第 6 章 查询

https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Select

```
[WITH CommonTableExpression (, CommonTableExpression)*]    (Note: Only available
starting with Hive 0.13.0)
SELECT [ALL | DISTINCT] select_expr, select_expr, ...
  FROM table_reference
  [WHERE where_condition]
  [GROUP BY col_list]
  [ORDER BY col_list]
  [CLUSTER BY col_list
    | [DISTRIBUTE BY col_list] [SORT BY col_list]
  ]
 [LIMIT number]
```

6.1 基本查询（Select…From）

6.1.1 全表和特定列查询

1）全表查询

```
hive (default)> select * from emp;
```

2）选择特定列查询

```
hive (default)> select empno, ename from emp;
```

注意：

（1）SQL 语言大小写不敏感。

（2）SQL 可以写在一行或者多行

（3）关键字不能被缩写也不能分行

（4）各子句一般要分行写。

（5）使用缩进提高语句的可读性。

6.1.2 列别名

1）重命名一个列。

2）便于计算。

3）紧跟列名，也可以在列名和别名之间加入关键字‘AS’

4）案例实操

（1）查询名称和部门

```
hive (default)> select ename AS name, deptno dn from emp;
```

6.1.3 算术运算符

运算符	描述
A+B	A 和 B 相加
A-B	A 减去 B
A*B	A 和 B 相乘
A/B	A 除以 B
A%B	A 对 B 取余
A&B	A 和 B 按位取与
A|B	A 和 B 按位取或
A^B	A 和 B 按位取异或
~A	A 按位取反

案例实操

查询出所有员工的薪水后加 1 显示。

```
hive (default)> select sal +1 from emp;
```

6.1.4 常用函数

1）求总行数（count）

```
hive (default)> select count(*) cnt from emp;
```

2）求工资的最大值（max）

```
hive (default)> select max(sal) max_sal from emp;
```

3）求工资的最小值（min）

```
hive (default)> select min(sal) min_sal from emp;
```

4）求工资的总和（sum）

```
hive (default)> select sum(sal) sum_sal from emp;
```

5）求工资的平均值（avg）

```
hive (default)> select avg(sal) avg_sal from emp;
```

6.1.5 Limit 语句

典型的查询会返回多行数据。LIMIT 子句用于限制返回的行数。

```
hive (default)> select * from emp limit 5;
```

6.2 Where 语句

1）使用 WHERE 子句，将不满足条件的行过滤掉。

2）WHERE 子句紧随 FROM 子句。

3）案例实操

查询出薪水大于 1000 的所有员工

```
hive (default)> select * from emp where sal >1000;
```

6.2.1 比较运算符（Between/In/ Is Null）

1）下面表中描述了谓词操作符，这些操作符同样可以用于 JOIN…ON 和 HAVING 语句中。

操作符	支持的数据类型	描述
A=B	基本数据类型	如果 A 等于 B 则返回 TRUE，反之返回 FALSE
A<=>B	基本数据类型	如果 A 和 B 都为 NULL，则返回 TRUE，其他的和等号（=）操作符的结果一致，如果任一为 NULL 则结果为 NULL
A<>B, A!=B	基本数据类型	A 或者 B 为 NULL 则返回 NULL；如果 A 不等于 B，则返回 TRUE，反之返回 FALSE
A<B	基本数据类型	A 或者 B 为 NULL，则返回 NULL；如果 A 小于 B，则返回 TRUE，反之返回 FALSE
A<=B	基本数据类型	A 或者 B 为 NULL，则返回 NULL；如果 A 小于等于 B，则返回 TRUE，反之返回 FALSE
A>B	基本数据类型	A 或者 B 为 NULL，则返回 NULL；如果 A 大于 B，则返回 TRUE，反之返回 FALSE
A>=B	基本数据类型	A 或者 B 为 NULL，则返回 NULL；如果 A 大于等于 B，则返回 TRUE，反之返回 FALSE
A [NOT] BETWEEN B AND C	基本数据类型	如果 A，B 或者 C 任一为 NULL，则结果为 NULL。如果 A 的值大于等于 B 而且小于或等于 C，则结果为 TRUE，反之为 FALSE。如果使用 NOT 关键字则可达到相反的效果。
A IS NULL	所有数据类型	如果 A 等于 NULL，则返回 TRUE，反之返回 FALSE
A IS NOT NULL	所有数据类型	如果 A 不等于 NULL，则返回 TRUE，反之返回 FALSE
IN(数值 1，数值 2)	所有数据类型	使用 IN 运算显示列表中的值
A [NOT] LIKE B	STRING 类型	B 是一个 SQL 下的简单正则表达式，如果 A 与其匹配的话，则返回 TRUE；反之返回 FALSE。B 的表达式说明如下：‘x%’表示 A 必须以字母‘x’开头，‘%x’表示 A 必须以字母’x’结尾，而‘%x%’表示 A 包含有字母’x’,可以位

