简单题上手

175 组合两个表

```
@author: sdubrz
@date: 6/12/2020 10:42:23 PM
难度：简单
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/combine-two-tables
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
```

表1: Person

```
+-------------+---------+
| 列名          | 类型      |
+-------------+---------+
| PersonId    | int     |
| FirstName   | varchar |
| LastName    | varchar |
+-------------+---------+
PersonId 是上表主键
```

表2: Address

```
+-------------+---------+
| 列名          | 类型     |
+-------------+---------+
| AddressId   | int     |
| PersonId    | int     |
| City        | varchar |
| State       | varchar |
+-------------+---------+
AddressId 是上表主键
```

编写一个 SQL 查询，满足条件：无论 person 是否有地址信息，都需要基于上述两表提供 person 的以下信息：

```
FirstName, LastName, City, State
```

通过次数136,248提交次数187,182

官方解法

因为表 Address 中的 personId 是表 Person 的外关键字，所以我们可以连接这两个表来获取一个人的地址信息。

考虑到可能不是每个人都有地址信息，我们应该使用 outer join 而不是默认的 inner join。

MySQL代码如下

```
select FirstName, LastName, City, State
from Person left join Address
on Person.PersonId = Address.PersonId
;
```

注意：如果没有某个人的地址信息，使用 where 子句过滤记录将失败，因为它不会显示姓名信息。

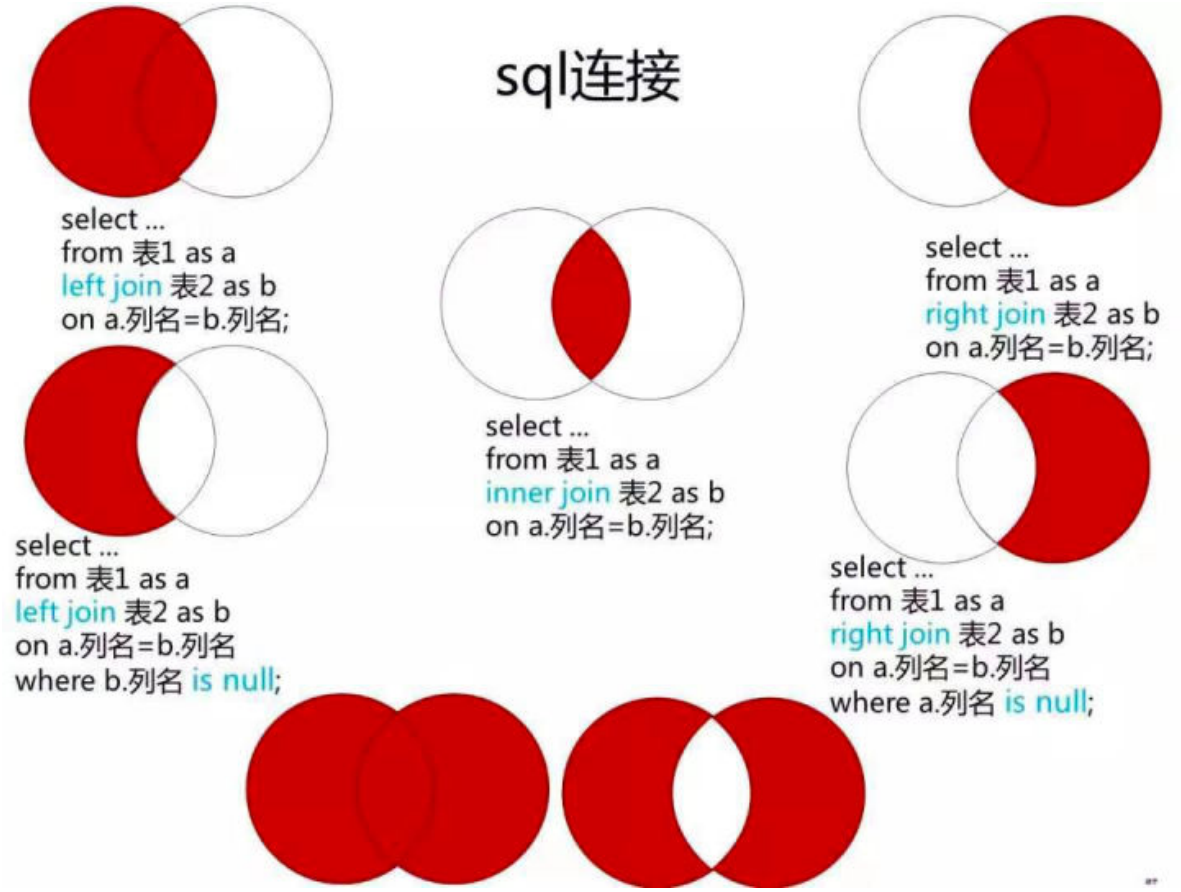
在 LeetCode 系统中提交的结果为

```
执行结果：通过 显示详情
执行用时 :282 ms, 在所有 MySQL 提交中击败了30.46%的用户
内存消耗 :0B, 在所有 MySQL 提交中击败了100.00%的用户
```

