DAY6

1303. 求团队人数

难度：简单

员工表：`Employee`

Column Name	Type
employee_id	int
team_id	int

employee_id 字段是这张表的主键，表中的每一行都包含每个员工的 ID 和他们所属的团队。
编写一个 SQL 查询，以求得每个员工所在团队的总人数。
查询结果中的顺序无特定要求。

查询结果格式示例如下：

`Employee` Table:

employee_id	team_id
1	8
2	8
3	8
4	7
5	9
6	9

`Result` table:

employee_id	team_size
1	3
2	3
3	3
4	1
5	2
6	2

ID 为 1、2、3 的员工是 team_id 为 8 的团队的成员，
ID 为 4 的员工是 team_id 为 7 的团队的成员，
ID 为 5、6 的员工是 team_id 为 9 的团队的成员。

代码

```
select
    employee_id
    , count(employee_id) over(partition by team_id) as team_size
from Employee
order by employee_id
```

614. Second Degree Follower【中等】

难度：中等

（中文版看不明白，所以看英文的吧）

Table: `Follow`

Column Name	Type
followee	varchar
follower	varchar

(followee, follower) is the primary key column for this table.
Each row of this table indicates that the user follower follows the user followee on a social network.
There will not be a user following themself.

A second-degree follower is a user who:
follows at least one user, and
is followed by at least one user.

Write an SQL query to report the second-degree users and the number of their followers.
Return the result table ordered by follower in alphabetical order.

The query result format is in the following example.
Input:
`Follow` table:

followee	follower
Alice	Bob
Bob	Cena
Bob	Donald
Donald	Edward

Output:

follower	num
Bob	2
Donald	1

Explanation:
User Bob has 2 followers. Bob is a second-degree follower because he follows Alice, so we include him in the result table.
User Donald has 1 follower. Donald is a second-degree follower because he follows Bob, so we include him in the result table.
User Alice has 1 follower. Alice is not a second-degree follower because she does not follow anyone, so we don not include her in the result table.

代码

```
select
    followee as follower
    , count(*) as num
from Follow
where followee in(
    select
        distinct follower
    from Follow
)
group by followee
order by follower
```

1045. 买下所有产品的客户【中等】

难度：中等

`Customer` 表：

Column Name	Type
customer_id	int
product_key	int

product_key 是 Customer 表的外键。
`Product` 表：

Column Name	Type
product_key	int

product_key 是这张表的主键。

写一条 SQL 查询语句，从 Customer 表中查询购买了 Product 表中所有产品的客户的 id。

示例：
`Customer` 表：

customer_id	product_key
1	5
2	6
3	5
3	6
1	6

`Product` 表：

product_key
5
6

`Result` 表：

customer_id
1
3

购买了所有产品（5 和 6）的客户的 id 是 1 和 3 。

思路

1. 按照customer_id分组，且每个customer_id买的distinct product_key的数量等于Product中product_key的数量
2. 为了保证买的就是来自于Product表中的，还得需要product_key in (select product_key from Product)

代码

```
select
    customer_id
from Customer
where product_key in (select product_key from Product)
group by customer_id
having count(distinct product_key) = (select count(product_key) from Product)
# 不要写错成
# select
#     customer_id
# from Customer
# where product_key in (select product_key from Product)
# and count(distinct product_key) = (select count(product_key) from Product)
# group by customer_id
```

1070. 产品销售分析 III【中等】

难度：中等

销售表 `Sales`：

Column Name	Type
sale_id	int
product_id	int
year	int
quantity	int
price	int

(sale_id, year) 是这张表的主键。
product_id 是产品表的外键。
这张表的每一行都表示：编号 product_id 的产品在某一年的销售额。
请注意，价格是按每单位计的。

