DAY7

176. 第二高的薪水【中等】

难度：中等

`Employee` 表：

Column Name	Type
id	int
salary	int

id 是这个表的主键。
表的每一行包含员工的工资信息。

编写一个 SQL 查询，获取并返回 Employee 表中第二高的薪水 。如果不存在第二高的薪水，查询应该返回 null 。

查询结果如下例所示。
示例1:
输入：
`Employee` 表：

id	salary
1	100
2	200
3	300

输出：

SecondHighestSalary
200

示例 2：
输入：
`Employee` 表：

id	salary
1	100

输出：

SecondHighestSalary
null

代码

```
# 这样不行
# select
#     ifnull(salary,null) as SecondHighestSalary
# from(
#     select
#         a.*
#         , dense_rank() over(order by salary desc) as rnk
#         # 不能是row_number，因为会有并列第一
#         # 也不能是rank，因为会变成113
#     from Employee a
# ) as tmp
# where rnk = 2

# 方法1：
# 这样可以，但是对于null的情况需要外层嵌套一下select选，我也不知道为什么不能ifnull
select(
    select
        distinct salary
    from(
        select
            a.*
            , dense_rank() over(order by salary desc) as rnk
            # 不能是row_number，因为会有并列第一
            # 也不能是rank，因为会变成113
        from Employee a
        ) as tmp
    where rnk = 2) as SecondHighestSalary
```

```
# 方法2：子查询匹配：
select
    max(salary) as SecondHighestSalary
from Employee
where salary <
# where salary not in
(
    select
        max(salary)
    from Employee
)
```

1164. 指定日期的产品价格【中等】

难度：中等

产品数据表: `Products`

Column Name	Type
product_id	int
new_price	int
change_date	date

这张表的主键是 (product_id, change_date)。
这张表的每一行分别记录了 某产品 在某个日期 更改后 的新价格。

写一段 SQL来查找在 2019-08-16 时全部产品的价格，假设所有产品在修改前的价格都是 10 。
以 任意顺序 返回结果表。

查询结果格式如下例所示。
输入：
`Products` 表:

product_id	new_price	change_date
1	20	2019-08-14
2	50	2019-08-14
1	30	2019-08-15
1	35	2019-08-16
2	65	2019-08-17
3	20	2019-08-18

输出：

product_id	price
2	50
1	35
3	10

思路

1. cte1用来的得到所有的distinct product_id；
2. cte2通过窗口函数得到规定日期内最新的价格；
3. cte1左连接cte2，得到所有商品最新价格，用ifnull来解决没有在规定日期内更新价格的商品（价格为10）。

代码

```
# 非窗口函数写法，还没写出来
# select
#     product_id
#     , new_price as price # 这么写，这里并不是最大日期对应的new_price
#     , max(change_date)
# from Products
# where change_date <= '2019-08-16'
# group by product_id

# 这么写也不行，这样筛出来是这样的：{"headers": ["product_id", "price"], "values": [[1, 20],
[2, 50]]}
# select
#     product_id
#     , new_price as price
# from Products
# where change_date <= '2019-08-16'
# group by product_id
# having max(change_date)

# 窗口函数写法
with cte1 as(
    select distinct product_id from Products
),
cte2 as(
select
    product_id
    , price
from(
    select
        product_id
        , new_price as price
        , change_date
        , dense_rank() over(partition by product_id order by change_date desc) as rnk
    from Products
```

```
    where change_date <= '2019-08-16'
) as tmp
where rnk = 1
)

select
    a.product_id
    , ifnull(price, 10) as price
from cte1 as a
left join cte2 as b
on a.product_id = b.product_id

# 如果这样写:
with cte1 as(
    select
        distinct product_id
    from Products
),
cte2 as(
    select
        a.*
        , row_number() over(partition by product_id order by change_date desc) as rnk
    from Products as a
    where change_date <= '2019-08-16 '
)
select
    c1.product_id
    , ifnull(c2.new_price,10) as price
from cte1 as c1
left join cte2 as c2
on c1.product_id = c2.product_id
where c2.rnk = 1 or c2.rnk is null # 注意这里要加c2.rnk is null
```

