

DAY 3

- DAY 3
1050. 合作过至少三次的演员和导演

1082. 销售分析 I

1083. 销售分析 II

1084. 销售分析III

580. 统计各专业学生人数【中等】

603. 连续空余座位【简单难度的中等题】

610. 判断三角形

613. 直线上的最近距离

612. 平面上的最近距离【中等】

1285. 找到连续区间的开始和结束数字【中等】

180. 连续出现的数字【中等】

1225. 报告系统状态的连续日期【困难】

601. 体育馆的人流量【困难】

1050. 合作过至少三次的演员和导演

难度：简单

ActorDirector 表：

Column Name	Type
actor_id	int
director_id	int
timestamp	int

timestamp 是这张表的主键.

写一条SQL查询语句获取合作过至少三次的演员和导演的 id 对 (actor_id, director_id)

示例：

ActorDirector 表：

actor_id	director_id	timestamp
1	1	0
1	1	1
1	1	2
1	2	3
1	2	4
2	1	5
2	1	6

Result 表：

actor_id	director_id
1	1

唯一的 id 对是 (1, 1)，他们恰好合作了 3 次。

代码

```
# 白给题
select
    actor_id
    , director_id
from ActorDirector
group by actor_id, director_id
having count(timestamp)>=3
```

1082. 销售分析 I

产品表：Product

Column Name	Type
product_id	int
product_name	varchar
unit_price	int

product_id 是这个表的主键。
该表的每一行显示每个产品的名称和价格。

销售表：Sales

Column Name	Type
seller_id	int
product_id	int
buyer_id	int
sale_date	date
quantity	int
price	int

这个表没有主键，它可以有重复的行。
product_id 是 Product 表的外键。
该表的每一行包含关于一个销售的一些信息。
编写一个 SQL 查询，查询总销售额最高的销售者，如果有并列的，就都展示出来。
以 任意顺序 返回结果表。
查询结果格式如下所示。

示例 1:

输入:

Product 表:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales 表:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

输出:

seller_id
1

思路

和DAY2那四道题一样

代码

```
# 方法1:
select
    seller_id
from(
    select
        seller_id
        , dense_rank() over(order by sum(price) desc) as rnk
    from Sales
    group by seller_id

) as tmp
where rnk = 1

# 方法2:
select
    seller_id
from Sales
group by seller_id
having sum(price)>=
all(
    select
        sum(price)
    from Sales
    group by seller_id
)
```

1083. 销售分析 II

难度：简单

编写一个 SQL 查询，查询购买了 S8 手机却没有购买 iPhone 的买家。注意这里 S8 和 iPhone 是 Product 表中的产品。

查询结果格式如下图表示：

Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	1	3	2019-06-02	1	800
3	3	3	2019-05-13	2	2800

Result table:

buyer_id
1

id 为 1 的买家购买了一部 S8，但是却没有购买 iPhone，而 id 为 3 的买家却同时购买了这 2 部手机。

思路

- 方法1：连接表，选择需要的字段，然后 `product_name = 'S8'` and `buyer_id not in` 买过iPhone的 `buyer_id`
- 方法2：连接表

```
group by buyer_id
having sum(product_name = 'iPhone') = 0
and sum(product_name = 'S8') >= 1
```

代码

```
# 方法1:
with cte as(
    select
        b.product_name
        , a.buyer_id
    from Sales a
    left join Product b
    on a.product_id = b.product_id
)

select
    distinct buyer_id # 注意这里一定要加distinct!
from cte
where product_name = 'S8'
and buyer_id not in
(
    select
        buyer_id
    from cte
    where product_name = 'iPhone'
)
```

```
# 方法2:
select
    distinct buyer_id
from Sales as a
left join Product as b
on a.product_id = b.product_id
group by buyer_id
having sum(product_name = 'iPhone') = 0
and sum(product_name = 'S8') >= 1 # sum(if(product_name='S8',1,0))>0
```

1084. 销售分析III

难度：简单

编写一个SQL查询，报告2019年春季才售出的产品。即仅在2019-01-01至2019-03-31（含）之间出售的商品。

以任意顺序返回结果表。

查询结果格式如下所示。

示例 1:

输入：

Product table:

product_id	product_name	unit_price
1	S8	1000
2	G4	800
3	iPhone	1400

Sales table:

seller_id	product_id	buyer_id	sale_date	quantity	price
1	1	1	2019-01-21	2	2000
1	2	2	2019-02-17	1	800
2	2	3	2019-06-02	1	800
3	3	4	2019-05-13	2	2800

