DAY5

1148. 文章浏览 I

难度：简单

`Views` 表：

Column Name	Type
article_id	int
author_id	int
viewer_id	int
view_date	date

此表无主键，因此可能会存在重复行。
此表的每一行都表示某人在某天浏览了某位作者的某篇文章。
请注意，同一人的 author_id 和 viewer_id 是相同的。

请编写一条 SQL 查询以找出所有浏览过自己文章的作者，结果按照 id 升序排列。

查询结果的格式如下所示：

`Views` 表：

article_id	author_id	viewer_id	view_date
1	3	5	2019-08-01
1	3	6	2019-08-02
2	7	7	2019-08-01
2	7	6	2019-08-02
4	7	1	2019-07-22
3	4	4	2019-07-21
3	4	4	2019-07-21

结果表：

id
4
7

代码

```
# 找自信题
select
    distinct author_id as id
from Views
where author_id = viewer_id
order by id
```

1149. 文章浏览 II【中等】

难度：中等

编写一条 SQL 查询来找出在同一天阅读至少两篇文章的人。
结果按照 id 升序排序。

查询结果的格式如下。
示例 1:
输入：
`Views` 表:

article_id	author_id	viewer_id	view_date
1	3	5	2019-08-01
3	4	5	2019-08-01
1	3	6	2019-08-02
2	7	7	2019-08-01
2	7	6	2019-08-02
4	7	1	2019-07-22
3	4	4	2019-07-21
3	4	4	2019-07-21

输出：

id
6

代码

```
select
    distinct viewer_id as id
from Views
group by view_date, viewer_id
having count(distinct article_id)>=2 # 注意这里的distinct
order by id
```

1211. 查询结果的质量和占比

难度：简单

查询表 Queries：

Column Name	Type
query_name	varchar
result	varchar
position	int
rating	int

此表没有主键，并可能有重复的行。
此表包含了一些从数据库中收集的查询信息。
“位置”（position）列的值为 1 到 500 。
“评分”（rating）列的值为 1 到 5 。评分小于 3 的查询被定义为质量很差的查询。

将查询结果的质量 quality 定义为：
各查询结果的评分与其位置之间比率的平均值。
将劣质查询百分比 poor_query_percentage 为：
评分小于 3 的查询结果占全部查询结果的百分比。

编写一组 SQL 来查找每次查询的名称(query_name)、质量(quality) 和 劣质查询百分比(poor_query_percentage)。
质量(quality) 和劣质查询百分比(poor_query_percentage) 都应四舍五入到小数点后两位。

查询结果格式如下所示：
Queries table:

query_name	result	position	rating
Dog	Golden Retriever	1	5
Dog	German Shepherd	2	5
Dog	Mule	200	1
Cat	Shirazi	5	2
Cat	Siamese	3	3
Cat	Sphynx	7	4

Result table:

query_name	quality	poor_query_percentage
Dog	2.50	33.33
Cat	0.66	33.33

Dog 查询结果的质量为 ((5 / 1) + (5 / 2) + (1 / 200)) / 3 = 2.50
Dog 查询结果的劣质查询百分比为 (1 / 3) * 100 = 33.33

Cat 查询结果的质量为 ((2 / 5) + (3 / 3) + (4 / 7)) / 3 = 0.66
Cat 查询结果的劣质查询百分比为 (1 / 3) * 100 = 33.33

代码

```
# 注意百分比
select
    query_name
    , round(ifnull(avg(quality_tmp),0),2) as quality
```

```
    , round(ifnull(100 * sum(poor_query)/count(poor_query),0),2) as
poor_query_percentage
from(
    select
    a.*
    , rating/position as quality_tmp
    , if(rating<3,1,0) as poor_query
    from Queries as a
) as tmp
group by query_name

# 可以直接这么写
select
    distinct query_name,
    ifnull(round(avg(rating/position),2),2) as quality,
    ifnull(round(100*sum(rating<3)/count(*),2),2) as poor_query_percentage
from Queries
group by query_name
```

1241. 每个帖子的评论数

难度：简单

表 `Submissions` 结构如下：

列名	类型
sub_id	int
parent_id	int

上表没有主键, 所以可能会出现重复的行。
每行可以是一个帖子或对该帖子的评论。
如果是帖子的话，parent_id 就是 null。
对于评论来说，parent_id 就是表中对应帖子的 sub_id。