1204. 最后一个能进入电梯的人【中等】

难度：中等

表: `Queue`

Column Name	Type
person_id	int
person_name	varchar
weight	int
turn	int

person_id 是这个表的主键。
该表展示了所有等待电梯的人的信息。
表中 person_id 和 turn 列将包含从 1 到 n 的所有数字，其中 n 是表中的行数。

有一群人在等着上公共汽车。然而，巴士有1000 公斤的重量限制，所以可能会有一些人不能上。
写一条 SQL 查询语句查找 最后一个 能进入电梯且不超过重量限制的 person_name 。题目确保队列中第一位的人可以进入电梯，不会超重。

查询结果如下所示。
输入：
`Queue` 表

person_id	person_name	weight	turn
5	George Washington	250	1
3	John Adams	350	2
6	Thomas Jefferson	400	3
2	Will Johnliams	200	4
4	Thomas Jefferson	175	5
1	James Elephant	500	6

输出：

person_name
Thomas Jefferson

解释：
为了简化，Queue 表按 turn 列由小到大排序。
上例中 George Washington(id 5), John Adams(id 3) 和 Thomas Jefferson(id 6) 将可以进入电梯,因为他们的体重和为 250 + 350 + 400 = 1000。
Thomas Jefferson(id 6) 是最后一个体重合适并进入电梯的人。

思路

内层：用窗口函数计算weight的累计和（按照turn的顺序）；外层：找到刚好满足sum_weight<=1000 的第一个人（order by turn desc
limit 1）。

代码

```sql
select
    person_name
from(
    select
        a.*
        , sum(weight) over(order by turn) as sum_weight
    from Queue as a
) as tmp
where sum_weight<=1000 # 注意<=不是<
order by turn desc
limit 1
```

1205. 每月交易II【中等】

`Transactions` 记录表

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。
状态列是类型为 [approved（已批准）、declined（已拒绝）] 的枚举。

`Chargebacks` 表

Column Name	Type
trans_id	int
trans_date	date

退单包含有关放置在事务表中的某些事务的传入退单的基本信息。
trans_id 是 transactions 表的 id 列的外键。
每项退单都对应于之前进行的交易，即使未经批准。

编写一个 SQL 查询，以查找每个月和每个国家/地区的信息：已批准交易的数量及其总金额、退单的数量及其总金额。
注意：在您的查询中，只需显示给定月份和国家，忽略所有为零的行。
以 任意顺序 返回结果表。

查询结果格式如下所示。
输入：
`Transactions` 表：

id	country	state	amount	trans_date
101	US	approved	1000	2019-05-18
102	US	declined	2000	2019-05-19
103	US	approved	3000	2019-06-10
104	US	declined	4000	2019-06-13
105	US	approved	5000	2019-06-15

`Chargebacks` 表：

trans_id	trans_date
102	2019-05-29
101	2019-06-30
105	2019-09-18

输出：

month	country	approved_count	approved_amount	chargeback_count	chargeback_amount
2019-05	US	1	1000	1	2000
2019-06	US	2	8000	1	1000
2019-09	US	0	0	1	5000

思路

方法1：

先做一个cte1，（将Transactions里面declined的去除） union all（Chargebacks和Transactions左连接，状态都是declined）
cte1的结果为：
{"headers":
["id", "country", "state", "amount", "trans_date"],
"values": [
[101, "US", "approved", 1000, "2019-05-18"],
[103, "US", "approved", 3000, "2019-06-10"],
[105, "US", "approved", 5000, "2019-06-15"],
[102, "US", "declined", 2000, "2019-05-29"],
[101, "US", "declined", 1000, "2019-06-30"],
[105, "US", "declined", 5000, "2019-09-18"]]}

然后按照年-月和国家分组统计即可。

代码

```
# 方法1:
with cte1 as(
    select * from Transactions where state <> 'declined'
    union all
    select
        a.trans_id as id
        , b.country
        , 'declined' as state
        , b.amount
        , a.trans_date
    from Chargebacks as a
    left join Transactions as b
    on a.trans_id = b.id
)
select
    date_format(trans_date, '%Y-%m') as month
    , country
    , sum(if(state = 'approved',1,0)) as approved_count
    , sum(if(state='approved',amount,0)) as approved_amount
    , sum(if(state = 'declined',1,0)) as chargeback_count
    , sum(if(state='declined',amount,0)) as chargeback_amount
from cte1
group by date_format(trans_date, '%Y%m'), country
```