		于开头，结尾或者字符串中间。如果使用 NOT 关键字则可达到相反的效果。
A RLIKE B, A REGEXP B	STRING 类型	B 是一个正则表达式，如果 A 与其匹配，则返回 TRUE；反之返回 FALSE。匹配使用的是 JDK 中的正则表达式接口实现的，因为正则也依据其中的规则。例如，正则表达式必须和整个字符串 A 相匹配，而不是只需与其字符串匹配。

2）案例实操

（1）查询出薪水等于 5000 的所有员工

```
hive (default)> select * from emp where sal =5000;
```

（2）查询工资在 500 到 1000 的员工信息

```
hive (default)> select * from emp where sal between 500 and 1000;
```

（3）查询 comm 为空的所有员工信息

```
hive (default)> select * from emp where comm is null;
```

（4）查询工资是 1500 和 5000 的员工信息

```
hive (default)> select * from emp where sal IN (1500, 5000);
```

6.2.2 Like 和 RLike

1）使用 LIKE 运算选择类似的值

2）选择条件可以包含字符或数字:

% 代表零个或多个字符(任意个字符)。

_ 代表一个字符。

3）RLIKE 子句是 Hive 中这个功能的一个扩展，其可以通过 Java 的正则表达式这个更强大的语言来指定匹配条件。

4）案例实操

（1）查找以 2 开头薪水的员工信息

```
hive (default)> select * from emp where sal LIKE '2%';
```

（2）查找第二个数值为 2 的薪水的员工信息

```
hive (default)> select * from emp where sal LIKE '_2%';
```

（3）查找薪水中含有 2 的员工信息

```
hive (default)> select * from emp where sal RLIKE '[2]';
```

6.2.3 逻辑运算符（And/Or/Not）

操作符	含义
AND	逻辑并
OR	逻辑或
NOT	逻辑否

案例实操

（1）查询薪水大于 1000，部门是 30

```
hive (default)> select * from emp where sal>1000 and deptno=30;
```

（2）查询薪水大于 1000，或者部门是 30

```
hive (default)> select * from emp where sal>1000 or deptno=30;
```

（3）查询除了 20 部门和 30 部门以外的员工信息

```
hive (default)> select * from emp where deptno not IN(30, 20);
```

6.3 分组

6.3.1 Group By 语句

GROUP BY 语句通常会和聚合函数一起使用，按照一个或者多个列队结果进行分组，然后对每个组执行聚合操作。

案例实操：

（1）计算 emp 表每个部门的平均工资

```
hive (default)> select t.deptno, avg(t.sal) avg_sal from emp t group by t.deptno;
```

（2）计算 emp 每个部门中每个岗位的最高薪水

```
hive (default)> select t.deptno, t.job, max(t.sal) max_sal from emp t group by t.deptno,
t.job;
```

6.3.2 Having 语句

1）having 与 where 不同点

（1）where 针对表中的列发挥作用，查询数据；having 针对查询结果中的列发挥作用，筛选数据。

（2）where 后面不能写分组函数，而 having 后面可以使用分组函数。

（3）having 只用于 group by 分组统计语句。

2）案例实操：

（1）求每个部门的平均薪水大于 2000 的部门

求每个部门的平均工资

```
hive (default)> select deptno, avg(sal) from emp group by deptno;
```

求每个部门的平均薪水大于 2000 的部门

```
hive (default)> select deptno, avg(sal) avg_sal from emp group by deptno having avg_sal > 2000;
```

6.4 Join 语句

6.4.1 等值 Join

Hive 支持通常的 SQL JOIN 语句，但是只支持等值连接，不支持非等值连接。

案例实操

（1）根据员工表和部门表中的部门编号相等，查询员工编号、员工名称和部门编号；

```
hive (default)> select e.empno, e.ename, d.deptno, d.dname from emp e join dept d on e.deptno = d.deptno;
```

6.4.2 表的别名

1）好处

（1）使用别名可以简化查询。

（2）使用表名前缀可以提高执行效率。

2）案例实操

合并员工表和部门表

```
hive (default)> select e.empno, e.ename, d.deptno from emp e join dept d on e.deptno = d.deptno;
```

