hivesql使用总结

常用命令

直接上例子

```
1. 1)修改标名称: ALTER TABLE livepm.live_anchor_daily_report_cc RENAME TO li
   vepm.live_anchor_daily_report_cc_4furong
2. 2)导入表: load data inpath '/user/livepm/deviceid.tmp' into table livepm.
   wangchan_device_data
3. 3)查看某个分区是否存在: show partitions online.bl_molive_event_detail parti
   tion(partition_date='20170809')
4. 4)删除某个分区: alter table live_anchor_user_event_theme_audio_cc drop par
   tition(partition_date=20170928);
5. 5)修改表字段名称 / 类型: alter table livepm.live_momo_audio_entrance_data_cc
    change status status string
```

hive 分析函数总结及实例

1. 级,百分点，n分片等

(1) Ntile

- 基本描述：把有序的数据集合平均分配到指定的数量个桶中，将桶号分配给每一行。如果不能平均分配，则优先分配较小编号的桶，并且各个桶中能放的行数最多相差元。
- 语法：

> `ntile(num) over([partition_clause] order_by_clause) as your_bucket_num`

- 然后根据桶号，选取前或后n分之几的平均消费。
- 通俗解释(bycc):将数据平均分配为num份,若数据集本身记录数不是偶数，则不能平均分配，优先分配给小编号

实例

```
create table livepm.audio_consum_data_daily_cc_ntile as
select
to_date(create_time) st_date,
momo_id,
sum(current_price) consum,
ntile(2) over(order by sum(current_price) desc) as partnum
from online.ml_molive_order_detail
where  current_price>0
and is_package=0
and live_mode=1
and partition_date=20171127
and to_date(create_time)='2017-11-27'
group by to_date(create_time),momo_id

```

(2) Rank,Dense_Rank,Row_Number

实例

```
select
to_date(create_time) st_date,
momo_id,
sum(current_price) consum,
rank() over(partition by to_date(create_time) order by sum(current_price) desc) rank_num,--若遇到相同的sum(current_price),则，rank_num输出相同的序号,且下一个序号不间断，即若两个都排3，则第5个rank_num从5开始
dense_rank() over(partition by to_date(create_time) order by sum(current_price) desc) denserank_num,--若遇到相同的sum(current_price),则，rank_num输出相同的序号,且下一个序号间断，即若两个都排3，则第5个rank_num从4开始
row_number() over(partition by to_date(create_time) order by sum(current_price) desc) rownumber_num --所有数值输出不同的序号，唯一且连续
from online.ml_molive_order_detail
where  current_price>0
and is_package=0
and live_mode=1
and partition_date=20171127
and to_date(create_time)>='2017-11-01'
```

```
group by to_date(create_time),momo_id;

```

2. Lag, Lead, First_value, Last_value

(1) lag,lead

- `lag(col,n,default)` 用于统计窗口内网上第n行值
- `lead(col,n,default)` 用于统计窗口内往下第n行值，与lag相反
- 以上两个函数内的第三个参数，如果不填写，模式值为null，若填写，则为null的就为设置的默认值

实例

```
select
to_date(create_time) st_date,
momo_id,
sum(current_price) consum,
lag(to_date(create_time),1,0000) over(partition by momo_id order by to_date(create_time) asc) next_cosumdate01,
--各用户每日消费记录的上一次消费日期，若无则设置为0000
lag(to_date(create_time),1,0000) over(partition by momo_id order by to_date(create_time) desc) next_cosumdate02,
--各用户没日消费记录的下一次消费日期
lead(to_date(create_time),1,0000) over(partition by momo_id order by to_date(create_time) asc) next_cosumdate03,
--各用户没日消费记录的下一次消费日期，若无则设置为0000
lead(to_date(create_time),1,0000) over(partition by momo_id order by to_date(create_time) desc) next_cosumdate04
--各用户没日消费记录的上一次消费日期
from online.ml_molive_order_detail
where  current_price>0
and is_package=0
and live_mode=1
and partition_date=20171127
--and to_date(create_time)>='2017-01-01'
group by to_date(create_time),momo_id;
```