1270. 向公司CEO汇报工作的所有人【中等】

难度：中等

员工表：`Employees`

Column Name	Type
employee_id	int
employee_name	varchar
manager_id	int

employee_id 是这个表的主键。
这个表中每一行中，employee_id 表示职工的 ID，employee_name 表示职工的名字，manager_id 表示该职工汇报工作的直线经理。
这个公司 CEO 是 employee_id = 1 的人。

用 SQL 查询出所有直接或间接向公司 CEO 汇报工作的职工的 employee_id 。
由于公司规模较小，经理之间的间接关系不超过 3 个经理。
可以以任何顺序返回无重复项的结果。

查询结果示例如下：
`Employees` table:

employee_id	employee_name	manager_id
1	Boss	1
3	Alice	3
2	Bob	1
4	Daniel	2
7	Luis	4
8	Jhon	3
9	Angela	8
77	Robert	1

`Result` table:

employee_id
2
77
4
7

公司 CEO 的 employee_id 是 1.
employee_id 是 2 和 77 的职员直接汇报给公司 CEO。
employee_id 是 4 的职员间接汇报给公司 CEO 4 --> 2 --> 1 。
employee_id 是 7 的职员间接汇报给公司 CEO 7 --> 4 --> 2 --> 1 。
employee_id 是 3, 8, 9 的职员不会直接或间接的汇报给公司 CEO。

思路

经理之间的间接关系不超过 3 个经理：join 2次。筛选条件：最后join表的manager_id = 1且最开始的表employee_id <> 1（不是CEO）

代码

```sql
select
    a1.employee_id
from Employees as a1
join Employees as a2
on a1.manager_id = a2.employee_id
join Employees as a3
on a2.manager_id = a3.employee_id
where a3.manager_id = 1 and a1.employee_id <> 1
```

1308. 不同性别每日分数总计【中等】

难度：中等

表: `Scores`

Column Name	Type
player_name	varchar
gender	varchar
day	date
score_points	int

(gender, day)是该表的主键
一场比赛是在女队和男队之间举行的
该表的每一行表示一个名叫 (player_name) 性别为 (gender) 的参赛者在某一天获得了 (score_points) 的分数
如果参赛者是女性，那么 gender 列为 'F'，如果参赛者是男性，那么 gender 列为 'M'

写一条SQL语句查询每种性别在每一天的总分。
返回按 gender 和 day 对查询结果 升序排序 的结果。

查询结果格式的示例如下。
输入：
`Scores`表:

player_name	gender	day	score_points
Aron	F	2020-01-01	17
Alice	F	2020-01-07	23
Bajrang	M	2020-01-07	7
Khali	M	2019-12-25	11
Slaman	M	2019-12-30	13
Joe	M	2019-12-31	3
Jose	M	2019-12-18	2
Priya	F	2019-12-31	23
Priyanka	F	2019-12-30	17

输出：

gender	day	total
F	2019-12-30	17
F	2019-12-31	40
F	2020-01-01	57
F	2020-01-07	80
M	2019-12-18	2
M	2019-12-25	13
M	2019-12-30	26
M	2019-12-31	29
M	2020-01-07	36

解释：
女性队伍:
第一天是 2019-12-30，Priyanka 获得 17 分，队伍的总分是 17 分
第二天是 2019-12-31, Priya 获得 23 分，队伍的总分是 40 分
第三天是 2020-01-01, Aron 获得 17 分，队伍的总分是 57 分
第四天是 2020-01-07, Alice 获得 23 分，队伍的总分是 80 分
男性队伍：
第一天是 2019-12-18, Jose 获得 2 分，队伍的总分是 2 分
第二天是 2019-12-25, Khali 获得 11 分，队伍的总分是 13 分
第三天是 2019-12-30, Slaman 获得 13 分，队伍的总分是 26 分

第四天是 2019-12-31, Joe 获得 3 分，队伍的总分是 29 分
第五天是 2020-01-07, Bajrang 获得 7 分，队伍的总分是 36 分