6.4.3 内连接

内连接：只有进行连接的两个表中都存在与连接条件相匹配的数据才会被保留下来。

```
hive (default)> select e.empno, e.ename, d.deptno from emp e join dept d on e.deptno = d.deptno;
```

6.4.4 左外连接

左外连接：JOIN 操作符左边表中符合 WHERE 子句的所有记录将会被返回。

hive (default)> select e.empno, e.ename, d.deptno from emp e left join dept d on e.deptno = d.deptno;

6.4.5 右外连接

右外连接：JOIN 操作符右边表中符合 WHERE 子句的所有记录将会被返回。

hive (default)> select e.empno, e.ename, d.deptno from emp e right join dept d on e.deptno = d.deptno;

6.4.6 满外连接

满外连接：将会返回所有表中符合 WHERE 语句条件的所有记录。如果任一表的指定字段没有符合条件的值的话，那么就使用 NULL 值替代。

hive (default)> select e.empno, e.ename, d.deptno from emp e full join dept d on e.deptno = d.deptno;

6.4.7 多表连接

注意：连接 n 个表，至少需要 n-1 个连接条件。例如：连接三个表，至少需要两个连接条件。

0）数据准备

location.txt

1）创建位置表

```
create table if not exists default.location(
loc int,
loc_name string
)
row format delimited fields terminated by '\t';
```

2）导入数据

hive (default)> load data local inpath '/opt/module/datas/location.txt' into table default.location;

3）多表连接查询

```
hive (default)>SELECT e.ename, d.deptno, l. loc_name
               FROM    emp e
               JOIN    dept d
```

```
ON        d.deptno = e.deptno
JOIN      location l
ON        d.loc = l.loc;
```

大多数情况下，Hive 会对每对 JOIN 连接对象启动一个 MapReduce 任务。本例中会首先启动一个 MapReduce job 对表 e 和表 d 进行连接操作，然后会再启动一个 MapReduce job 将第一个 MapReduce job 的输出和表 l;进行连接操作。

注意：为什么不是表 d 和表 l 先进行连接操作呢？这是因为 Hive 总是按照从左到右的顺序执行的。

6.4.8 笛卡尔积

1）笛卡尔集会在下面条件下产生:

（1）省略连接条件

（2）连接条件无效

（3）所有表中的所有行互相连接

2）案例实操

```
hive (default)> select empno, deptno from emp, dept;
```

FAILED: SemanticException Column deptno Found in more than One Tables/Subqueries

6.4.9 连接谓词中不支持 or

```
hive (default)> select e.empno, e.ename, d.deptno from emp e join dept d on e.deptno = d.deptno or e.ename=d.ename;
```

错误的

6.5 排序

6.5.1 全局排序（Order By）

Order By：全局排序，一个 MapReduce

1）使用 ORDER BY 子句排序

ASC（ascend）：升序（默认）

DESC（descend）：降序

2）ORDER BY 子句在 SELECT 语句的结尾。

3）案例实操

（1）查询员工信息按工资升序排列

```
hive (default)> select * from emp order by sal;
```

（2）查询员工信息按工资降序排列

```
hive (default)> select * from emp order by sal desc;
```

6.5.2 按照别名排序

按照员工薪水的 2 倍排序

```
hive (default)> select ename, sal*2 twosal from emp order by twosal;
```

6.5.3 多个列排序

按照部门和工资升序排序

```
hive (default)> select ename, deptno, sal from emp order by deptno, sal ;
```

6.5.4 每个 MapReduce 内部排序（Sort By）

Sort By：每个 MapReduce 内部进行排序，对全局结果集来说不是排序。

1）设置 reduce 个数

```
hive (default)> set mapreduce.job.reduces=3;
```

2）查看设置 reduce 个数

```
hive (default)> set mapreduce.job.reduces;
```

3）根据部门编号降序查看员工信息

```
hive (default)> select * from emp sort by empno desc;
```

4）将查询结果导入到文件中（按照部门编号降序排序）

```
hive (default)> insert overwrite local directory '/opt/module/datas/sortby-result' select *
from emp sort by deptno desc;
```

6.5.5 分区排序（Distribute By）

Distribute By：类似 MR 中 partition，进行分区，结合 sort by 使用。

注意，Hive 要求 DISTRIBUTE BY 语句要写在 SORT BY 语句之前。

对于 distribute by 进行测试，一定要分配多 reduce 进行处理，否则无法看到 distribute by 的效果。

案例实操：

（1）先按照部门编号分区，再按照员工编号降序排序。

hive (default)> set mapreduce.job.reduces=3;

hive (default)> insert overwrite local directory '/opt/module/datas/distribute-result' select * from emp distribute by deptno sort by empno desc;

