
12	5763	200408	17928757.08	69186545.02
13	5764	200405	12487791.94	12487791.94
14	5764	200406	13295187.67	25782979.61
15	5764	200407	13444093.76	39227073.37
16	5764	200408	13929695.09	53156768.46
17	5765	200405	25057737.47	25057737.47
18	5765	200406	26058461.31	51116198.78
19	5765	200407	26301881.4	77418080.18

Blog：http://www.cnblogs.com/linjiqin/

J2EE、Android、Linux、Oracle QQ交流群：142463980、158560018(满)

另见：《Oracle分析函数ROW_NUMBER()|RANK()|LAG()使用详解》