代码

```
# 方法1:
select
    gender
    , day
    , sum(score_points) over(partition by gender order by day) as total
from Scores as a

# 方法2:不用窗口函数
select
    a.gender
    , a.day
    , sum(b.score_points) as total
from Scores as a, Scores as b
where a.gender = b.gender
and a.day >= b.day
group by 1,2
order by 1,2
```

1321. 餐馆营业额变化增长【中等】

难度：中等

表: `Customer`

Column Name	Type
customer_id	int
name	varchar
visited_on	date
amount	int

(customer_id, visited_on) 是该表的主键。
该表包含一家餐馆的顾客交易数据。
visited_on 表示 (customer_id) 的顾客在 visited_on 那天访问了餐馆。
amount 是一个顾客某一天的消费总额。

你是餐馆的老板，现在你想分析一下可能的营业额变化增长（每天至少有一位顾客）。
写一条 SQL 查询计算以 7 天（某日期 + 该日期前的 6 天）为一个时间段的顾客消费平均值。average_amount 要 保留两位小数。
查询结果按 visited_on 排序。

查询结果格式的例子如下。
输入：
`Customer` 表:

customer_id	name	visited_on	amount
1	Jhon	2019-01-01	100
2	Daniel	2019-01-02	110
3	Jade	2019-01-03	120
4	Khaled	2019-01-04	130
5	Winston	2019-01-05	110
6	Elvis	2019-01-06	140
7	Anna	2019-01-07	150
8	Maria	2019-01-08	80
9	Jaze	2019-01-09	110
1	Jhon	2019-01-10	130
3	Jade	2019-01-10	150

输出：

visited_on	amount	average_amount
2019-01-07	860	122.86
2019-01-08	840	120
2019-01-09	840	120
2019-01-10	1000	142.86

解释：
第一个七天消费平均值从 2019-01-01 到 2019-01-07 是restaurant-growth/restaurant-growth/ (100 + 110 + 120 + 130 + 110 + 140 + 150)/7 = 122.86
第二个七天消费平均值从 2019-01-02 到 2019-01-08 是 (110 + 120 + 130 + 110 + 140 + 150 + 80)/7 = 120
第三个七天消费平均值从 2019-01-03 到 2019-01-09 是 (120 + 130 + 110 + 140 + 150 + 80 + 110)/7 = 120
第四个七天消费平均值从 2019-01-04 到 2019-01-10 是 (130 + 110 + 140 + 150 + 80 + 110 + 130 + 150)/7 = 142.86

思路

先按照日期sum(amount)，然后用 rows 6 preceding 计算sum和avg，最外层筛选日期。

代码

```sql
with cte as (
    select
        visited_on
        , sum(amount) as amount
    from Customer
    group by visited_on
)

#  不能range 6 preceding，因为是date数据
#  注意是6不是7
select
    visited_on
    , amount
    , average_amount
from(
    select
        visited_on
        , sum(amount) over(order by visited_on rows 6 preceding ) as amount
        , round(avg(amount) over(order by visited_on rows 6 preceding ),2) as
average_amount
    from cte
    # 这里不能直接where，要再加一层，这里过滤的话，是先过滤日期再窗口函数了
    # where datediff(visited_on, (select min(visited_on) from Customer))>=6
) as tmp
where datediff(visited_on, (select min(visited_on) from Customer))>=6
```

615. 平均工资：部门与公司比较【困难】

难度：困难

给如下两个表，写一个查询语句，求出在每一个工资发放日，每个部门的平均工资与公司的平均工资的比较结果（高 / 低 / 相同）。

表： `salary`

id	employee_id	amount	pay_date
1	1	9000	2017-03-31
2	2	6000	2017-03-31
3	3	10000	2017-03-31
4	1	7000	2017-02-28
5	2	6000	2017-02-28
6	3	8000	2017-02-28

employee_id 字段是表 employee 中 employee_id 字段的外键。

employee_id	department_id
1	1
2	2
3	2

对于如上样例数据，结果为：

pay_month	department_id	comparison
2017-03	1	higher
2017-03	2	lower
2017-02	1	same
2017-02	2	same