6.5.6 Cluster By

当 distribute by 和 sorts by 字段相同时，可以使用 cluster by 方式。

cluster by 除了具有 distribute by 的功能外还兼具 sort by 的功能。但是排序只能是倒序排序，不能指定排序规则为 ASC 或者 DESC。

1）以下两种写法等价

hive (default)> select * from emp cluster by deptno;

hive (default)> select * from emp distribute by deptno sort by deptno;

注意：按照部门编号分区，不一定就是固定死的数值，可以是 20 号和 30 号部门分到一个分区里面去。

6.6 分桶及抽样查询

6.6.1 分桶表数据存储

分区针对的是数据的存储路径；分桶针对的是数据文件。

分区提供一个隔离数据和优化查询的便利方式。不过，并非所有的数据集都可形成合理的分区，特别是之前所提到过的要确定合适的划分大小这个疑虑。

分桶是将数据集分解成更容易管理的若干部分的另一个技术。

1）先创建分桶表，通过直接导入数据文件的方式

（0）数据准备

student.txt

（1）创建分桶表

```
create table stu_buck(id int, name string)
clustered by(id)
into 4 buckets
row format delimited fields terminated by '\t';
```

（2）查看表结构

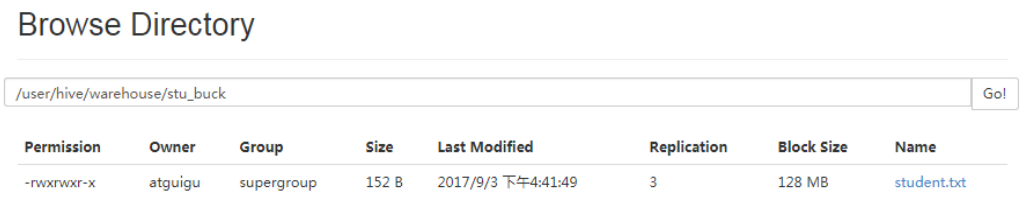
hive (default)> desc formatted stu_buck;

Num Buckets: 4

（3）导入数据到分桶表中

```
hive (default)> load data local inpath '/opt/module/datas/student.txt' into table stu_buck;
```

（4）查看创建的分桶表中是否分成 4 个桶



发现并没有分成 4 个桶。是什么原因呢？

2）创建分桶表时，数据通过子查询的方式导入

（1）先建一个普通的 stu 表

```
create table stu(id int, name string)
row format delimited fields terminated by '\t';
```

（2）向普通的 stu 表中导入数据

```
load data local inpath '/opt/module/datas/student.txt' into table stu;
```

（3）清空 stu_buck 表中数据

```
truncate table stu_buck;
select * from stu_buck;
```

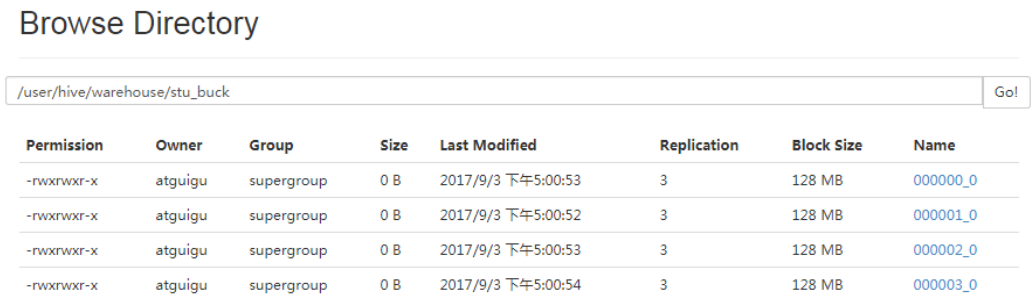
（4）导入数据到分桶表，通过子查询的方式

```
insert into table stu_buck
select id, name from stu;
```

（5）发现还是只有一个分桶

（6）需要设置一个属性

```
hive (default)> set hive.enforce.bucketing=true;
hive (default)> set mapreduce.job.reduces=-1;
hive (default)> insert into table stu_buck
                select id, name from stu;
```



