DAY 4

607. 销售员

难度：简单

表: `SalesPerson`

Column Name	Type
sales_id	int
name	varchar
salary	int
commission_rate	int
hire_date	date

sales_id 是该表的主键列。
该表的每一行都显示了销售人员的姓名和 ID ，以及他们的工资、佣金率和雇佣日期。

表: `Company`

Column Name	Type
com_id	int
name	varchar
city	varchar

com_id 是该表的主键列。
该表的每一行都表示公司的名称和 ID ，以及公司所在的城市。

表: `Orders`

Column Name	Type
order_id	int
order_date	date
com_id	int
sales_id	int
amount	int

order_id 是该表的主键列。
com_id 是 `Company` 表中 com_id 的外键。
sales_id 是来自销售员表 sales_id 的外键。
该表的每一行包含一个订单的信息。这包括公司的 ID 、销售人员的 ID 、订单日期和支付的金额。

编写一个SQL查询，报告没有任何与名为 “RED” 的公司相关的订单的所有销售人员的姓名。

以 任意顺序 返回结果表。

查询结果格式如下所示。

示例：

输入：
`SalesPerson` 表:

sales_id	name	salary	commission_rate	hire_date
1	John	100000	6	4/1/2006
2	Amy	12000	5	5/1/2010
3	Mark	65000	12	12/25/2008
4	Pam	25000	25	1/1/2005
5	Alex	5000	10	2/3/2007

`Company` 表:

com_id	name	city
1	RED	Boston
2	ORANGE	New York
3	YELLOW	Boston
4	GREEN	Austin

`Orders` 表:

order_id	order_date	com_id	sales_id	amount
1	1/1/2014	3	4	10000
2	2/1/2014	4	5	5000
3	3/1/2014	1	1	50000
4	4/1/2014	1	4	25000

输出：

name
Amy
Mark
Alex

解释：
根据表 `orders` 中的订单 '3' 和 '4'，容易看出只有 'John' 和 'Pam' 两个销售员曾经向公司 'RED' 销售过。
所以我们需要输出表 `salesperson` 中所有其他人的名字。

思路

不要上来看见三个表就合并，要考虑怎么使用，这里不需要三表连接。用where sales_id not in。考点是子查询匹配。

代码

```
select
    name
from SalesPerson
where sales_id not in
(
     select
        sales_id
    from Orders a
    left join Company b
    on a.com_id = b.com_id
    where b.name = 'RED'
)
```

597. 好友申请 I：总体通过率

难度：简单
表：`FriendRequest`

Column Name	Type
sender_id	int
send_to_id	int
request_date	date

此表没有主键，它可能包含重复项。
该表包含发送请求的用户的 ID ，接受请求的用户的 ID 以及请求的日期。

表：`RequestAccepted`

Column Name	Type
requester_id	int
accepter_id	int
accept_date	date

此表没有主键，它可能包含重复项。
该表包含发送请求的用户的 ID ，接受请求的用户的 ID 以及请求通过的日期。

写一个查询语句，求出好友申请的通过率，用 2 位小数表示。通过率由接受好友申请的数目除以申请总数。

提示：
通过的好友申请不一定都在表 friend_request 中。你只需要统计总的被通过的申请数（不管它们在不在表 FriendRequest 中），并将它除以申请总数，得到通过率
一个好友申请发送者有可能会给接受者发几条好友申请，也有可能一个好友申请会被通过好几次。这种情况下，重复的好友申请只统计一次。
如果一个好友申请都没有，你应该返回 accept_rate 为 0.00 。
查询结果应该如下例所示。

示例 1：
输入：
`FriendRequest` 表：

sender_id	send_to_id	request_date
1	2	2016/06/01
1	3	2016/06/01
1	4	2016/06/01
2	3	2016/06/02
3	4	2016/06/09

`RequestAccepted` 表：

requester_id	accepter_id	accept_date
1	2	2016/06/03
1	3	2016/06/08
2	3	2016/06/08
3	4	2016/06/09
3	4	2016/06/10

输出：

accept_rate
0.8

解释：
总共有 5 个请求，有 4 个不同的通过请求，所以通过率是 0.80

作业

进阶:

- 你能写一个查询语句得到每个月的通过率吗?
- 你能求出每一天的累计通过率吗?