解释：
在三月，公司的平均工资是 (9000+6000+10000)/3 = 8333.33...
由于部门 '1' 里只有一个 employee_id 为 '1' 的员工，所以部门 '1' 的平均工资就是此人的工资 9000 。因为 9000 > 8333.33 ，所以比较结果是 'higher'。
第二个部门的平均工资为 employee_id 为 '2' 和 '3' 两个人的平均工资，为 (6000+10000)/2=8000 。因为 8000 < 8333.33 ，所以比较结果是 'lower' 。
在二月用同样的公式求平均工资并比较，比较结果为 'same' ，因为部门 '1' 和部门 '2' 的平均工资与公司的平均工资相同，都是 7000 。

思路

代码

```
with cte1 as (
    select
        pay_date
        , department_id
        , avg(amount) over(partition by department_id, pay_date) as avg_department_pay
        , avg(amount) over(partition by pay_date) as avg_company_pay
    from salary as a
    left join employee as b
    on a.employee_id = b.employee_id
)
# 自己一开始写的，但是感觉最后通过group by来distinct的方式不好
select
    date_format(pay_date,'%Y-%m') as pay_month
    , department_id
    , case when avg_department_pay>avg_company_pay then 'higher'
```

```
            when avg_department_pay=avg_company_pay then 'same'
      else 'lower' end as comparison
from cte1
group by department_id,date_format(pay_date,'%Y-%m')

# 外面套一个select distinct * 更好
select
    distinct *
from (
    select
        date_format(pay_date,'%Y-%m') as pay_month
        , department_id
        , case when avg_department_pay>avg_company_pay then 'higher'
            when avg_department_pay=avg_company_pay then 'same'
        else 'lower' end as comparison
    from cte1
) as tmp
```

1127. 用户购买平台【困难】

支出表: `Spending`

Column Name	Type
user_id	int
spend_date	date
platform	enum
amount	int

这张表记录了用户在一个在线购物网站的支出历史，该在线购物平台同时拥有桌面端（'desktop'）和手机端（'mobile'）的应用程序。
这张表的主键是 (user_id, spend_date, platform)。
平台列 platform 是一种 ENUM ，类型为（'desktop', 'mobile'）。

写一段 SQL 来查找每天 仅 使用手机端用户、仅 使用桌面端用户和 同时 使用桌面端和手机端的用户人数和总支出金额。

查询结果格式如下例所示：

`Spending` table:

user_id	spend_date	platform	amount
1	2019-07-01	mobile	100
1	2019-07-01	desktop	100
2	2019-07-01	mobile	100
2	2019-07-02	mobile	100
3	2019-07-01	desktop	100
3	2019-07-02	desktop	100

`Result` table:

spend_date	platform	total_amount	total_users
2019-07-01	desktop	100	1
2019-07-01	mobile	100	1
2019-07-01	both	200	1
2019-07-02	desktop	100	1
2019-07-02	mobile	100	1
2019-07-02	both	0	0

在 2019-07-01, 用户1 同时 使用桌面端和手机端购买, 用户2 仅 使用了手机端购买，而用户3 仅 使用了桌面端购买。
在 2019-07-02, 用户2 仅 使用了手机端购买, 用户3 仅 使用了桌面端购买，且没有用户 同时 使用桌面端和手机端购买。

```
with cte1 as(
    select distinct spend_date, 'desktop' as platform from Spending
    union all
    select distinct spend_date, 'mobile' as platform from Spending
    union all
    select distinct spend_date, 'both' as platform from Spending
),

# 定义什么是both、desktop和mobile
cte2 as (
    select
        user_id
        , spend_date
        # 注意这个if非常巧妙
        , if(count(distinct platform)=2, 'both', platform) as platform
        , sum(amount) as amount # 注意虽然下面没按照platform分组，但是下面的if已经合并了
```

```
    from Spending
    group by spend_date, user_id
)

select
    a.spend_date
    , a.platform
    , ifnull(sum(b.amount), 0) as total_amount
    , ifnull(count(distinct b.user_id), 0) as total_users
from cte1 as a
left join cte2 as b
on a.spend_date = b.spend_date and a.platform = b.platform
group by a.spend_date, a.platform
```