（7）查询分桶的数据

```
hive (default)> select * from stu_buck;
OK
stu_buck.id	stu_buck.name
1004	ss4
1008	ss8
1012	ss12
1016	ss16
1001	ss1
1005	ss5
1009	ss9
1013	ss13
1002	ss2
1006	ss6
1010	ss10
1014	ss14
1003	ss3
1007	ss7
1011	ss11
1015	ss15
```

6.6.2 分桶抽样查询

对于非常大的数据集，有时用户需要使用的是一个具有代表性的查询结果而不是全部结果。Hive 可以通过对表进行抽样来满足这个需求。

查询表 stu_buck 中的数据。

```
hive (default)> select * from stu_buck tablesample(bucket 1 out of 4 on id);
```

注：tablesample 是抽样语句，语法：TABLESAMPLE(BUCKET x OUT OF y) 。

y 必须是 table 总 bucket 数的倍数或者因子。hive 根据 y 的大小，决定抽样的比例。例如，table 总共分了 4 份，当 y=2 时，抽取(4/2=)2 个 bucket 的数据，当 y=8 时，抽取(4/8=)1/2 个 bucket 的数据。

x 表示从哪个 bucket 开始抽取，如果需要取多个分区，以后的分区号为当前分区号加上

y。例如，table 总 bucket 数为 4，tablesample(bucket 2 out of 4)，表示总共抽取（4/2=）2 个 bucket 的数据，抽取第 1(x)个和第 3(x+y)个 bucket 的数据。

注意：x 的值必须小于等于 y 的值，否则

FAILED: SemanticException [Error 10061]: Numerator should not be bigger than denominator in sample clause for table stu_buck

6.6.3 数据块抽样

Hive 提供了另外一种按照百分比进行抽样的方式，这种是基于行数的，按照输入路径下的数据块百分比进行的抽样。

```
hive (default)> select * from stu tablesample(0.1 percent) ;
```

提示：这种抽样方式不一定适用于所有的文件格式。另外，这种抽样的最小抽样单元是一个 HDFS 数据块。因此，如果表的数据大小小于普通的块大小 128M 的话，那么将会返回所有行。

6.7 行转列查询

1）数据准备

name	constellation	blood_type
孙悟空	白羊座	A
大海	射手座	A
宋宋	白羊座	B
猪八戒	白羊座	A
凤姐	射手座	A

2）需求：把星座和血型一样的人归类到一起。结果如下：

```
射手座,A            大海|凤姐
白羊座,A            孙悟空|猪八戒
白羊座,B            宋宋
```

3）创建本地 constellation.txt，导入数据

```
[atguigu@hadoop102 datas]$ vi constellation.txt
```

```
孙悟空  白羊座  A
大海    射手座  A
宋宋    白羊座  B
猪八戒  白羊座  A
凤姐    射手座  A
```

4）创建 hive 表并导入数据

```
create table person_info(
    name string,
    constellation string,
    blood_type string)
row format delimited fields terminated by "\t";

load data local inpath “/opt/module/datas/person_info.txt” into table person_info;
```

5）按需求查询数据

```
select
    t1.base,
    concat_ws('|', collect_set(t1.name)) name
from
    (select
        name,
        concat(constellation, ",", blood_type) base
    from
        person_info) t1
group by
    t1.base;
```

6）相关函数说明

CONCAT(string A/col, string B/col…):返回输入字符串连接后的结果，支持任意个输入字符串;

CONCAT_WS(separator, str1, str2,...):它是一个特殊形式的 CONCAT()。第一个参数剩余参数间的分隔符。分隔符可以是与剩余参数一样的字符串。如果分隔符是 NULL，返回值也将为 NULL。这个函数会跳过分隔符参数后的任何 NULL 和空字符串。分隔符将被加到被连接的字符串之间;

COLLECT_SET(col):函数只接受基本数据类型，它的主要作用是将某字段的值进行去重汇总，产生 array 类型字段。

6.8 列转行查询

1）数据准备

movie	category
《疑犯追踪》	悬疑,动作,科幻,剧情
《Lie to me》	悬疑,警匪,动作,心理,剧情

《战狼 2》	战争,动作,灾难

2）需求：将电影分类中的数组数据展开。结果如下：

```
《疑犯追踪》      悬疑
《疑犯追踪》      动作
《疑犯追踪》      科幻
《疑犯追踪》      剧情
《Lie to me》    悬疑
《Lie to me》    警匪
《Lie to me》    动作
《Lie to me》    心理
《Lie to me》    剧情
《战狼 2》        战争
《战狼 2》        动作
《战狼 2》        灾难
```