编写 SQL 语句以查找每个帖子的评论数。
结果表应包含帖子的 post_id 和对应的评论数 number_of_comments 并且按 post_id 升序排列。
Submissions 可能包含重复的评论。您应该计算每个帖子的唯一评论数。
Submissions 可能包含重复的帖子。您应该将它们视为一个帖子。
结果表应该按 post_id 升序排序。

查询结果格式如下例所示。
示例 1:
输入：
`Submissions` table:

sub_id	parent_id
1	Null
2	Null
1	Null
12	Null
3	1
5	2
3	1
4	1
9	1
10	2
6	7

输出：

post_id	number_of_comments
1	3
2	2
12	0

解释：
表中 ID 为 1 的帖子有 ID 为 3、4 和 9 的三个评论。表中 ID 为 3 的评论重复出现了，所以我们只对它进行了一次计数。
表中 ID 为 2 的帖子有 ID 为 5 和 10 的两个评论。
ID 为 12 的帖子在表中没有评论。
表中 ID 为 6 的评论是对 ID 为 7 的已删除帖子的评论，因此我们将其忽略。

思路

自连接，首先这么连接：

```
select
        *
from Submissions s1
left join Submissions s2
on s1.sub_id = s2.parent_id
where s1.parent_id is null
```

这样的结果是

```
{"headers":
["sub_id", "parent_id", "sub_id", "parent_id"],
"values": [
[1, null, 9, 1],
[1, null, 4, 1],
[1, null, 3, 1],
[1, null, 3, 1],
[2, null, 10, 2],
[2, null, 5, 2],
[1, null, 9, 1],
[1, null, 4, 1],
[1, null, 3, 1],
[1, null, 3, 1],
[12, null, null, null]]}
```

然后按照s1.sub_id分组，count(distinct s2.sub_id, s2.parent_id)

代码

```
select
    s1.sub_id as post_id
    , ifnull(count(distinct s2.sub_id,s2.parent_id),0) as number_of_comments
from Submissions s1
left join Submissions s2
on s1.sub_id = s2.parent_id
where s1.parent_id is null
group by s1.sub_id
```

1251. 平均售价

难度：简单

Table: Prices

Column Name	Type
product_id	int
start_date	date
end_date	date
price	int

(product_id，start_date，end_date) 是 Prices 表的主键。
Prices 表的每一行表示的是某个产品在一段时期内的价格。
每个产品的对应时间段是不会重叠的，这也意味着同一个产品的价格时段不会出现交叉。

Table: UnitsSold

Column Name	Type
product_id	int
purchase_date	date
units	int

UnitsSold 表没有主键，它可能包含重复项。
UnitsSold 表的每一行表示的是每种产品的出售日期，单位和产品 id。

编写SQL查询以查找每种产品的平均售价。
average_price 应该四舍五入到小数点后两位。

查询结果格式如下例所示：
Prices table:

product_id	start_date	end_date	price
1	2019-02-17	2019-02-28	5
1	2019-03-01	2019-03-22	20
2	2019-02-01	2019-02-20	15
2	2019-02-21	2019-03-31	30

UnitsSold table:

product_id	purchase_date	units
1	2019-02-25	100
1	2019-03-01	15
2	2019-02-10	200
2	2019-03-22	30

Result table:

product_id	average_price
1	6.96
2	16.96

平均售价 = 产品总价 / 销售的产品数量。
产品 1 的平均售价 = ((100 * 5)+(15 * 20))/ 115 = 6.96
产品 2 的平均售价 = ((200 * 15)+(30 * 30))/ 230 = 16.96

代码

```
select
    b.product_id as product_id
    , round(
        sum(a.units * b.price)/sum(a.units)
      ,2) as average_price
from UnitsSold a
left join Prices b
on a.product_id = b.product_id
where a.purchase_date between b.start_date and b.end_date
group by b.product_id
```

1280. 学生们参加各科测试的次数

难度：简单
学生表: `Students`

Column Name	Type
student_id	int
student_name	varchar

主键为 student_id（学生ID），该表内的每一行都记录有学校一名学生的信息。

科目表: `Subjects`

Column Name	Type
subject_name	varchar

主键为 subject_name（科目名称），每一行记录学校的一门科目名称。

考试表: `Examinations`

Column Name	Type
student_id	int
subject_name	varchar

这张表压根没有主键，可能会有重复行。
学生表里的一个学生修读科目表里的每一门科目，而这张考试表的每一行记录就表示学生表里的某个学生参加了一次科目表里某门科目的测试。