思路

- 两个键一起distinct的方式是`(distinct requester_id, accepter_id)`，两表不用连接，不一定是select ... from ...的结构也可以是直接select ...
- 注意round和ifnull。

代码

```
select
    round(
        ifnull(
            count(distinct requester_id, accepter_id)
             /(select count(distinct sender_id,send_to_id) from FriendRequest)
        ,0)
    ,2) as accept_rate
from RequestAccepted
#或者写成
select
    round(
        ifnull(
            (select count(distinct requester_id, accepter_id) from RequestAccepted) #
注意括号
             /(select count(distinct sender_id,send_to_id) from FriendRequest)
```

```
        ,0)
    ,2) as accept_rate
```

代码（进阶）

每个月的通过率思路：用cte更清晰，首先两个表各自group by year-month，count出(distinct sender_id,send_to_id)作为cte1和cte2；

然后按照year-month join，计算通过率即可。

```
# 每个月的通过率
with cte1 as(
     select
        count(distinct sender_id,send_to_id) as cnt1  # count(distinct
concat(requester_id, accepter_id))这样写吧
        , DATE_FORMAT(request_date,'%Y-%m') as ym
    from FriendRequest
   group by ym
),
cte2 as(
    select
        count(distinct requester_id, accepter_id) as cnt2 # count(distinct
concat(sender_id, send_to_id))
        , DATE_FORMAT(accept_date,'%Y-%m') as ym
    from RequestAccepted
   group by ym
)

select
    round(
        ifnull(
            cnt2/cnt1
        ,0)
    ,2) as accept_rate_ym
from cte2 as c2
left join cte1 as c1
on c1.ym = c2.ym
```

每一天的累计通过率思路：（请直接参考思路2）

```
# 每一天的累计通过率，好奇怪的题，其实不太理解问法，因为通过和申请不一定是同一天啊
# 想法1，count(distinct concat(sender_id, send_to_id)) over(partition by request_date
order by request_date) as cnt1，但是窗口函数好像不让这么count，所以这个想法夭折了。所以最好先
count每天的各自的申请和通过，按照日期连接后再groupby日期后在累计上面的count
# with cte1 as(
#      select
#         count(distinct concat(sender_id, send_to_id)) over(partition by request_date
order by request_date) as cnt1
#     from FriendRequest
```

```
# ),
# cte2 as(
#     select
#         count(distinct concat(requester_id, accepter_id)) over(partition by
accept_date order by accept_date) as cnt2
#     from RequestAccepted
# )

# 想法2， 如果日期都想要包含的话（即两个表的日期都要需要全连接），发现mysql不支持full outer join，但是要是只要有通过的日期的可以
# FriendRequest中按日期count申请数
with cte1 as(
     select
        count(distinct concat(sender_id, send_to_id)) as cnt1
        , request_date as date
    from FriendRequest  # 重复申请是否要剔除?
    group by request_date
)
# RequestAccepted中按日期count通过数，但是注意这里要剔除重复的接受好友
,cte2 as(
    select
        count(distinct concat(requester_id, accepter_id))  as cnt2
        , accept_date as date
    from # 注意这里要剔除重复的接受好友
    (
        select
            *
        from RequestAccepted
        group by requester_id, accepter_id
    ) as tmp
    group by accept_date
)
# 相除得到通过率
# select
#     round(
#         ifnull(
#             cnt2/cnt1
#         ,0)
#     ,2) as accept_rate_every_day
#     , date
# from cte2 as c2
# full outer join cte1 as c1 # 如果日期都想要包含的话（即两个表的日期都要需要全连接），发现mysql不支持full outer join，于是这个方法也不行了，但是要是只要有通过的日期的可以
# using date
# group by date
select
    round(
        ifnull(
            cnt2/cnt1
```

```
        ,0)
    ,2) as accept_rate_every_day
    , c2.date
from cte2 as c2
left join cte1 as c1 # 这样可以
on c1.date = c2.date
group by date

# 想法3:(不太会写，不要参考)大概思路是先得到⬇️
# "headers": ["cnt", "date", "state"], "values":
# [[3, "2016-06-01", "request"],
# [1, "2016-06-02", "request"],
# [1, "2016-06-09", "request"],
# [1, "2016-06-03", "accept"],
# [2, "2016-06-08", "accept"],
# [1, "2016-06-09", "accept"]]}
# with cte as(
#     select
#         count(distinct concat(sender_id, send_to_id)) as cnt
#         , request_date as date
#         , 'request' as state
#     from FriendRequest # 不知道这里要不要剔除重复
#     group by request_date
#     union all
#     select
#         count(distinct concat(requester_id, accepter_id)) as cnt
#         , accept_date as date
#         , 'accept' as state
#     from  # 注意这里要剔除重复的接受好友
#         (
#             select
#                 *
#             from RequestAccepted
#             group by requester_id, accepter_id
#         ) as tmp
#     group by accept_date
# )
# 然后
# select *
# from cte as c1, cte as c2
# where c1.state = "request"
# and c2.state = "accept"
# and c1.date = c2.date
{"headers": ["cnt", "date", "state", "cnt", "date", "state"],
"values": [[1, "2016-06-09", "request", 1, "2016-06-09", "accept"]]}
# 相除就行了但是不知道怎么除，因为有很多日期
```

