```
create external table stu_external(id int,name string)
partitioned by (month string,day string)
row format delimited fields terminated by '\t'
location '/student';

load data local inpath '/home/centos/hiveSource/student.txt' into table
stu_external partition(month='201708',day='12');

desc formatted stu_external;
```

where子句 紧跟from 子句，where子句中不能使用字段别名

group by 后面的查询字段要和 from 前面的查询字段保持一致

给表起别名 提高查询效率

计算emp表每个部门的平均工资：

```
select e.ename,e.deptno,avg(e.sal)from emp e group by e.ename,e.deptno;
```

计算emp每个部门中每个岗位的最高薪水：

```
select e.job,e.deptno,max(e.sal)from emp e group by e.deptno,e.job;
```

where后面不能跟分组函数，having后面可以跟分组函数

having 只能用于分组查询的条件语句

having 条件里用到了 前面表里的字段，需要给这个字段起别名

求每个部门的平均工资：

```
select e.deptno,avg(sal) avg_sal from emp e group by e.deptno;
```

求每个部门的平均薪水大于2000的部门：

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

根据员工表和部门表中的部门编号相等，查询员工编号、员工名称和部门名称；

```
select e.deptno,e.ename,e.empno,d.deptno,d.dname
```

```
from emp e join dept d on e.deptno = d.deptno;
```

左外连接

```
select e.deptno,e.ename,d.deptno from emp e left join dept d on e.deptno = d.deptno;
```

多表连接

```
select e.ename,d.dname,l.loc_name
from emp e
join dept d
on d.deptno = e.deptno
join location l
on d.loc = l.loc;
```

Hive会对每对JOIN连接对象启动一个MapReduce任务

优化：当对3个或者更多表进行join连接时，
如果每个on子句都使用相同的连接键的话，
那么只会产生一个MapReduce job。

注意: join 中是不支持or的

无论多少个子句order by(全局排序,只有一个Reducer) 子句一定要在SELECT语句的结尾

按照部门和工资升序排序:

```
select e.ename,e.deptno,e.sal from emp e order by e.deptno,e.sal asc;
```

distribute by:分区排序 ,控制某个特定行应该到哪个reducer,结合sort by(每个mapreduce 内部排序)使用

distribute by 子句要写在sort by 子句之前

求出不同部门男女各多少人:

```
select dept_id,
sum(case sex when '男' then 1 else 0 end) maleCount,
sum(case sex when '女' then 1 else 0 end) femaleCount
```

```
from emp_sex group by dept_id;
```

行转列:

要求把星座和血型一样的人归类到一起:

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

列转行:

```
select movie,category_info
from movie_info
lateral view explode(category) tbl_tmp as category_info;
```

查询在2017年4月份购买过的顾客及总人数:

区别两种写法:

这个sql统计的是按名字分组后，每个名字出现几次

```
select name,count(*) from business
where substring(orderdate,1,7) = '2017-04'
group by name;
```

(这个sql 统计的是按名字分组后，不同名字的个数)

```
select name,count(*) over() from business
where substring(orderdate,1,7) = '2017-04'
group by name;
```

查询顾客的购买明细及月购买总额:

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

根据上述的场景，将每个顾客的cost按照日期进行累加

```
select b.name,b.cost,b.orderdate,
```

```
sum(b.cost) over(partition by name order by orderdate) from business b;
```

或者可以这样写:

```
select b.name,b.orderdate,b.cost,sum(b.cost)
over(partition by name rows between unbounded preceding and current row)
from business b;
```

查询每个顾客上次的购买时间:

```
select b.name,b.orderdate,b.cost,lag(b.orderdate,1,-1)
over(partition by b.name order by b.orderdate)
from business b;
```

查询前20%时间的订单信息:

ntile(5):会给给每一条数据后面添加一个组的编号:

```
select * from
(select b.name,b.cost,b.orderdate,ntile(5) over(order by b.orderdate) num
from business b) t1 where t1.num = 1;
```

rank() 排序相同时会重复，总数不会变

dense_rank() 排序相同时会重复，总数会减少

row_number() 会根据顺序计算

```
core dense_rank rank row_number
100 1 1 1
99 2 2 2
99 2 2 3
98 3 4 4
96 4 5 5
96 4 5 6
95 5 7 7
```

计算每门学科成绩排名:

```
select name,subject,
```

```
score,
rank() over(partition by subject order by score desc) rank1,
dense_rank() over(partition by subject order by score desc) rank2,
row_number() over(partition by subject order by score desc) rank3
from score
```

统计视频观看数Top10:

使用order by按照views字段做一个全局排序即可，同时我们设置只显示前10条。

```
select videoId,uploader,age,category,length,views,rate,ratings,comments
from gulivideo_orc
order by views
desc limit 10;
```

统计视频类别热度Top10:

思路:

1）即统计每个类别有多少个视频，显示出包含视频最多的前10个类别。

2）我们需要按照类别group by聚合，然后count组内的videoId个数即可。

3）因为当前表结构为：一个视频对应一个或多个类别。所以如果要group by类别，需要先将类别进行列转行(展开)，然后再进行count即可。

4）最后按照热度排序，显示前10条。

```
select category_name , count(t1.videoId) as hot
from (select videoId,category_name from
gulivideo_orc lateral view explode(category) t_catetory as category_name) t1
group by t1.category_name
order by hot
desc limit 10;
```

统计出视频观看数最高的20个视频的所属类别以及类别包含Top20视频的个数:

思路:

1）先找到观看数最高的20个视频所属条目的所有信息，降序排列

2）把这20条信息中的category分裂出来(列转行)

3）最后查询视频分类名称和该分类下有多少个Top20的视频

```
select
```

```
category_name as category,
count(t2.videoId) as hot_with_views
from (select videoId,category_name from
(select * from gulivideo_orc order by views desc limit 20) t1 lateral view explode(category) t_catetory as category_name) t2
group by category_name
order by hot_with_views
desc;
```

统计视频观看数Top50所关联视频的所属类别排序:

1)查询出观看数最多的前50个视频的所有信息(当然包含了每个视频对应的关联视频)，记为临时表t1

2)将找到的50条视频信息的相关视频relatedId列转行，记为临时表t2

3)将相关视频的id和gulivideo_orc表进行inner join操作

4)得到两列数据，一列是category，一列是之前查询出来的相关视频id

5)按照视频类别进行分组，统计每组视频个数，然后排行

```
select category_name as category, count(t5.videoId) as hot
from (select videoId, category_name
from (select distinct(t2.videoId), t3.category
from (select explode(relatedId) as videoId
from (select *
from gulivideo_orc order by views desc limit 50) t1) t2 inner join gulivideo_orc t3 on t2.videoId = t3.videoId) t4 lateral view explode(category) t_catetory as category_name) t5
group by
category_name
order by
hot
desc;
```

统计每个类别视频观看数Top10:

1）先得到categoryId展开的表数据

2）子查询按照categoryId进行分区，然后分区内排序，并生成递增数字，该递增数字这一列起名为rank列

3）通过子查询产生的临时表，查询rank值小于等于10的数据行即可。

```
select t1.*
from (select videoId,categoryId,views,row_number() over(partition by categoryId order by views desc)
rank from gulivideo_category) t1
where rank <= 10;
```

统计每个类别中视频流量Top10，以Music为例：

1)创建视频类别展开表（categoryId列转行后的表）

2)按照ratings排序即可

```
select videoId,views,ratings
from gulivideo_category
where categoryId = "Music"
order by ratings
desc limit 10;
```

统计上传视频最多的用户Top10以及他们上传的观看次数在前20的视频：

1)先找到上传视频最多的10个用户的用户信息

2)通过uploader字段与gulivideo_orc表进行join，得到的信息按照views观看次数进行排序即可。

```
select t2.videoId,t2.views,t2.ratings,t1.videos,t1.friends
from (select * from gulivideo_user_orc order by videos desc limit 10) t1
join gulivideo_orc t2
on t1.uploader = t2.uploader
order by views desc
limit 20;
```

背景说明：

以下表记录了用户每天的蚂蚁森林低碳生活领取的记录流水。

table_name：user_low_carbon

```
seq（key）  user_id data_dt  low_carbon
流水号      用户      日期       减少碳排放（g）
```

xxxxx01 u_001 2017/1/1 10

xxxxx02 u_001 2017/1/2 150

xxxxx03 u_001 2017/1/2 110

xxxxx04 u_001 2017/1/2 10

xxxxx05 u_001 2017/1/4 50

xxxxx06 u_001 2017/1/4 10

xxxxx07 u_001 2017/1/6 45

xxxxx08 u_001 2017/1/6 90

xxxxx09 u_002 2017/1/1 10

xxxxx10 u_002 2017/1/2 150

xxxxx11 u_002 2017/1/2 70

xxxxx12 u_002 2017/1/3 30

xxxxx13 u_002 2017/1/3 80

xxxxx14 u_002 2017/1/4 150

xxxxx14 u_002 2017/1/5 101

xxxxx15 u_002 2017/1/6 68

xxxxx16 u_002 2017/1/6 120

蚂蚁森林植物换购表，用于记录申领环保植物所需要减少的碳排放量

table_name: plant_carbon

plant_id plant_name low_carbon

植物编号 植物名 换购植物所需要的碳

p001 梭梭树 17900

p002 沙柳 19680

p003 樟子树 146210

p004 胡杨 215680

----题目

1、蚂蚁森林植物申领统计

问题：假设2017年1月1日开始记录低碳数据（user_low_carbon），假设2017年10月1日之前满足申领条件的用户都申领了一颗p004-胡杨，

剩余的能量全部用来领取“p002-沙柳” 。

统计在10月1日累计申领“p002-沙柳” 排名前10的用户信息；以及他比后一名多领了几颗沙柳。

得到的统计结果如下表样式：

user_id plant_count less_count(比后一名多领了几颗沙柳)

u_101   1000   100

u_088   900   400

u_103   500   …

1.按照用户ID进行能量累加

```
select user_id,sum(low_carbon) low_carbon_sum

from user_low_carbon

group by user_id

where data_dt<=2017/10/1

order by low_carbon_sum

desc limit 11;
```