要求写一段 SQL 语句，查询出每个学生参加每一门科目测试的次数，结果按 student_id 和 subject_name 排序。

查询结构格式如下所示：
`Students` table:

student_id	student_name
1	Alice
2	Bob
13	John
6	Alex

`Subjects` table:

subject_name
Math
Physics
Programming

`Examinations` table:

student_id	subject_name
1	Math
1	Physics
1	Programming
2	Programming
1	Physics
1	Math
13	Math
13	Programming
13	Physics
2	Math
1	Math

`Result` table:

student_id	student_name	subject_name	attended_exams
1	Alice	Math	3
1	Alice	Physics	2
1	Alice	Programming	1
2	Bob	Math	1
2	Bob	Physics	0
2	Bob	Programming	1
6	Alex	Math	0
6	Alex	Physics	0
6	Alex	Programming	0
13	John	Math	1
13	John	Physics	1
13	John	Programming	1

结果表需包含所有学生和所有科目（即便测试次数为0）：
Alice 参加了 3 次数学测试, 2 次物理测试，以及 1 次编程测试；
Bob 参加了 1 次数学测试, 1 次编程测试，没有参加物理测试；
Alex 啥测试都没参加；
John 参加了数学、物理、编程测试各 1 次。

代码

```
with cte as(
    select * from Students, Subjects
) # 注意，一定要先全都连起来，因为attended_exams结果有0
select
    a.student_id
    , a.student_name
    , a.subject_name
    , ifnull(count(b.subject_name),0) as attended_exams # 注意这里count的东西，如果是
count(*)那至少都是1
from cte as a
left join Examinations as b
on a.student_id = b.student_id and a.subject_name = b.subject_name
group by student_id, subject_name
order by student_id, subject_name
```

1294. 不同国家的天气类型

难度：简单

国家表：`Countries`

Column Name	Type
country_id	int
country_name	varchar

country_id 是这张表的主键。
该表的每行有 country_id 和 country_name 两列。

天气表：`Weather`

Column Name	Type
country_id	int
weather_state	varchar
day	date

(country_id, day) 是该表的复合主键。
该表的每一行记录了某个国家某一天的天气情况。

写一段 SQL 来找到表中每个国家在 2019 年 11 月的天气类型。
天气类型的定义如下：
当 weather_state 的平均值小于或等于15返回 Cold，
当 weather_state 的平均值大于或等于 25 返回 Hot，
否则返回 Warm。

你可以以任意顺序返回你的查询结果。

查询结果格式如下所示：
`Countries` table:

country_id	country_name
2	USA
3	Australia
7	Peru
5	China
8	Morocco
9	Spain

`Weather` table:

country_id	weather_state	day
2	15	2019-11-01
2	12	2019-10-28
2	12	2019-10-27
3	-2	2019-11-10
3	0	2019-11-11
3	3	2019-11-12
5	16	2019-11-07
5	18	2019-11-09
5	21	2019-11-23
7	25	2019-11-28
7	22	2019-12-01
7	20	2019-12-02
8	25	2019-11-05
8	27	2019-11-15
8	31	2019-11-25
9	7	2019-10-23
9	3	2019-12-23

`Result` table:

country_name	weather_type
USA	Cold
Austraila	Cold
Peru	Hot
China	Warm
Morocco	Hot

USA 11 月的平均 weather_state 为 (15) / 1 = 15 所以天气类型为 Cold。
Australia 11 月的平均 weather_state 为 (-2 + 0 + 3) / 3 = 0.333 所以天气类型为 Cold。
Peru 11 月的平均 weather_state 为 (25) / 1 = 25 所以天气类型为 Hot。
China 11 月的平均 weather_state 为 (16 + 18 + 21) / 3 = 18.333 所以天气类型为 Warm。
Morocco 11 月的平均 weather_state 为 (25 + 27 + 31) / 3 = 27.667 所以天气类型为 Hot。
我们并不知道 Spain 在 11 月的 weather_state 情况所以无需将他包含在结果中。

思路

主要是考察case when 和 DATE_FORMAT

```
with cte as(
    select
        b.country_name
        , avg(a.weather_state) as avg_weather
    from Weather as a
    left join Countries as b
    on a.country_id = b.country_id
    where DATE_FORMAT(day,'%Y-%m') = '2019-11'
    group by a.country_id
)
select
    country_name
    ,case
        when avg_weather <= 15 then 'Cold'
        when avg_weather >= 25 then 'Hot'
     else 'Warm' end
     as weather_type
from cte

# 也可以这样写
select
    country_name,
    case
        when avg(weather_state)<=15 then 'Cold'
        when avg(weather_state)>=25 then 'Hot'
        else 'Warm'
        end as weather_type
from Countries a
left join Weather b
on a.country_id = b.country_id
where day between '2019-11-01' and '2019-11-30'
group by a.country_id
```