备注

左外连接可以理解为在内连接的基础上保证左表的数据全部显示；类似的，右外连接可以理解为在内连接的基础上保证右表的数据全部显示。

下面是一个网友给出的描述SQL多表联结的图示，比较形象。



181 超过经理收入的员工

```
@author: sdubrz
@date: 6/12/2020 11:16:38 PM
难度：简单
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/employees-earning-more-than-their-managers
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
```

SQL架构：

```
Create table If Not Exists Employee (Id int, Name varchar(255), Salary int,
ManagerId int)
Truncate table Employee
insert into Employee (Id, Name, Salary, ManagerId) values ('1', 'Joe', '70000',
'3')
insert into Employee (Id, Name, Salary, ManagerId) values ('2', 'Henry',
'80000', '4')
insert into Employee (Id, Name, Salary, ManagerId) values ('3', 'Sam', '60000',
'None')
insert into Employee (Id, Name, Salary, ManagerId) values ('4', 'Max', '90000',
'None')
```

Employee 表包含所有员工，他们的经理也属于员工。每个员工都有一个 Id，此外还有一列对应员工的经理的 Id。

```
+----+-------+--------+-----------+
| Id | Name  | Salary | ManagerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | NULL      |
| 4  | Max   | 90000  | NULL      |
+----+-------+--------+-----------+
```

给定 Employee 表，编写一个 SQL 查询，该查询可以获取收入超过他们经理的员工的姓名。在上面的表格中，Joe 是唯一一个收入超过他的经理的员工。

```
+----------+
| Employee |
+----------+
| Joe      |
+----------+
```

通过次数69,363提交次数100,082

我的解法

给这张表分两个别名，然后用内连接的方法：

```sql
select a.Name as Employee
from Employee as a, Employee as b
where a.ManagerId=b.Id and a.Salary>b.Salary;
```

在 LeetCode 系统中提交的结果为

```
执行结果：通过 显示详情
执行用时 :381 ms, 在所有 MySQL 提交中击败了33.57%的用户
内存消耗 :0B, 在所有 MySQL 提交中击败了100.00%的用户
```

182 查找重复的电子邮箱

```
@author: sdubrz
@date: 6/13/2020 3:56:14 PM
难度： 简单
来源：力扣（LeetCode）
链接：https://leetcode-cn.com/problems/duplicate-emails
著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。
```

编写一个 SQL 查询，查找 Person 表中所有重复的电子邮箱。

示例：

```
+----+---------+
| Id | Email   |
+----+---------+
| 1  | a@b.com |
| 2  | c@d.com |
| 3  | a@b.com |
+----+---------+
```

根据以上输入，你的查询应返回以下结果：

```
+---------+
| Email   |
+---------+
| a@b.com |
+---------+
```

说明：所有电子邮箱都是小写字母。

通过次数82,673提交次数104,629

我的解法

可以用聚合函数count：

```
select Email
from Person
group by Email
having count(Email)>1;
```

在 LeetCode 系统中提交的结果为

```
执行结果：通过 显示详情
执行用时 :250 ms, 在所有 MySQL 提交中击败了51.92%的用户
内存消耗 :0B, 在所有 MySQL 提交中击败了100.00%的用户
```

解法二

有网友在评论区说，下面的方法速度要快于 group by 的方法，但是我在系统中提交的结果显示并不比 group by 快。 LeetCode 中SQL这部分多次提交时间会有较大的差别，也有可能是这种原因造成的。

```
select distinct a.Email
from Person a, Person b
where a.Email=b.Email and a.Id<>b.Id;
```