262. 行程和用户【困难】（非70题）

难度：困难

表：`Trips`

Column Name	Type
id	int
client_id	int
driver_id	int
city_id	int
status	enum
request_at	date

id 是这张表的主键。
这张表中存所有出租车的行程信息。每段行程有唯一 id ，其中 client_id 和 driver_id 是 Users 表中 users_id 的外键。
status 是一个表示行程状态的枚举类型，枚举成员为(‘completed’, ‘cancelled_by_driver’, ‘cancelled_by_client’) 。

表：`Users`

Column Name	Type
users_id	int
banned	enum
role	enum

users_id 是这张表的主键。
这张表中存所有用户，每个用户都有一个唯一的 users_id ，role 是一个表示用户身份的枚举类型，枚举成员为 ('client', 'driver', 'partner') 。
banned 是一个表示用户是否被禁止的枚举类型，枚举成员为 ('Yes', 'No') 。

取消率 的计算方式如下：(被司机或乘客取消的非禁止用户生成的订单数量) / (非禁止用户生成的订单总数)。
写一段 SQL 语句查出 "2013-10-01" 至 "2013-10-03" 期间非禁止用户（乘客和司机都必须未被禁止）的取消率。
非禁止用户即 banned 为 No 的用户，禁止用户即 banned 为 Yes 的用户。
返回结果表中的数据可以按任意顺序组织。其中取消率 Cancellation Rate 需要四舍五入保留 两位小数 。

查询结果格式如下例所示。
输入：
`Trips` 表：

id	client_id	driver_id	city_id	status	request_at
1	1	10	1	completed	2013-10-01
2	2	11	1	cancelled_by_driver	2013-10-01
3	3	12	6	completed	2013-10-01
4	4	13	6	cancelled_by_client	2013-10-01
5	1	10	1	completed	2013-10-02
6	2	11	6	completed	2013-10-02
7	3	12	6	completed	2013-10-02
8	2	12	12	completed	2013-10-03
9	3	10	12	completed	2013-10-03
10	4	13	12	cancelled_by_driver	2013-10-03

`Users` 表：

users_id	banned	role
1	No	client
2	Yes	client
3	No	client
4	No	client
10	No	driver
11	No	driver
12	No	driver
13	No	driver

输出：

Day	Cancellation Rate
2013-10-01	0.33
2013-10-02	0.00
2013-10-03	0.50

解释：
2013-10-01：
 共有 4 条请求，其中 2 条取消。
 然而，id=2 的请求是由禁止用户（user_id=2）发出的，所以计算时应当忽略它。
 因此，总共有 3 条非禁止请求参与计算，其中 1 条取消。
 取消率为 (1 / 3) = 0.33
2013-10-02：
 共有 3 条请求，其中 0 条取消。
 然而，id=6 的请求是由禁止用户发出的，所以计算时应当忽略它。
 因此，总共有 2 条非禁止请求参与计算，其中 0 条取消。
 取消率为 (0 / 2) = 0.00
2013-10-03：
 共有 3 条请求，其中 1 条取消。
 然而，id=8 的请求是由禁止用户发出的，所以计算时应当忽略它。
 因此，总共有 2 条非禁止请求参与计算，其中 1 条取消。
 取消率为 (1 / 2) = 0.50

思路

首先要理解题意，

1. 取消率=(被司机或乘客取消的非禁止用户生成的订单数量) / (非禁止用户生成的订单总数)：分子分母都要去除掉被ban 的id，即where client_id not in ... (select... where banned = 'Yes')还有where driver_id not in ... (select... where banned = 'Yes')
2. 注意日期区间
3. 注意要按照日期进行分组
4. 注意保留两位小数

代码

```
# 注意理解题意，条件很多，不要落
select
    request_at as Day
    , round(
        sum(status = 'cancelled_by_driver' or status = 'cancelled_by_client')
        /count(*)
    ,2) as 'Cancellation Rate'
from Trips
where client_id not in(
    select
        users_id
    from Users
```

```
    where banned = 'Yes' # and role = 'client'
)
and driver_id not in (
    select
        users_id
    from Users
    where banned = 'Yes' # and role = 'driver'
)
and request_at between "2013-10-01" and "2013-10-03"
group by request_at
```