输出：

product_id	product_name
1	S8

解释:

id为1的产品仅在2019年春季销售。
id为2的产品在2019年春季销售，但也在2019年春季之后销售。
id 3的产品在2019年春季之后销售。
我们只退回产品1，因为它是2019年春季才销售的产品。

思路

连接，选需要的字段，按照product_id分组，having用于筛选符合条件的日期

代码

```
select
    a.product_id
    ,b.product_name
from Sales a
left join Product b
on a.product_id = b.product_id
group by a.product_id
having sum(sale_date<'2019-01-01') = 0
and sum(sale_date>'2019-03-31') = 0
# 或者写成
# min(s.sale_date) >= '2019-01-01' and max(s.sale_date) <= '2019-03-31'
# 注意一定是>=和<=, 不等号和等号的位置不能反, 特别是<=,不等号在前
```

580. 统计各专业学生人数【中等】

难度：中等

表: Student

Column Name	Type
student_id	int
student_name	varchar
gender	varchar
dept_id	int

Student_id是该表的主键。
dept_id是Department表中dept_id的外键。
该表的每一行都表示学生的姓名、性别和所属系的id。

表: Department

Column Name	Type
dept_id	int
dept_name	varchar

Dept_id是该表的主键。

该表的每一行包含一个部门的id和名称。

编写一个SQL查询，为 **Department** 表中的所有部门(甚至是没有当前学生的部门)报告各自的部门名称和每个部门的学生人数。

按 **student_number** 降序 返回结果表。如果是平局，则按 **dept_name** 的 字母顺序 排序。

查询结果格式如下所示。

示例 1:

输入:

Student 表:

student_id	student_name	gender	dept_id
1	Jack	M	1
2	Jane	F	1
3	Mark	M	2

Department 表:

dept_id	dept_name
1	Engineering
2	Science
3	Law

输出:

dept_name	student_number
Engineering	2
Science	1
Law	0

代码


```
# 基本是白给题，注意细节，一个是group by a.dept_id的a.；另一个是排序
select
    dept_name
    , ifnull(count(student_id),0) as student_number
from Department a
left join Student b
on a.dept_id = b.dept_id
group by a.dept_id
order by student_number desc, dept_name asc
```

603. 连续空余座位【简单难度的中等题】

难度：简单

表: Cinema

Column Name	Type
seat_id	int
free	bool

Seat_id是该表的自动递增主键列。
该表的每一行表示第i个座位是否空闲。1表示空闲，0表示被占用。
编写一个SQL查询来报告电影院所有连续可用的座位。
返回按 seat_id 升序排序 的结果表。
测试用例的生成使得两个以上的座位连续可用。
查询结果格式如下所示。

示例 1:

输入:

Cinema 表:

seat_id	free
1	1
2	0
3	1
4	1
5	1

输出:

seat_id
3
4
5

思路

自连接的方式找到连续座位

代码

```
# 连续和self join一般都有关系，这题要会
# seat_id 相差1，说明座位连续
# 再加上free = 1，说明座位连续可用
select
    distinct a.seat_id
from Cinema a
join Cinema b
where a.seat_id <> b.seat_id
and a.free = 1
and b.free = 1
and abs(a.seat_id - b.seat_id) = 1
order by a.seat_id
```

610. 判断三角形

难度：简单

表: Triangle

Column Name	Type
x	int
y	int
z	int

(x, y, z)是该表的主键列。
该表的每一行包含三个线段的长度。
写一个SQL查询，每三个线段报告它们是否可以形成一个三角形。
以 任意顺序 返回结果表。
查询结果格式如下所示。
示例 1:
输入:
Triangle 表:

x	y	z
13	15	30
10	20	15

输出:

x	y	z	triangle
13	15	30	No
10	20	15	Yes

知识点

```
case when
    cond1 then a
    cond2 then b
else c
end
```

代码

```
select
    a.*,
    case when
        a.x+a.y>a.z and a.x+a.z>a.y and a.y+a.z>a.x then 'Yes'
    else 'No'
    end as triangle
from Triangle as a
```