602. 好友申请 II： 谁有最多的好友【中等】

难度：中等

写一个查询语句，找出拥有最多的好友的人和他拥有的好友数目。
生成的测试用例保证拥有最多好友数目的只有 1 个人。
查询结果格式如下例所示。
示例：
输入：
RequestAccepted 表：

requester_id	accepter_id	accept_date
1	2	2016/06/03
1	3	2016/06/08
2	3	2016/06/08
3	4	2016/06/09

输出：

id	num
3	3

解释：
编号为 3 的人是编号为 1 ，2 和 4 的人的好友，所以他总共有 3 个好友，比其他人都多。
进阶：在真实世界里，可能会有多个人拥有好友数相同且最多，你能找到所有这些人吗?

思路
首先union all（注意union【只有不同的值】和union all【有重复的值】的区别）变成下表：

id
1
1
2
3
2
3
3
4

然后思路和DAY2那四道题一样了

代码

```
with cte as
(   select
        requester_id as id
    from RequestAccepted
    union all
    select
        accepter_id # 这里就不用写as id了
    from RequestAccepted)

select
    id,
    count(*) as num
from cte
group by id
having count(id)>=
all(
    select
        count(*)
    from cte
    group by id
)
# 或者
select
    id
    ,num
from
(
        select
        id
        , count(*) as num
        , dense_rank() over(order by count(*) desc) as rnk
    from cte
    group by id
) a
where rnk = 1
#或者(无法解决进阶问题)
select
    id
    , count(*) as num
from cte
group by id
order by count(*) desc
limit 1
```

1173. 即时食物配送 I

难度：简单
配送表: `Delivery`

Column Name	Type
delivery_id	int
customer_id	int
order_date	date
customer_pref_delivery_date	date

delivery_id 是表的主键。
该表保存着顾客的食物配送信息，顾客在某个日期下了订单，并指定了一个期望的配送日期（和下单日期相同或者在那之后）。
如果顾客期望的配送日期和下单日期相同，则该订单称为 「即时订单」，否则称为「计划订单」。
写一条 SQL 查询语句获取即时订单所占的百分比， 保留两位小数。
查询结果如下所示。
示例 1:
输入：
`Delivery` 表:

delivery_id	customer_id	order_date	customer_pref_delivery_date
1	1	2019-08-01	2019-08-02
2	5	2019-08-02	2019-08-02
3	1	2019-08-11	2019-08-11
4	3	2019-08-24	2019-08-26
5	4	2019-08-21	2019-08-22
6	2	2019-08-11	2019-08-13

输出：

immediate_percentage
33.33

解释：2 和 3 号订单为即时订单，其他的为计划订单。

```
select
    round(
        ifnull(
        100 * count(*)
        / (select count(*) from Delivery)
```

```
        ,0)
    ,2) as immediate_percentage
from Delivery
where order_date = customer_pref_delivery_date

# 另一种逻辑真值的写法：
select
    round(
        100*sum(order_date=customer_pref_delivery_date)
        /count(*)
    ,2) as immediate_percentage
from Delivery
```