183 从不订购的客户

```
@date:8/7/2020 10:43:46 AM
```

某网站包含两个表，Customers 表和 Orders 表。编写一个 SQL 查询，找出所有从不订购任何东西的客户。

Customers 表：

```
+----+-------+
| Id | Name  |
+----+-------+
| 1  | Joe   |
| 2  | Henry |
| 3  | Sam   |
| 4  | Max   |
+----+-------+
```

Orders 表：

```
+----+------------+
| Id | CustomerId |
+----+------------+
| 1  | 3          |
| 2  | 1          |
+----+------------+
```

例如给定上述表格，你的查询应返回：

```
+-----------+
| Customers |
+-----------+
| Henry     |
| Max       |
+-----------+
```

我的解法

```
select Name as Customers
 from Customers
 where Customers.Id not in (select Customers.Id from Customers, Orders where
Customers.Id=Orders.CustomerId);
```

提交结果

```
执行结果：通过 显示详情
执行用时：451 ms, 在所有 MySQL 提交中击败了22.32%的用户
内存消耗：0B, 在所有 MySQL 提交中击败了100.00%的用户
```

官方解法

其实子查询中不用连接两个表：

```
select customers.name as 'Customers'
from customers
where customers.id not in
(
    select customerid from orders
);
```

197 上升的温度

```
@date:2020-08-07
```

给定一个 Weather 表，编写一个 SQL 查询，来查找与之前（昨天的）日期相比温度更高的所有日期的 Id。

```
+---------+------------------+------------------+
| Id(INT) | RecordDate(DATE) | Temperature(INT) |
+---------+------------------+------------------+
|       1 |       2015-01-01 |               10 |
|       2 |       2015-01-02 |               25 |
|       3 |       2015-01-03 |               20 |
|       4 |       2015-01-04 |               30 |
+---------+------------------+------------------+
```

例如，根据上述给定的 Weather 表格，返回如下 Id:

```
+----+
| Id |
+----+
|  2 |
|  4 |
+----+
```

通过次数46,043 提交次数88,982

我的错误题解

```
select A.Id
from Weather as A, Weather as B
where A.Temperature>B.Temperature and A.RecordDate=B.RecordDate+1;
```

在MySQL中比较日期应该用 `DATEDIFF` 函数，上面我用的加号提交之后显示结果错误。 `DATEDIFF` 函数的具体用法为

```
mysql> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
        -> 1
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
        -> -31
```

正确题解

```
select A.Id
from Weather as A, Weather as B
where A.Temperature>B.Temperature and DATEDIFF(A.RecordDate, B.RecordDate)=1;
```

提交结果为

```
执行结果：通过 显示详情
执行用时：401 ms, 在所有 MySQL 提交中击败了54.65%的用户
内存消耗：0B, 在所有 MySQL 提交中击败了100.00%的用户
```

569 超过5名学生的课

有一个courses 表 ，有: student (学生) 和 class (课程)。

请列出所有超过或等于5名学生的课。

例如,表:

```
+---------+------------+
| student | class      |
+---------+------------+
| A       | Math       |
| B       | English    |
| C       | Math       |
| D       | Biology    |
| E       | Math       |
| F       | Computer   |
| G       | Math       |
| H       | Math       |
| I       | Math       |
+---------+------------+
```

应该输出:

```
+---------+
| class   |
+---------+
| Math    |
+---------+
```

Note:
学生在每个课中不应被重复计算。

我的错误解法

一种比较容易想到的方法就是用聚合函数来统计课程出现次数，代码如下所示，但是当表中存在重复记录时，这种方法并不能返回正确的结果。

```
select class
from courses
group by class
having count(*)>=5;
```

正确解法

所以要用`distinct`来去掉重复记录，正确代码如下

```
select class
from courses
group by class
having count(distinct student)>=5;
```

提交结果为

```
执行结果：通过 显示详情
执行用时：216 ms, 在所有 MySQL 提交中击败了73.99%的用户
内存消耗：0B, 在所有 MySQL 提交中击败了100.00%的用户
```

力扣精选70题

1068 产品销售分析 I

```
@date: 2020-08-07
@difficulty: easy
```

销售表 Sales：

```
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| sale_id     | int   |
| product_id  | int   |
| year        | int   |
| quantity    | int   |
| price       | int   |
+-------------+-------+
(sale_id, year) 是销售表 Sales 的主键.
product_id 是关联到产品表 Product 的外键.
注意: price 表示每单位价格
```

产品表 Product：

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
+--------------+---------+
product_id 是表的主键.
```

写一条SQL 查询语句获取 Sales 表中所有产品对应的 产品名称 product_name 以及该产品的所有 售卖年份 year 和 价格 price 。

示例：