3）创建本地 movie.txt，导入数据

```
[atguigu@hadoop102 datas]$ vi movie.txt
《疑犯追踪》      悬疑,动作,科幻,剧情
《Lie to me》 悬疑,警匪,动作,心理,剧情
《战狼 2》    战争,动作,灾难
```

4）创建 hive 表并导入数据

```
create table movie_info(
    movie string,
    category array<string>)
row format delimited fields terminated by "\t"
collection items terminated by ",";

load data local inpath "/opt/module/datas/movie.txt" into table movie_info;
```

5）按需求查询数据

```
select
    movie,
    category_name
from
    movie_info lateral view explode(category) table_tmp as category_name;
```

6）相关函数说明

LATERAL VIEW:

用法:LATERAL VIEW udtf(expression) tableAlias AS columnAlias

解释:用于和 split，explode 等 UDTF 一起使用，它能够将一列数据拆成多行数据，在此基础上可以对拆分后的数据进行聚合。

EXPLODE(col):将 hive 一列中复杂的 array 或者 map 结构拆分成多行。

6.9 开窗函数查询

1）数据准备:name,orderdate,cost

jack,2017-01-01,10
tony,2017-01-02,15
jack,2017-02-03,23
tony,2017-01-04,29
jack,2017-01-05,46
jack,2017-04-06,42
tony,2017-01-07,50
jack,2017-01-08,55
mart,2017-04-08,62
mart,2017-04-09,68
neil,2017-05-10,12
mart,2017-04-11,75
neil,2017-06-12,80
mart,2017-04-13,94

2）需求:

（1）查询在 2017 年 4 月份购买过的顾客及总人数

（2）查询顾客的购买明细及月购买总额

（3）上述的场景,要将 cost 按照日期进行累加

（4）查询顾客上次的购买时间

（5）查询前 20%时间的订单信息

3）创建本地 business.txt，导入数据

[atguigu@hadoop102 datas]$ vi business.txt

4）创建 hive 表并导入数据

```
create table business(
name string,
orderdate
string,cost int
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

load data local inpath "/opt/module/datas/business.txt" into table business;
```

5）按需求查询数据

（1）查询在 2017 年 4 月份购买过的顾客及总人数

```
select name,count(*) over ()
```

```
from business
where substring(orderdate,1,7) = '2015-04'
group by name;
```

（2）查询顾客的购买明细及月购买总额

```
select name,orderdate,cost,sum(cost) over(partition by month(orderdate)) from business;
```

（3）上述的场景,要将 cost 按照日期进行累加

```
select name,orderdate,cost,
sum(cost) over() as sample1,--所有行相加
sum(cost) over(partition by name) as sample2,--按 name 分组，组内数据相加
sum(cost) over(partition by name order by orderdate) as sample3,--按 name 分组，组内数据累加
sum(cost) over(partition by name order by orderdate rows between UNBOUNDED PRECEDING
and current row ) as sample4 ,--和 sample3 一样,由起点到当前行的聚合
sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING and current
row) as sample5, --当前行和前面一行做聚合
sum(cost) over(partition by name order by orderdate rows between 1 PRECEDING AND 1
FOLLOWING ) as sample6,--当前行和前边一行及后面一行
sum(cost) over(partition by name order by orderdate rows between current row and
UNBOUNDED FOLLOWING ) as sample7 --当前行及后面所有行
from business;
```

（4）查看顾客上次的购买时间

```
select name,orderdate,cost,
lag(orderdate,1,'1900-01-01') over(partition by name order by orderdate ) as time1,
lag(orderdate,2) over (partition by name order by orderdate) as time2
from business;
```

（5）查询前 20%时间的订单信息

```
select * from (
    select name,orderdate,cost, ntile(5) over(order by orderdate) sorted
    from business
) t
where sorted = 1;
```

6）相关函数说明

OVER():指定分析函数工作的数据窗口大小，这个数据窗口大小可能会随着行的变化而变化

CURRENT ROW:当前行

PRECEDING n:往前 n 行数据

FOLLOWING n:往后 n 行数据

UNBOUNDED:起点，UNBOUNDED PRECEDING 表示从前面的起点， UNBOUNDED FOLLOWING 表示到后面的终点

LAG(col,n):往前第 n 行数据

LEAD(col,n):往后第 n 行数据

NTILE(n):把有序分区中的行分发到指定数据的组中，各个组有编号，编号从 1 开始，对于每一行，NTILE 返回此行所属的组的编号。注意：n 必须为 int 类型。