613. 直线上的最近距离

难度：简单

表 point 保存了一些点在 x 轴上的坐标，这些坐标都是整数。

写一个查询语句，找到这些点中最近两个点之间的距离。

x
-1
0
2

最近距离显然是 '1'，是点 '-1' 和 '0' 之间的距离。所以输出应该如下：

shortest
1

注意：每个点都与其他点坐标不同，表 table 不会有重复坐标出现。

进阶：如果这些点在 x 轴上从左到右都有一个编号，输出结果时需要输出最近点对的编号呢？

代码

```
select
    min(abs(p1.x-p2.x)) as shortest
from
    point as p1,
    point as p2
where p1.x <> p2.x
```

612. 平面上的最近距离【中等】

难度：中等

Point2D 表：

Column Name	Type
x	int
y	int

(x, y) 是这张表的主键

这张表的每一行表示 X-Y 平面上一个点的位置

p1(x1, y1) 和 p2(x2, y2) 这两点之间的距离是 $\sqrt{(x2 - x1)^2 + (y2 - y1)^2}$ 。

请你写一个 SQL 查询报告 Point2D 表中任意两点之间的最短距离。保留 2 位小数。

查询结果格式如下例所示。

示例：

输入：

Point2D table:

x	y
-1	-1
0	0
-1	-2

输出：

shortest
1.00

解释：最短距离是 1.00 ，从点 (-1, -1) 到点 (-1, 2) 。

代码

```
select
    round(
        min(
            sqrt(power(p1.x-p2.x,2)+power(p1.y-p2.y,2))
        )
    ,2) as shortest
from Point2D p1, Point2D p2
where (p1.x,p1.y)<>(p2.x,p2.y) # 保险起见写成这样
# p1.x<>p2.x and p1.y <> p2.y # 错误
# p1.x<>p2.x or p1.y <> p2.y # 正确
# !(p1.x = p2.x and p1.y = p2.y) # 正确
```

1285. 找到连续区间的开始和结束数字【中等】

难度：中等

表：Logs

Column Name	Type
log_id	int

id 是上表的主键。

上表的每一行包含日志表中的一个 ID。

后来一些 ID 从 Logs 表中删除。编写一个 SQL 查询得到 Logs 表中的连续区间的开始数字和结束数字。

将查询表按照 start_id 排序。

查询结果格式如下面的例子。

示例 1：

输入：

Logs 表：

log_id
1
2
3
7
8
10

输出：

start_id	end_id
1	3
7	8
10	10

解释：

结果表应包含 Logs 表中的所有区间。

从 1 到 3 在表中。

从 4 到 6 不在表中。

从 7 到 8 在表中。

9 不在表中。

10 在表中。

思路

log_id	row_number	log_id - row_number
1	1	0
2	2	0
3	3	0
7	4	3
8	5	3
10	6	4

技巧在于：group by log_id - row_number，然后取每组的min为 start_id，max为 end_id

代码

```

select
    min(log_id) as start_id
    , max(log_id) as end_id
from
(
    select
        a.log_id
        , a.log_id - row_number() over(order by log_id) as rnk
    from Logs as a
) as tmp
group by rnk

```

180. 连续出现的数字【中等】

难度：中等

表：Logs

Column Name	Type
id	int
num	varchar

id 是这个表的主键。

编写一个 SQL 查询，查找所有至少连续出现三次的数字。

返回的结果表中的数据可以按 任意顺序 排列。

查询结果格式如下面的例子所示：

示例 1:

输入：

Logs 表：

Id	Num
1	1
2	1
3	1
4	2
5	1
6	2
7	2

输出：

Result 表：

ConsecutiveNums
1

解释：1 是唯一连续出现至少三次的数字。

思路

Id	Num	a: row_number() over(order by a.id)	b: row_number() over(partition by a.Num order by a.Id)	a - b
1	1	1	1	0
2	1	2	2	0
3	1	3	3	0
4	2	4	1	3
5	1	5	4	1
6	2	6	2	4
7	2	7	3	4

按照num和a-b分组，然后找到count(*)>3的

代码

```
select
    distinct num as ConsecutiveNums # 注意distinct
from
(
    select
        a.*
        , row_number() over(order by a.id)
        - row_number() over(partition by a.num order by a.id) as rnk
    from Logs as a
)as tmp
group by num, rnk # 注意按照这两个分组
having count(*)>=3
```