```
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      |
+------------+--------------+

Result 表：
+--------------+-------+-------+
| product_name | year  | price |
+--------------+-------+-------+
| Nokia        | 2008  | 5000  |
| Nokia        | 2009  | 5000  |
| Apple        | 2011  | 9000  |
+--------------+-------+-------+
```

解法

只需要简单地连接两个表就可以了

```
select product_name, year, price
from Product, Sales
where Product.product_id=Sales.product_id;
```

1303 求团队人数

```
@date: 2020-08-07
@difficulty: easy
```

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。
```

解法

可以用一个子查询来统计每个团队的人数，然后与原表内连接

```sql
select A.employee_id, B.team_size
from Employee as A , (
    select team_id, count(team_id) as team_size
    from Employee
    group by team_id
) as B
where A.team_id=B.team_id;
```

提交结果为

```
执行用时：240 ms，在所有 MySQL 提交中击败了27.99%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

1069 产品销售分析II

```
@date: 2020-08-07
@difficulty: easy
```

销售表：Sales

```
+-------------+-------+
| Column Name | Type  |
+-------------+-------+
| sale_id     | int   |
| product_id  | int   |
| year        | int   |
| quantity    | int   |
| price       | int   |
+-------------+-------+
sale_id 是这个表的主键。
product_id 是 Product 表的外键。
请注意价格是每单位的。
```

产品表：Product

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
+--------------+---------+
product_id 是这个表的主键。
```

编写一个 SQL 查询，按产品 id product_id 来统计每个产品的销售总量。

查询结果格式如下面例子所示:

```
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      |
+------------+--------------+

Result 表：
+--------------+----------------+
| product_id   | total_quantity |
+--------------+----------------+
| 100          | 22             |
| 200          | 15             |
+--------------+----------------+
```

解法

其实只用到了Sales表

```
select product_id, sum(quantity) as total_quantity
from Sales
group by product_id;
```

提交结果为

```
执行用时：881 ms，在所有 MySQL 提交中击败了59.23%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

613 直线上的最近距离

```
@date: 2020-08-07
@difficulty: easy
```

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？

解法

```
select min(A.x-B.x) as shortest
from point as A, point as B
where A.x>B.x;
```

提交结果为

```
执行用时：318 ms，在所有 MySQL 提交中击败了6.35%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

1251 平均售价

```
@date: 2020-08-07
@difficulty: easy
```

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
```

解法

这里需要注意**保留小数位数的方法，可以使用 `round()` 函数。**

```
select A.product_id, round(sum(A.total_price)/sum(A.units), 2) as average_price
from
(select Prices.product_id as product_id, Prices.price*UnitsSold.units as
total_price, UnitsSold.units as units
from Prices, UnitsSold
where Prices.product_id=UnitsSold.product_id and datediff(purchase_date,
start_date)>=0 and datediff(purchase_date, end_date)<=0) as A
group by product_id;
```

提交结果为

```
执行用时：584 ms，在所有 MySQL 提交中击败了24.06%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

584 寻找用户推荐人

```
@date: 2020-08-07
@difficulty: easy
```

给定表 `customer`，里面保存了所有客户信息和他们的推荐人。

```
+------+------+-----------+
| id   | name | referee_id|
+------+------+-----------+
|    1 | Will |      NULL |
|    2 | Jane |      NULL |
|    3 | Alex |         2 |
|    4 | Bill |      NULL |
|    5 | Zack |         1 |
|    6 | Mark |         2 |
+------+------+-----------+
```

写一个查询语句，返回一个编号列表，列表中编号的推荐人的编号都 **不是** 2。

对于上面的示例数据，结果为：

```
+------+
| name |
+------+
| Will |
| Jane |
| Bill |
| Zack |
+------+
```

通过次数4,882 提交次数6,360

解法

需要注意当推荐人为空时的情况，**判断是否为空应该用 `is NULL` 而不是 `=NULL`。而如果要判断不为空的话应该用 `is not NULL`**

```
select name
from customer
where referee_id<>2 or referee_id is NULL;
```

提交结果为

```
执行用时：403 ms，在所有 MySQL 提交中击败了37.93%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

1173 即时食物配送 I

```
@date: 2020-08-07
@difficulty: easy
```

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：

```
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                  |
+-------------+-------------+------------+-----------------------------+