1174. 即时食物配送 II【中等】

难度：中等

如果顾客期望的配送日期和下单日期相同，则该订单称为 「即时订单」，否则称为「计划订单」。

「首次订单」是顾客最早创建的订单。我们保证一个顾客只会有一个「首次订单」。

写一条 SQL 查询语句获取即时订单在所有用户的首次订单中的比例。保留两位小数。

查询结果如下所示：

`Delivery` 表：

delivery_id	customer_id	order_date	customer_pref_delivery_date
1	1	2019-08-01	2019-08-02
2	2	2019-08-02	2019-08-02
3	1	2019-08-11	2019-08-12
4	3	2019-08-24	2019-08-24
5	3	2019-08-21	2019-08-22
6	2	2019-08-11	2019-08-13
7	4	2019-08-09	2019-08-09

`Result` 表：

immediate_percentage
50.00

1 号顾客的 1 号订单是首次订单，并且是计划订单。

2 号顾客的 2 号订单是首次订单，并且是即时订单。

3 号顾客的 5 号订单是首次订单，并且是计划订单。

4 号顾客的 7 号订单是首次订单，并且是即时订单。

因此，一半顾客的首次订单是即时的。

代码

```
# 错误写法
```

```
with cte as(
    select *
    from Delivery
    group by customer_id
    having min(order_date) # 这种写法不行!
)
select * from cte
# 这样的结果是:
{"headers":
["delivery_id", "customer_id", "order_date", "customer_pref_delivery_date"],
"values": [
[1, 1, "2019-08-01", "2019-08-02"],
[2, 2, "2019-08-02", "2019-08-02"],
[4, 3, "2019-08-24", "2019-08-24"],  # 注意这一行并不能够选到3号顾客的5号订单 （首次订单），所以这种写法不行
[7, 4, "2019-08-09", "2019-08-09"]]}
select
    round(
        ifnull(
            100 * count(*)
            / (select count(*) from cte)
        ,0)
    ,2) as immediate_percentage
from cte
where order_date = customer_pref_delivery_date
```

```
# 正确写法
select
    round(
        ifnull(
            100 * sum(if(order_date = customer_pref_delivery_date,1,0))
            / count(*)
        ,0)
    ,2) as immediate_percentage
from Delivery
where (customer_id,order_date) in
(
    select
        customer_id
        , min(order_date)
    from Delivery
    group by customer_id
)
```

1113. 报告的记录

难度：简单

动作表：`Actions`

Column Name	Type
user_id	int
post_id	int
action_date	date
action	enum
extra	varchar

此表没有主键，所以可能会有重复的行。
action 字段是 ENUM 类型的，包含:('view', 'like', 'reaction', 'comment', 'report', 'share')
extra 字段是可选的信息（可能为 null），其中的信息例如有：1.报告理由(a reason for report) 2.反应类型(a type of reaction)

编写一条SQL，查询每种 报告理由（report reason）在昨天的不同报告数量（post_id）。假设今天是 2019-07-05。

查询及结果的格式示例：

`Actions` table:

user_id	post_id	action_date	action	extra
1	1	2019-07-01	view	null
1	1	2019-07-01	like	null
1	1	2019-07-01	share	null
2	4	2019-07-04	view	null
2	4	2019-07-04	report	spam
3	4	2019-07-04	view	null
3	4	2019-07-04	report	spam
4	3	2019-07-02	view	null
4	3	2019-07-02	report	spam
5	2	2019-07-04	view	null
5	2	2019-07-04	report	racism
5	5	2019-07-04	view	null
5	5	2019-07-04	report	racism

`Result` table:

report_reason	report_count
spam	1
racism	2

注意，我们只关心报告数量非零的结果。

思路

注意读题，虽然题目说的可能不太清楚

代码

```
select
    extra as report_reason
    , count(distinct post_id) as report_count
from Actions
where datediff('2019-07-05', action_date) = 1
and action = 'report'
#and extra is not null
group by extra
```

1132. 报告的记录 II【中等】

难度：中等

动作表：`Actions`

Column Name	Type
user_id	int
post_id	int
action_date	date
action	enum
extra	varchar

这张表没有主键，并有可能存在重复的行。
action 列的类型是 ENUM，可能的值为 ('view', 'like', 'reaction', 'comment', 'report', 'share')。
extra 列拥有一些可选信息，例如：报告理由（a reason for report）或反应类型（a type of reaction）等。