585. 2016年的投资【中等】

难度：中等

写一个查询语句，将 2016 年 (TIV_2016) 所有成功投资的金额加起来，保留 2 位小数。

对于一个投保人，他在 2016 年成功投资的条件是：
他在 2015 年的投保额 (TIV_2015) 至少跟一个其他投保人在 2015 年的投保额相同。
他所在的城市必须与其他投保人都不同（也就是说维度和经度不能跟其他任何一个投保人完全相同）。

输入格式:
表 insurance 格式如下：

Column Name	Type
PID	INTEGER(11)
TIV_2015	NUMERIC(15,2)
TIV_2016	NUMERIC(15,2)
LAT	NUMERIC(5,2)
LON	NUMERIC(5,2)

PID 字段是投保人的投保编号， TIV_2015 是该投保人在2015年的总投保金额， TIV_2016 是该投保人在2016年的投保金额， LAT 是投保人所在城市的维度， LON 是投保人所在城市的经度。

样例输入

PID	TIV_2015	TIV_2016	LAT	LON
1	10	5	10	10
2	20	20	20	20
3	10	30	20	20
4	10	40	40	40

样例输出

TIV_2016
45.00

解释
就如最后一个投保人，第一个投保人同时满足两个条件：

1. 他在 2015 年的投保金额 TIV_2015 为 '10' ，与第三个和第四个投保人在 2015 年的投保金额相同。
2. 他所在城市的经纬度是独一无二的。

第二个投保人两个条件都不满足。他在 2015 年的投资 TIV_2015 与其他任何投保人都不相同。
且他所在城市的经纬度与第三个投保人相同。基于同样的原因，第三个投保人投资失败。

所以返回的结果是第一个投保人和最后一个投保人的 TIV_2016 之和，结果是 45 。

代码

```
# 方法1:用in筛选
# 1. 维度和经度不能跟其他任何一个投保人完全相同 -> where ... not in count(*)>=2
# 2. 在 2015 年的投保额 (TIV_2015) 至少跟一个其他投保人在 2015 年的投保额相同 -> group by
TIV_2015 having count(*)>=2
select
    round(ifnull(sum(TIV_2016),0),2) as TIV_2016
from insurance
where (LAT,LON) not in
(
    select
        LAT, LON
    from insurance
    group by LAT, LON
    having count(*) >= 2
)
and TIV_2015 in
(
    select
        TIV_2015
    from insurance
    group by TIV_2015
    having count(*) >= 2
)
# 这样写不行，因为最内层会直接去掉一部分数据，如果说有一个成功的投资者在2015年投资金额与被去掉的数据中
某人的投资金额相同，但又和没被去掉的人的投资金额不同，那么他就会被当作失败者，导致结果不一致。
# select
#     sum(TIV_2016) as TIV_2016
# from(
#     select
#         round(ifnull(sum(TIV_2016),0),2) as TIV_2016
#     from insurance
#     where (LAT,LON) not in
#     (
#         select
#             LAT, LON
#         from insurance
#         group by LAT, LON
#         having count(*) >= 2
#     )
#     group by TIV_2015
#     having count(*) >= 2
# ) as tmp
```

```
# 方法2:窗口函数
select
    round(ifnull(sum(TIV_2016),0),2) as TIV_2016
from(
    select
        a.*
        , count(*) over(partition by TIV_2015) as cnt1
        , count(*) over(partition by LAT, LON) as cnt2
    from insurance as a
) as tmp
where cnt1>1 and cnt2=1
```

608. 树节点【中等】

难度：中等
给定一个表 tree，id 是树节点的编号， p_id 是它父节点的 id 。

id	p_id
1	null
2	1
3	1
4	2
5	2

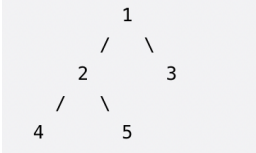
树中每个节点属于以下三种类型之一：
叶子：如果这个节点没有任何孩子节点。
根：如果这个节点是整棵树的根，即没有父节点。
内部节点：如果这个节点既不是叶子节点也不是根节点。