Result 表:
+----------------------+
| immediate_percentage |
+----------------------+
| 33.33                |
+----------------------+
2 和 3 号订单为即时订单，其他的为计划订单。
```

通过次数2,786 提交次数3,696

解法

要注意两个子查询要用括号括起来

```
select round(
    (select count(*) from Delivery where datediff(order_date,
customer_pref_delivery_date)=0)
    /(select count(*) from Delivery)*100, 2) as immediate_percentage;
```

提交结果为

```
执行用时：568 ms，在所有 MySQL 提交中击败了6.07%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

LeetCode官方给出的另外两种解法

我们可以使用 `sum` 和 `case when` 计算出即时订单的数量。当满足条件的时候，使用 `case when` 将 `sum` 值加 1。

```
select round (
    sum(case when order_date = customer_pref_delivery_date then 1 else 0 end) /
    count(*) * 100,
    2
) as immediate_percentage
from Delivery
```

我们还可以直接使用 `sum`。当 `order_date = customer_pref_delivery_date` 为真时，`sum` 值加 1。

```sql
select round (
    sum(order_date = customer_pref_delivery_date) /
    count(*) * 100,
    2
) as immediate_percentage
from Delivery
```

1082 销售分析 I

```
@date: 2020-08-08
@difficulty: easy
```

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.
```

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.
```

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。

查询结果格式如下所示：

```
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  |
+-----------+------------+----------+------------+----------+-------+

Result 表:
+-------------+
| seller_id   |
+-------------+
| 1           |
| 3           |
+-------------+
Id 为 1 和 3 的销售者，销售总金额都为最高的 2800。
```

通过次数4,114 提交次数5,461

解法

```
select seller_id
from Sales
group by seller_id
having sum(price)>=all(
    select sum(price)
    from Sales
    group by seller_id);
```

提交结果为

```
执行用时：846 ms，在所有 MySQL 提交中击败了75.92%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

1050 合作过至少三次的演员和导演

```
@date: 2020-08-08
@difficulty: easy
```

`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。
```

通过次数4,331 提交次数5,754

解法

```sql
select actor_id, director_id
from ActorDirector
group by actor_id, director_id
having count(*)>=3;
```

586 订单最多的客户

```
@date: 2020-08-08
@difficulty: easy
```

在表 **orders** 中找到订单数最多客户对应的 **customer_number** 。

数据保证订单数最多的顾客恰好只有一位。

表 **orders** 定义如下：

```
| Column            | Type      |
|-------------------|-----------|
| order_number (PK) | int       |
| customer_number   | int       |
| order_date        | date      |
| required_date     | date      |
| shipped_date      | date      |
| status            | char(15)  |
| comment           | char(200) |
```

样例输入

```
| order_number | customer_number | order_date | required_date | shipped_date |
status | comment |
|--------------|-----------------|------------|---------------|--------------|--
------|---------|
| 1            | 1               | 2017-04-09 | 2017-04-13    | 2017-04-12   |
Closed |         |
| 2            | 2               | 2017-04-15 | 2017-04-20    | 2017-04-18   |
Closed |         |
| 3            | 3               | 2017-04-16 | 2017-04-25    | 2017-04-20   |
Closed |         |
| 4            | 3               | 2017-04-18 | 2017-04-28    | 2017-04-25   |
Closed |         |
```

样例输出

```
| customer_number |
|-----------------|
| 3               |
```

解释

```
customer_number 为 '3' 的顾客有两个订单，比顾客 '1' 或者 '2' 都要多，因为他们只有一个订单
所以结果是该顾客的 customer_number ，也就是 3 。
```

进阶： 如果有多位顾客订单数并列最多，你能找到他们所有的 customer_number 吗?

通过次数4,957 提交次数6,616

解法

```
select customer_number
from orders
group by customer_number
having count(customer_number) >= all(
    select count(customer_number)
    from orders
    group by customer_number
);
```

提交结果为

```
执行用时：513 ms，在所有 MySQL 提交中击败了13.24%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

官方解法

官方给出了一个更为快速的方法，不过不能应对多个顾客并列第一的情况：

```sql
SELECT
    customer_number
FROM
    orders
GROUP BY customer_number
ORDER BY COUNT(*) DESC
LIMIT 1
;
```

1148 文章浏览 I

```
@date: 2020-08-08
@difficulty: easy
```

`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    |
+------+
```