(2) FIRST_VALUE, LAST_VALUE

- FIRST_VALUE：取分组内排序后，截止到当前行往前的首个值
- LAST_VALUE：取分组内排序后，截止到当前行往前的最后一个值
- first_value(desc): 获得组内全局的最后一个值

```
create table livepm.audio_consum_data_daily_cc_first_last as
select
to_date(create_time) st_date,
momo_id,
sum(current_price) consum,
--生序排序，则返回排序规则最小值，降序则返回排序规最大值
```

```
first_value(sum(current_price)) over(partition by to_date(create_time) order by sum(current_price)) consum_first,
last_value(sum(current_price)) over(partition by to_date(create_time) order by sum(current_price)) consum_last,
first_value(sum(current_price)) over(partition by to_date(create_time) order by sum(current_price) desc) consum_last_golbal
from online.ml_molive_order_detail
where  current_price>0
and is_package=0
and live_mode=1
and partition_date=20171127
and to_date(create_time)>='2017-01-01'
group by to_date(create_time),momo_id;
```

3. SUM,AVG,MIN,MAX

(1) sum

- 以sum() over() 为例子
- 特别注意以下

4.窗口函数总结（参考sql sever）

- 窗口函数与聚合函数的区别：通常来说，聚合后的行数都要小于聚合前的行数。而对于窗口函数来说，输入结果等于输出结果
- 执行顺序：窗口函数是整个SQL语句最后被执行的部分，这意味着窗口函数是在SQL查询的结果集上进行的，因此不会受到Group By， Having，Where子句的影响（窗口函数是SQL语句最后执行的函数，因此可以把SQL结果集想象成输入数据）。

> PRECEDING：往前
> FOLLOWING：往后
> CURRENT ROW：当前行
> UNBOUNDED：起点，
> UNBOUNDED PRECEDING 表示从前面的起点，
> UNBOUNDED FOLLOWING：表示到后面的终点

5.其他函数总结（参考sql sever）

```
GROUPING SETS 使用方法
SELECT a, b, SUM( c ) FROM tab1 GROUP BY a, b GROUPING SETS ( (a, b), a, b, ( ) )
--- 和以下结果一样
SELECT a, b, SUM( c ) FROM tab1 GROUP BY a, b
```

```
UNION all
SELECT a, null, SUM( c ) FROM tab1 GROUP BY a, null
UNION all
SELECT null, b, SUM( c ) FROM tab1 GROUP BY null, b
UNION all
SELECT null, null, SUM( c ) FROM tab1


with cube 使用方法
GROUP BY a, b, c WITH CUBE is equivalent to
GROUP BY a, b, c GROUPING SETS ( (a, b, c), (a, b), (b, c), (a, c),
 (a), (b), (c), ( )).


with ROLLUP 使用方法
GROUP BY a, b, c, WITH ROLLUP is equivalent to GROUP BY a, b, c GROUPIN
G SETS ( (a, b, c), (a, b), (a), ( )).
```

```
create table livepm.audio_consum_data_daily_cc_over as
select
to_date(create_time) st_date,
momo_id,
sum(current_price) consum,
--从起点到当前行
sum(current_price) over(partition by to_date(create_time) order by
sum(current_price)) consum01,
--从起点到当前行
sum(current_price) over(partition by to_date(create_time) order by
sum(current_price) ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) co
nsum02,
--当前行+往前3行
sum(current_price) over(partition by to_date(create_time) order by
sum(current_price) ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) consum03,
--当前行+往前3行+往后1行
sum(current_price) over(partition by to_date(create_time) order by
sum(current_price) ROWS BETWEEN 3 PRECEDING AND 1 FOLLOWING) consum04,
---当前行+往后所有行
sum(current_price) over(partition by to_date(create_time) order by
sum(current_price) ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING) co
nsum04
from online.ml_molive_order_detail
where  current_price>0
and is_package=0
and live_mode=1
and partition_date=20171127
and to_date(create_time)>='2017-01-01'
group by to_date(create_time),momo_id
;
```

```
24.
```

hive 常用统计函数总结及实例

percentile_approx(参数，array(0.05,0.5,...),default) ,计算百分位数，其中`array()里的值，不包括0和1

```
1. select
2.         percentile_approx(cast(extend_map['access_duration'] as
   double),
3.         array(0.05,0.1,0.2,0.25,0.3,0.4,0.5,0.6,0.7,0.75,0.8,0.9,0.95))
4. from online.bl_molive_uniform_event_detail
5. where partition_date='20171218' and event_name = 'live.v3.room.p.exit_r
   oom' and extend_map['access_duration']>=0
6. and momo_id!=10013
7.
```

stddev 标准差
stddev_pop 标准差
stddev_samp 样本标准差
var_pop 方差
var_samp 样本方差

删除表的某列

目前没发现直接删除的，只能用更改表的语句，如下
```sql
alter table livepm.live_audio_anchors_incomeoflayer_report_m_cc replace columns
(
st_month string comment '统计月份',
income_level string comment '营收分层',
usertype string comment '用户类型',
is_guild string comment '是否工会',
anchor_num string comment '播主数',
audio_income double comment '语音收益',
audio_showtime string comment '语音开播时长',
total_income double comment '总收益',
total_showtime string comment'总开播时长'
)
```