产品表 `Product`：

Column Name	Type
product_id	int
product_name	varchar

product_id 是这张表的主键。
这张表的每一行都标识：每个产品的 id 和 产品名称。

编写一个 SQL 查询，选出每个销售产品 第一年 销售的 产品 id、年份、数量 和 价格。
结果表中的条目可以按 任意顺序 排列。

查询结果格式如下例所示：
输入：
`Sales` 表：

sale_id	product_id	year	quantity	price
1	100	2008	10	5000
2	100	2009	12	5000
7	200	2011	15	9000

`Product` 表：

product_id	product_name
100	Nokia
200	Apple
300	Samsung

输出：

product_id	first_year	quantity	price
100	2008	10	5000
200	2011	15	9000

思路

子查询匹配

代码

```
select
    product_id
    , year as first_year
    , quantity
    , price
from Sales
where (product_id, year) in(
    select
        product_id
        , min(year)
    from Sales
    group by product_id
)
```

1112. 每位学生的最高成绩【中等】

难度：中等

表：`Enrollments`

Column Name	Type
student_id	int
course_id	int
grade	int

(student_id, course_id) 是该表的主键。

编写一个 SQL 查询，查询每位学生获得的最高成绩和它所对应的科目，若科目成绩并列，取 course_id 最小的一门。查询结果需按 student_id 增序进行排序。
以 任意顺序 返回结果表。

查询结果格式如下所示。
输入：
`Enrollments` 表：

student_id	course_id	grade
2	2	95
2	3	95
1	1	90
1	2	99
3	1	80
3	2	75
3	3	82

输出：

student_id	course_id	grade
1	2	99
2	2	95
3	3	82

代码

```
# 方法1：
select
    student_id
    , min(course_id) as course_id  # 用于找到课程id最小
    , grade
from Enrollments
where (student_id, grade) in(
    select
        student_id
        , max(grade)
    from Enrollments
    group by student_id
)
group by student_id # 用于找到课程id最小
order by student_id

# 方法2：（窗口函数）
select
    student_id
    , course_id
    , grade
from(
```

```
    select
    a.*,
    row_number() over(partition by student_id order by grade desc, course_id asc) as
rnk
    from Enrollments a
) as tmp
where rnk = 1
```

1126. 查询活跃业务【中等】

难度：中等

事件表：`Events`

Column Name	Type
business_id	int
event_type	varchar
occurences	int

此表的主键是 (business_id, event_type)。
表中的每一行记录了某种类型的事件在某些业务中多次发生的信息。

写一段 SQL 来查询所有活跃的业务。
如果一个业务的某个事件类型的发生次数大于此事件类型在所有业务中的平均发生次数，并且该业务至少有两个这样的事件类型，那么该业务就可被看做是活跃业务。

查询结果格式如下所示：
`Events` table:

business_id	event_type	occurences
1	reviews	7
3	reviews	3
1	ads	11
2	ads	7
3	ads	6
1	page views	3
2	page views	12

结果表

business_id
1

'reviews'、 'ads' 和 'page views' 的总平均发生次数分别是 (7+3)/2=5, (11+7+6)/3=8, (3+12)/2=7.5。
id 为 1 的业务有 7 个 'reviews' 事件（大于 5）和 11 个 'ads' 事件（大于 8），所以它是活跃业务。

代码

```
# 方法1:窗口函数
select
    business_id
from(
    select
        a.*
        , avg(occurences) over(partition by event_type) as avg_oc
    from Events as a
) as tmp
where occurences>avg_oc
group by business_id # 这行是关键
having count(*)>1 # 这行也是关键

# 方法2:
# 至少有两个
with cte as(
    select
        event_type
        , avg(occurences) as avg_occurences
    from Events
    group by event_type
)
select
    business_id
from Events a
left join cte b
on a.event_type = b.event_type
where a.occurences > b.avg_occurences # 这里如果把where改成and，就得把left join改成join，但是瞬时没想明白为什么
group by business_id
having count(*) >= 2
```