2.

```
select user_id,sum(low_carbon) over(partitions by user_id) low_carbon_sum
from user_low_carbon where data_dt<2017/10/1;t1

select low_carbon from plant_carbon where plant_id='p004';t2

select low_carbon from plant_carbon where plant_id='p002';t3

select user_id,

(low_carbon_sum-t2.low_carbon)\t3.low_carbon over(partitions by user_id)
plant_count,

from t1,t2,t3

sort by plant_count

DESC limit 10;t4

select user_id,plant_count,plant_count-lead(plant_count) from t4;
```

最终SQL：

```
SELECT

user_id,

plant_count,

plant_count - lead (plant_count,1)

FROM

(

SELECT
```

user_id,

(

low_carbon_sum - t2.low_carbon

) \ t3.low_carbon over (PARTITIONS BY user_id) plant_count,

```
    FROM
        (
            SELECT
                user_id,
                sum(low_carbon) over (PARTITIONS BY user_id) low_carbon_sum
            FROM
                user_low_carbon
            WHERE
                data_dt < 2017 / 10 / 1
        ) t1,
        (
            SELECT
                low_carbon
            FROM
                plant_carbon
            WHERE
                plant_id = 'p004'
        ) t2,
        (
            SELECT
                low_carbon
            FROM
                plant_carbon
            WHERE
                plant_id = 'p002'
        ) t3 sort BY plant_count DESC
    LIMIT 10
);
```

2、蚂蚁森林低碳用户排名分析

问题：查询user_low_carbon表中每日流水记录，条件为：

用户在2017年，连续三天（或以上）的天数里，

每天减少碳排放（low_carbon）都超过100g的用户低碳流水。

需要查询返回满足以上条件的user_low_carbon表中的记录流水。

例如用户u_002符合条件的记录如下，因为2017/1/2~2017/1/5连续四天的碳排放量之和都大于等于100g：

seq（key） user_id data_dt low_carbon

xxxxx10 u_002 2017/1/2 150

xxxxx11 u_002 2017/1/2 70

xxxxx12 u_002 2017/1/3 30

xxxxx13 u_002 2017/1/3 80

xxxxx14 u_002 2017/1/4 150

xxxxx14 u_002 2017/1/5 101

1.

```
select user_id,data_dt,sum(low_carbon) over(PARTITION BY user_id,data_dt) low_carbon_sum
from user_low_carbon
where low_carbon_sum>100 and substring(data_dt,1,4)='2017';

select user_id,sum(low_carbon) over(PARTITION by user_id,data_dt order by data_dt) low_carbon_sum from user_low_carbon
having low_carbon_sum>100;t1
```

Mapper:

```
String[] split = value.toString().split("\t");

String user_id = split[0];
String date = split[1];
String low_carbon_sum = split[2];

k.set(user_id);
v.set(date + "\t" + low_carbon_sum);

context.write(k, v);
```

Reducer:

```
int date1 = 0;
```

```
int date2;
for (Text value : values) {
String[] split = value.toString().split("\t");
String dataStr = split[0];
date2 = Integer.parseInt(dataStr);
if (date2 - date1 <= 1 && date1 == 0) {
list.add(value.toString());
date1 = Integer.parseInt(dataStr);
} else {
if (list.size() >= 3) {
for (String s : list) {
v.set(s);
context.write(key, v);
}
date1 = 0;
list.clear();
}
}
}
```

备注：统计方法不限于sql、procedure、python,java等

sql:思想技巧总结：

当需求中设计关键词“每个”时 考虑用group by 和"聚合函数" + over(partition by xxx order by xxx)，选哪个就要看

限制条件是不是 聚合函数，如果不是聚合函数 就用group by、如果是 选择后者

当需求中涉及关键词“类别” 首先看后面的条件和类别是否存在一对多的关系，考虑是否可用laternal explode()

当需求中涉及关键词“前20、后20”就要用order by xxx desc/asc limit count

涉及到两个表或者多个表的的时候 就要用join on

设计到排名问题 可以考虑 rank() + over()。。。。。或者order by xxx desc/asc limit

count

怎么写sql呢，先根据需求，想像出要出的表结构，在结合知识写语句