表增加字段

```
```

```
1.  alter table livepm.cc_table add columns(c1 int,c2 string)
```

新增的字段将加到表的最有，分区字段之前，并且在以前的分区中，这两个字段都为NULL

hive表总结

建表基本字段说明：

1. `partitioned` 表示的是分区，不同的分区会以文件夹的形式存在，在查询的时候制定分区查询将会大大加快查询的时间
2. `clustered` 表示的是按照某列聚类,例如在插入数据中有两项 “张三，数学“和”张三，英语”，若是 `clustered by name` ,则只会一项，“张三，(数学,英语)”，这个机制也是为了加快查询的操作
3. `stored` 是指定排序的形式，是降序还是升序。
4. `buckets` 是置顶了分桶的信息，
5. `row format` 是置顶了行的参数,还要指定列的信息，如 `row format delimited fields terminated by '\t' lines terminated by '\n'`
6. `stored as` 是指定文件的存储格式。hive中基本提供两种文件格式:sequencefile和textfile,序列文件是一种压缩的格式,通常可以提供更高的性能
7. `location` 指的是在hdfs存储的位置
8. ALTER TABLE livepm.live_category_anchor_4bi_cc CHANGE choose_tags choose_tags string

执行顺序

Group By 和 Having, Where ,Order by这些关键字是按照如下顺序进行执行的：Where, Group By, Having, Order by。只有order by语句可以只用最终视图的列名（有些数据库语言不是，有得在having层就可以使用）

不熟悉的总结

1) WITH CUBE
2) grouping sets

```
1.  -- grouping sets
2.  select
3.         order_id,
4.         departure_date,
5.         count(*) as cnt
6.    from ord_test
7.   where order_id=410341346
8.   group by order_id,
9.         departure_date
```

```
 grouping sets (order_id,(order_id,departure_date))
;

---- 等价于以下
group by order_id
union all
group by order_id,departure_date
```

3) with rollup
4)复制表结构 CREATE TABLE livepm.live_anchor_audio_detail_data_whitelist_cc LIKE livepm.live_anchor_audio_detail_data_v2_cc

常用总结

datediff(p1,p2) ;p1>p2, 返回值大于0，当p1<p2，返回值小于0

建表语句说明

```
--- external 说明的是一张外部表
CREATE EXTERNAL TABLE IF NOT EXISTS offline.tl_guild_anchor_transfer_data_import
(
key STRING COMMENT 'rediskey',
member STRING COMMENT '成员'
)
PARTITIONED BY ( partition_date STRING COMMENT 'par' )
---行格式分隔符
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\001' ---- 列格式分隔符为'\001'
LINES TERMINATED BY '\n' --行分隔符为''\n

--  COLLECTION ITEMS TERMINATED BY '\002'
-- MAP KEYS TERMINATED BY '\003'

-- 表明文件数据是纯文本
STORED AS TEXTFILE;
```

以上更多可参考

一些统计需求

求连续活跃天频：https://blog.csdn.net/ganghaodream/article/details/100083543?utm_medium=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_t0.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase

> 个人尝试

原始数据

需求：统计各数值连续出现最大次数

ID	num
1	1
2	1
3	1
4	2
5	1
6	2
7	1

第一步处理：按照NUM分组，ID升序排序，并新增排序ID(row_number() over(partition by num order by id) as ranknum,得到如下数据

ID	NUM	ranknum
1	1	1
2	1	2
3	1	3
5	1	4
7	1	5
4	2	1
6	2	2

第二步处理：增加辅助列help_index=ID-ranknum

ID	NUM	ranknum	help_index
1	1	1	0
2	1	2	0
3	1	3	0
5	1	4	1
7	1	5	2
4	2	1	3
6	2	2	4

第三步处理：按照NUM，help_index,分组，统计条数，即可得到1,help_index出现的次数，即去1最大的次数即为出现次数

NUM	help_index	show_num
1	0	3
1	1	1
1	2	1
2	3	1
2	4	1