1193. 每月交易 I【中等】

难度：中等

Table: Transactions

Column Name	Type
id	int
country	varchar
state	enum
amount	int
trans_date	date

id 是这个表的主键。
该表包含有关传入事务的信息。
state 列类型为 “[”批准“，”拒绝“] 之一。

编写一个 sql 查询来查找每个月和每个国家/地区的事务数及其总金额、已批准的事务数及其总金额。
以 任意顺序 返回结果表。

查询结果格式如下所示。
输入：
Transactions table:

id	country	state	amount	trans_date
121	US	approved	1000	2018-12-18
122	US	declined	2000	2018-12-19
123	US	approved	2000	2019-01-01
124	DE	approved	2000	2019-01-07

输出：

month	country	trans_count	approved_count	trans_total_amount	approved_total_amount
2018-12	US	2	1	3000	1000
2019-01	US	1	1	2000	2000
2019-01	DE	1	1	2000	2000

代码

```sql
select
    DATE_FORMAT(trans_date, '%Y-%m') as month
    , country
    , count(*) as trans_count
    , sum(state = 'approved') as approved_count
    # , count(if(state = 'approved',1,null)) as approved_count # 注意要是这么写的话是null不是0
    , sum(amount) as trans_total_amount
    , sum(if(state='approved',amount,0)) approved_total_amount
from Transactions
group by country, DATE_FORMAT(trans_date, '%Y-%m')
```

1212. 查询球队积分【中等】

难度：中等

表: `Teams`

Column Name	Type
team_id	int
team_name	varchar

此表的主键是 team_id。
表中的每一行都代表一支独立足球队。

表: `Matches`

Column Name	Type
match_id	int
host_team	int
guest_team	int
host_goals	int
guest_goals	int

此表的主键是 match_id。
表中的每一行都代表一场已结束的比赛。
比赛的主客队分别由它们自己的 id 表示，他们的进球由 host_goals 和 guest_goals 分别表示。

您希望在所有比赛之后计算所有球队的比分。积分奖励方式如下:
如果球队赢了比赛(即比对手进更多的球)，就得 3 分。
如果双方打成平手(即，与对方得分相同)，则得 1 分。
如果球队输掉了比赛(例如，比对手少进球)，就 不得分 。
写出一条SQL语句以查询每个队的 team_id，team_name 和 num_points。

返回的结果根据 num_points 降序排序，如果有两队积分相同，那么这两队按 team_id 升序排序。

查询结果格式如下。
输入：
`Teams` table:

team_id	team_name
10	Leetcode FC
20	NewYork FC
30	Atlanta FC
40	Chicago FC
50	Toronto FC

`Matches` table:

match_id	host_team	guest_team	host_goals	guest_goals
1	10	20	3	0
2	30	10	2	2
3	10	50	5	1
4	20	30	1	0
5	50	30	1	0

输出：

team_id	team_name	num_points
10	Leetcode FC	7
20	NewYork FC	3
50	Toronto FC	3
30	Atlanta FC	1
40	Chicago FC	0

思路

为了方便group by，无论主客场都通过union all的方式归到一列去

代码

```
with cte as(
    select
```

```
        host_team as a_team
        , guest_team as b_team
        , host_goals as a_score
        , guest_goals as b_score
    from Matches
    union all
    select
        guest_team as a_team
        , host_team as b_team
        , guest_goals as a_score
        , host_goals as b_score
    from Matches
)

select
    b.team_id
    , b.team_name
    , ifnull(sum(point),0) as num_points
from(
    select
        tmp.a_team as team_id
        , case when a_score>b_score then 3
            when a_score=b_score then 1
        else 0 end as point
    from cte as tmp
) as a
right join Teams as b
on b.team_id = a.team_id
group by team_id
order by num_points desc, team_id asc

# 也可以写成:

select
    a.team_id
    , a.team_name
    , sum(
        case
            when a_score>b_score then 3
            when a_score=b_score then 1
        else 0 end
    ) as num_points
from Teams as a
left join cte as b
on a.team_id = b.a_team
group by a.team_id
order by num_points desc, a.team_id
```