移除表：`Removals`

Column Name	Type
post_id	int
remove_date	date

这张表的主键是 post_id。
这张表的每一行表示一个被移除的帖子，原因可能是由于被举报或被管理员审查。

编写一段 SQL 来查找：在被报告为垃圾广告的帖子中，被移除的帖子的每日平均占比，四舍五入到小数点后 2 位。

以 任意顺序 返回结果表。

查询结果的格式如下。
示例 1:

输入：
`Actions` table:

user_id	post_id	action_date	action	extra
1	1	2019-07-01	view	null
1	1	2019-07-01	like	null
1	1	2019-07-01	share	null
2	2	2019-07-04	view	null
2	2	2019-07-04	report	spam
3	4	2019-07-04	view	null
3	4	2019-07-04	report	spam
4	3	2019-07-02	view	null
4	3	2019-07-02	report	spam
5	2	2019-07-03	view	null
5	2	2019-07-03	report	racism
5	5	2019-07-03	view	null
5	5	2019-07-03	report	racism

`Removals` table:

post_id	remove_date
2	2019-07-20
3	2019-07-18

输出：

average_daily_percent
75.00

解释：
2019-07-04 的垃圾广告移除率是 50%，因为有两张帖子被报告为垃圾广告，但只有一个得到移除。
2019-07-02 的垃圾广告移除率是 100%，因为有一张帖子被举报为垃圾广告并得到移除。
其余几天没有收到垃圾广告的举报，因此平均值为：(50 + 100) / 2 = 75%
注意，输出仅需要一个平均值即可，我们并不关注移除操作的日期。

思路

关键是在group by之后能想到`count(distinct b.post_id)/count(distinct a.post_id)`

内层：每个action_date的移除rate；外层：取平均

代码

```
select
    round(
        ifnull(100 * avg(rate),0)
    ,2) as average_daily_percent
from(
    select
        count(distinct b.post_id)/count(distinct a.post_id) as rate
    from Actions as a
    left join Removals as b
    on a.post_id = b.post_id
    where a.extra = 'spam'
    group by action_date
)as tmp
```

1142. 过去30天的用户活动 II

难度：简单
Activity 表：

Column Name	Type
user_id	int
session_id	int
activity_date	date
activity_type	enum

该表没有主键，它可能有重复的行。
activity_type 列是 ENUM 类型，可以取（“ open_session”，“ end_session”，“ scroll_down”，“ send_message”）四种活动类型之一。
该表显示了社交媒体网站的用户活动。
请注意，每个会话只属于一个用户。

编写 SQL 查询以查找截至 2019-07-27（含）的 30 天内每个用户的平均会话数，四舍五入到小数点后两位。只统计那些会话期间用户至少进行一项活动的有效会话。

查询结果格式如下例所示。

示例：

输入：
Activity 表：

user_id	session_id	activity_date	activity_type
1	1	2019-07-20	open_session
1	1	2019-07-20	scroll_down
1	1	2019-07-20	end_session
2	4	2019-07-20	open_session
2	4	2019-07-21	send_message
2	4	2019-07-21	end_session
3	2	2019-07-21	open_session
3	2	2019-07-21	send_message
3	2	2019-07-21	end_session
3	5	2019-07-21	open_session
3	5	2019-07-21	scroll_down
3	5	2019-07-21	end_session
4	3	2019-06-25	open_session
4	3	2019-06-25	end_session

输出：

average_sessions_per_user
1.33

思路

1. 内层：按照user_id, session_id分组（选出唯一的（user_id, session_id））；中层：按照user_id分组，count一下；外层：求中层的平均
2. 一种更为巧妙的方法：`count(distinct session_id)/count(distinct user_id)`

代码

```
# 方法1:
select
    round(ifnull(avg(cnt),0),2) as average_sessions_per_user
from(
    select
        count(*) as cnt
    from(
        select
```

```
            *
        from Activity
        where datediff('2019-07-27',activity_date)<30
        group by user_id, session_id
    ) tmp1
    group by user_id
) tmp2

# 方法2:
select
    ifnull(
        round(
            count(distinct session_id)/
            count(distinct user_id)
        ,2)
    ,0) as average_sessions_per_user
from activity
where datediff('2019-07-27',activity_date)<30
```