写一个查询语句，输出所有节点的编号和节点的类型，并将结果按照节点编号排序。上面样例的结果为：

id	Type
1	Root
2	Inner
3	Leaf
4	Leaf
5	Leaf

解释

节点 '1' 是根节点，因为它的父节点是 NULL ，同时它有孩子节点 '2' 和 '3' 。
节点 '2' 是内部节点，因为它有父节点 '1' ，也有孩子节点 '4' 和 '5' 。
节点 '3', '4' 和 '5' 都是叶子节点，因为它们都有父节点同时没有孩子节点。
样例中树的形态如下：



注意：如果树中只有一个节点，你只需要输出它的根属性。

代码

```
select
    id
    , case when p_id is null then 'Root'
           when id in (select distinct p_id from tree) and p_id is not null then
'Inner'
      else 'Leaf' end as type
from tree

# 如果这样写，从来不会返回Leaf
# select
#     id,
#     case when p_id is null then "Root"
#          when id not in (select p_id from tree) then "Leaf"
#          else "Inner"
#     end as Type
# from tree

# A not in B的原理是拿A表值与B表值做是否不等的比较，也就是a != b. 在sql中，null是缺失未知值而不是空值(详情请见MySQL reference).
# 当你判断任意值a != null时，官方说，"You cannot use arithmetic comparison operators such as =, <, or <> to test for NULL"，任何与null值的对比都将返回null. 因此返回结果为否,这点可以用代码 select if(1 = null, 'true', 'false')证实.
# 从上述原理可见，当询问 id not in (select p_id from tree)时，因为p_id有null值，返回结果全为false，于是跳到else的结果，返回值为inner. 所以在答案中,leaf结果从未彰显,全被inner取代.（我是没看懂但是记住了）
```

1097. 游戏玩法分析 V【困难】

难度：困难

`Activity` 活动记录表

Column Name	Type
player_id	int
device_id	int
event_date	date
games_played	int

（player_id，event_date）是此表的主键
这张表显示了某些游戏的玩家的活动情况
每一行表示一个玩家的记录，在某一天使用某个设备注销之前，登录并玩了很多游戏（可能是 0）

玩家的 安装日期 定义为该玩家的第一个登录日。

玩家的 第一天留存率 定义为：假定安装日期为 X 的玩家的数量为 N ，其中在 X 之后的一天重新登录的玩家数量为 M ，M/N 就是第一天留存率，四舍五入到小数点后两位。

编写一个 SQL 查询，报告所有安装日期、当天安装游戏的玩家数量和玩家的第一天留存率。

查询结果格式如下所示：

`Activity` 表：

player_id	device_id	event_date	games_played
1	2	2016-03-01	5
1	2	2016-03-02	6
2	3	2017-06-25	1
3	1	2016-03-01	0
3	4	2016-07-03	5

`Result` 表：

install_dt	installs	Day1_retention
2016-03-01	2	0.50
2017-06-25	1	0.00

玩家 1 和 3 在 2016-03-01 安装了游戏，但只有玩家 1 在 2016-03-02 重新登录，所以 2016-03-01 的第一天留存率是 1/2=0.50
玩家 2 在 2017-06-25 安装了游戏，但在 2017-06-26 没有重新登录，因此 2017-06-25 的第一天留存率为 0/1=0.00

思路

1. 找所有玩家的玩家的 **安装日期**；

2. 通过连接且`datediff(b.event_date, first_login) = 1`找到是否有第二天；
3. 分日期统计。

代码

```
with cte1 as(
    select
        player_id
        , min(event_date) as first_login
    from Activity
    group by player_id
),
cte2 as (
    select
        a.player_id
        , a.first_login
        , b.event_date
    from cte1 as a
    left join Activity as b
    on a.player_id = b.player_id
    and datediff(b.event_date, first_login) = 1
)
# 此时cte2的结果为：
#{"headers": ["player_id", "first_login", "event_date"], "values": [[1, "2016-03-01",
"2016-03-02"], [2, "2017-06-25", null], [3, "2016-03-01", null]]}
# 注意and datediff(b.event_date, first_login) = 1不能写成where
datediff(b.event_date,a.event_date) = 1，如果写成where，那么cte2的结果为：
# {"headers": ["player_id", "install_date", "rentention_date"], "values": [[1, "2016-
03-01", "2016-03-02"]]}
# 这样的原因是，on ... and... 的搭配，如果没有符合条件的，那么b.event_date会返回null，但是不会删除掉a表中的数据；如果是on...where...的搭配，那么如果x玩家没有第二天重新登陆的情况的话，a表对应的x的记录会直接被筛选没。
select
    first_login as install_dt
    , count(player_id) as installs
    , round(ifnull(count(event_date)/count(player_id),0),2) as Day1_retention
from cte2
group by first_login
```