1264. 页面推荐【中等】

难度：中等

朋友关系列表：`Friendship`

Column Name	Type
user1_id	int
user2_id	int

这张表的主键是 (user1_id, user2_id)。
这张表的每一行代表着 user1_id 和 user2_id 之间存在着朋友关系。

喜欢列表：`Likes`

Column Name	Type
user_id	int
page_id	int

这张表的主键是 (user_id, page_id)。
这张表的每一行代表着 user_id 喜欢 page_id。

写一段 SQL 向user_id = 1 的用户，推荐其朋友们喜欢的页面。不要推荐该用户已经喜欢的页面。
返回的结果中不应当包含重复项。

返回结果的格式如下例所示。
输入：
`Friendship` table:

user1_id	user2_id
1	2
1	3
1	4
2	3
2	4
2	5
6	1

`Likes` table:

user_id	page_id
1	88
2	23
3	24
4	56
5	11
6	33
2	77
3	77
6	88

输出：

recommended_page
23
24
56
33
77

解释：
用户1 同 用户2, 3, 4, 6 是朋友关系。
推荐页面为： 页面23 来自于 用户2, 页面24 来自于 用户3, 页面56 来自于 用户3 以及 页面33 来自于 用户6。
页面77 同时被 用户2 和 用户3 推荐。
页面88 没有被推荐，因为 用户1 已经喜欢了它。

思路

和1212题一样，首先union一下方便找到用户1的朋友，然后用where... in (not in)...结构确定推荐页面。

代码

```
with cte as(
    select
        user1_id as id1
        , user2_id as id2
    from Friendship
    union all
```

```
    select
        user2_id as id2
        , user1_id as id1
    from Friendship
)

select
    distinct page_id as recommended_page
from Likes
where user_id in (
    select
        id2
    from cte
    where id1 = 1
)
and page_id not in(
    select
        page_id
    from Likes
    where user_id = 1
)
```

618. 学生地理信息报告【困难】

难度：困难

表： `student`

Column Name	Type
name	varchar
continent	varchar

该表没有主键。它可能包含重复的行。
该表的每一行表示学生的名字和他们来自的大陆。

一所学校有来自亚洲、欧洲和美洲的学生。

写一个查询语句实现对大洲（continent）列的 透视表 操作，使得每个学生按照姓名的字母顺序依次排列在对应的大洲下面。输出的标题应依次为美洲（America）、亚洲（Asia）和欧洲（Europe）。

测试用例的生成使得来自美国的学生人数不少于亚洲或欧洲的学生人数。

查询结果格式如下所示。
输入:
`Student` table:

name	continent
Jane	America
Pascal	Europe
Xi	Asia
Jack	America

输出:

America	Asia	Europe
Jack	Xi	Pascal
Jane	null	null

进阶：如果不能确定哪个大洲的学生数最多，你可以写出一个查询去生成上述学生报告吗?

思路

先按照continent使用窗口函数进行排序，排序行即可以作为分组的依据不影响其他字段的使用，分组后用if决定写name还是null，用as...作为列名。

为什么需要使用max/min？这里牵扯到了使用group by后，系统留存的虚拟表格，系统默认我们只对group by筛选出的第一行进行判断而舍弃了剩下的数据行。正确的做法是加上聚合函数。具体请参考：
https://leetcode.cn/problems/students-report-by-geography/solution/zong-jie-ge-lei-biao-ge-ge-shi-hua-wen-t-tl4e/
和
https://leetcode.cn/problems/reformat-department-table/solution/group-byben-zhi-lun-by-loverxp-7mgy/

代码

```
select
    max(if(continent = 'America', name, null)) as America
    , max(if(continent = 'Asia', name, null)) as Asia
    , max(if(continent = 'Europe', name, null)) as Europe
    #     max(case when continent = 'America' then name else null end) America,
    #     max(case when continent = 'Asia' then name else null end) Asia,
    #     max(case when continent = 'Europe' then name else null end) Europe
from
    (select
        name,
        continent,
        row_number()over(partition by continent order by name) cur_rank
    from
        student)t
group by cur_rank
```