1098. 小众书籍【中等】

难度：中等

书籍表 `Books`：

Column Name	Type
book_id	int
name	varchar
available_from	date

book_id 是这个表的主键。
订单表 `Orders`：

Column Name	Type
order_id	int
book_id	int
quantity	int
dispatch_date	date

order_id 是这个表的主键。
book_id 是 Books 表的外键。

你需要写一段 SQL 命令，筛选出过去一年中订单总量 少于10本 的 书籍 。
注意：不考虑 上架（available from）距今 不满一个月 的书籍。并且 假设今天是 2019-06-23 。

下面是样例输出结果：
`Books` 表：

book_id	name	available_from
1	"Kalila And Demna"	2010-01-01
2	"28 Letters"	2012-05-12
3	"The Hobbit"	2019-06-10
4	"13 Reasons Why"	2019-06-01
5	"The Hunger Games"	2008-09-21

`Orders` 表：

order_id	book_id	quantity	dispatch_date
1	1	2	2018-07-26
2	1	1	2018-11-05
3	3	8	2019-06-11
4	4	6	2019-06-05
5	4	5	2019-06-20
6	5	9	2009-02-02
7	5	8	2010-04-13

`Result` 表：

book_id	name
1	"Kalila And Demna"
2	"28 Letters"
5	"The Hunger Games"

思路

两个日期问题先用cte筛选好

注意result表中，没有quantity的书也写进去了，所以千万注意ifnull

代码

```sql
with cte1 as(
    select
        book_id
        , name
    from Books
    where datediff('2019-06-23 ',available_from)>30
),
cte2 as(
    select
       book_id
       , quantity
    from Orders
    where datediff('2019-06-23 ',dispatch_date)<365
)
select
    c1.book_id as book_id
    , c1.name as name
from cte1 as c1
left join cte2 as c2
on c1.book_id = c2.book_id
group by c1.book_id
having sum(ifnull(c2.quantity,0))<10 # 注意ifnull，可不要直接sum了
```

1194. 锦标赛优胜者【困难】

难度：困难

Players 玩家表

Column Name	Type
player_id	int
group_id	int

player_id 是此表的主键。
此表的每一行表示每个玩家的组。
Matches 赛事表

Column Name	Type
match_id	int
first_player	int
second_player	int
first_score	int
second_score	int

match_id 是此表的主键。
每一行是一场比赛的记录，first_player 和 second_player 表示该场比赛的球员 ID。
first_score 和 second_score 分别表示 first_player 和 second_player 的得分。
你可以假设，在每一场比赛中，球员都属于同一组。
每组的获胜者是在组内累积得分最高的选手。如果平局，player_id 最小 的选手获胜。
编写一个 SQL 查询来查找每组中的获胜者。
返回的结果表单 没有顺序要求 。

查询结果格式如下所示。
示例 1:
输入：
Players 表:

player_id	group_id
15	1
25	1
30	1
45	1
10	2
35	2
50	2
20	3
40	3

Matches 表:

match_id	first_player	second_player	first_score	second_score
1	15	45	3	0
2	30	25	1	2
3	30	15	2	0
4	40	20	5	2
5	35	50	1	1

输出：

group_id	player_id
1	15
2	35
3	40

思路

1. 使用union把所有人的分数聚集到一个表上；
2. 连接上表和Players表，按照player_id分组，计算每个player_id的总分；
3. 使用窗口函数找到每个group_id中sum_score最大的player_id。

代码

```
with cte1 as(
    select
        first_player as player_id
        , first_score as score
    from Matches
    union all
    select
        second_player as player_id
        , second_score as score
    from Matches
),
cte2 as(
    select
        b.group_id
        , a.player_id
        , sum(score) as sum_score
    from cte1 as a
    left join Players as b
    # using(player_id) # 这个select的时候就不用a.
    on a.player_id = b.player_id
    group by a.player_id
)
select
    group_id
    , player_id
from(
    select
        group_id
        , player_id
        , row_number() over(partition by group_id order by sum_score desc) as rnk
    from cte2
) as tmp
where rnk = 1
```

```
# 这样写不行👇，having后面可以不跟聚合函数，但他只能根据group by后面字段进行再次筛选，比如：group by A having A = xxx ,或者 group by A having max(A) = xxx。
# select
#     group_id
#     , player_id
# from cte2
# group by group_id
# having sum_score>=
# all(
#     select
#         sum_score
#     from cte2
#     group by group_id
# )
```