1225. 报告系统状态的连续日期【困难】

难度：困难

Table: Failed

Column Name	Type
fail_date	date

该表主键为 fail_date。
该表包含失败任务的天数。

Table: Succeeded

Column Name	Type
success_date	date

该表主键为 success_date。
该表包含成功任务的天数。

系统 每天 运行一个任务。每个任务都独立于先前的任务。任务的状态可以是失败或是成功。

编写一个 SQL 查询 2019-01-01 到 2019-12-31 期间任务连续同状态 period_state 的起止日期 (start_date 和 end_date) 。即如果任务失败了，就是失败状态的起止日期，如果任务成功了，就是成功状态的起止日期。
最后结果按照起始日期 start_date 排序
查询结果样例如下所示:

Failed table:

fail_date
2018-12-28
2018-12-29
2019-01-04
2019-01-05

Succeeded table:

success_date
2018-12-30
2018-12-31
2019-01-01
2019-01-02
2019-01-03
2019-01-06

Result table:

period_state	start_date	end_date
succeeded	2019-01-01	2019-01-03
failed	2019-01-04	2019-01-05
succeeded	2019-01-06	2019-01-06

结果忽略了 2018 年的记录，因为我们只关心从 2019-01-01 到 2019-12-31 的记录
 从 2019-01-01 到 2019-01-03 所有任务成功，系统状态为 "succeeded"。
 从 2019-01-04 到 2019-01-05 所有任务失败，系统状态为 "failed"。
 从 2019-01-06 到 2019-01-06 所有任务成功，系统状态为 "succeeded"。

思路

先使用union将两个表连接，然后参考180题的思路。注意筛选日期。

代码

```

with cte as(
select
    fail_date as date,
    "failed" as period_state
from failed
union
select
    success_date,
    "succeeded"
from succeeded
order by 1)

select
    period_state,
    min(date) as start_date,
    max(date) as end_date
from
(select
    date,
    period_state,
    row_number() over(order by date) -
    row_number() over(partition by period_state order by date) as rnk_diff
from cte
where date between '2019-01-01' and '2019-12-31') tmp
group by period_state, rnk_diff
order by start_date
  
```

601. 体育馆的人流量【困难】

难度：困难

表：Stadium

Column Name	Type
id	int
visit_date	date
people	int

visit_date 是表的主键
每日人流量信息被记录在这三列信息中：序号 (id)、日期 (visit_date)、人流量 (people)
每天只有一行记录，日期随着 id 的增加而增加
编写一个 SQL 查询以找出每行的人数大于或等于 100 且 id 连续的三行或更多行记录。
返回按 visit_date 升序排列 的结果表。
查询结果格式如下所示。

示例 1:

输入:

Stadium 表:

id	visit_date	people
1	2017-01-01	10
2	2017-01-02	109
3	2017-01-03	150
4	2017-01-04	99
5	2017-01-05	145
6	2017-01-06	1455
7	2017-01-07	199
8	2017-01-09	188

输出:

id	visit_date	people
5	2017-01-05	145
6	2017-01-06	1455
7	2017-01-07	199
8	2017-01-09	188

解释：

id 为 5、6、7、8 的四行 id 连续，并且每行都有 ≥ 100 的人数记录。

请注意，即使第 7 行和第 8 行的 visit_date 不是连续的，输出也应当包含第 8 行，因为我们只需要考虑 id 连续的记录。

不输出 id 为 2 和 3 的行，因为至少需要三条 id 连续的记录。

思路

id	visit_date	people	row_number() over(order by id)	id-row_number() over(order by id)
2	2017-01-02	109	1	1
3	2017-01-03	150	2	1
5	2017-01-05	145	3	2
6	2017-01-06	1455	4	2
7	2017-01-07	199	5	2
8	2017-01-09	188	6	2

按照 id-row_number() over(order by id) 分组，count大于等于3的取出来，但是这里要格外注意不能直接group by，详见代码。

代码

```
# 一开始的错误写法：
# select
#     id
#     , visit_date
#     , people
# from(
#     select
#         a.*
#         , id - row_number() over(order by id) as diff
#     from Stadium a
#     where people>=100
# ) as tmp
# group by diff # 错在于这里不能group, group直接行会少，只保留第一行了
# having count(id)>=3
# order by visit_date asc
```

正确写法:

```
with cte as(  
    select  
        a.*  
        , id - row_number() over(order by id) as diff  
    from Stadium a  
    where people>=100  
)  
  
select  
    id  
    , visit_date  
    , people  
from cte  
where diff in  
(  
    select  
        diff  
    from cte  
    group by diff  
    having count(*)>=3  
)
```