通过次数2,741 提交次数3,704

解法

```
select distinct(author_id) as id
from Views
where author_id=viewer_id
order by author_id asc;
```

提交结果为

```
执行用时：325 ms，在所有 MySQL 提交中击败了66.06%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

511 游戏玩法分析

```
@date: 2020-08-08
@difficulty: easy
```

活动表 `Activity`：

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
表的主键是 (player_id, event_date)。
这张表展示了一些游戏玩家在游戏平台上的行为活动。
每行数据记录了一名玩家在退出平台之前，当天使用同一台设备登录平台后打开的游戏的数目（可能是 0 个）。
```

写一条 SQL 查询语句获取每位玩家 **第一次登陆平台的日期。**

查询结果的格式如下所示：

```
Activity 表：
+-----------+-----------+------------+--------------+
| player_id | device_id | event_date | games_played |
+-----------+-----------+------------+--------------+
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-05-02 | 6            |
| 2         | 3         | 2017-06-25 | 1            |
| 3         | 1         | 2016-03-02 | 0            |
| 3         | 4         | 2018-07-03 | 5            |
+-----------+-----------+------------+--------------+

Result 表：
+-----------+-------------+
| player_id | first_login |
+-----------+-------------+
| 1         | 2016-03-01  |
| 2         | 2017-06-25  |
| 3         | 2016-03-02  |
```

```
+-----------+-------------+
```

通过次数5,587 提交次数7,649

解法

```
select player_id, min(event_date) as first_login
from Activity
group by player_id;
```

提交结果为

```
执行用时：401 ms, 在所有 MySQL 提交中击败了82.00%的用户
内存消耗：0B, 在所有 MySQL 提交中击败了100.00%的用户
```

577 员工奖金

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

选出所有 bonus < 1000 的员工的 name 及其 bonus。

`Employee` 表单

```
+-------+--------+-----------+--------+
| empId |  name  | supervisor| salary |
+-------+--------+-----------+--------+
|   1   | John   |  3        | 1000   |
|   2   | Dan    |  3        | 2000   |
|   3   | Brad   |  null     | 4000   |
|   4   | Thomas |  3        | 4000   |
+-------+--------+-----------+--------+
empId 是这张表单的主关键字
```

`Bonus` 表单

```
+-------+-------+
| empId | bonus |
+-------+-------+
| 2     | 500   |
| 4     | 2000  |
+-------+-------+
empId 是这张表单的主关键字
```

输出示例：

```
+-------+-------+
| name  | bonus |
+-------+-------+
| John  | null  |
| Dan   | 500   |
| Brad  | null  |
+-------+-------+
```

通过次数4,951 提交次数6,974

解法

因为有的员工没有奖金，所以这题用左外连接比较合适

```
select Employee.name as name, Bonus.bonus as bonus
from Employee left join Bonus
on Employee.empId=Bonus.empId
where Bonus.bonus<1000 or Bonus.bonus is NULL;
```

提交结果为

```
执行用时：281 ms，在所有 MySQL 提交中击败了16.51%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

603 连续空余座位

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

几个朋友来到电影院的售票处，准备预约连续空余座位。

你能利用表 `cinema` ，帮他们写一个查询语句，获取所有空余座位，并将它们按照 seat_id 排序后返回吗?

```
| seat_id | free |
|---------|------|
| 1       | 1    |
| 2       | 0    |
| 3       | 1    |
| 4       | 1    |
| 5       | 1    |
```

对于如上样例，你的查询语句应该返回如下结果。

```
| seat_id |
|---------|
| 3       |
| 4       |
| 5       |
```

注意：

- seat_id 字段是一个自增的整数，free 字段是布尔类型（'1' 表示空余，'0' 表示已被占据）。
- 连续空余座位的定义是大于等于 2 个连续空余的座位。

通过次数4,349 提交次数6,391

我的解法

```
select distinct(A.seat_id) as seat_id
from cinema as A, cinema as B
where (A.seat_id=B.seat_id-1 or A.seat_id=B.seat_id+1)
    and A.free=1 and B.free=1
order by seat_id;
```

提交结果

```
执行用时：307 ms，在所有 MySQL 提交中击败了38.08%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

官方解法

官方给出的解法与我的解法基本一致

```
select distinct a.seat_id
from cinema a join cinema b
  on abs(a.seat_id - b.seat_id) = 1
  and a.free = true and b.free = true
order by a.seat_id
;
```

607 销售员

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

描述

给定 3 个表：`salesperson`，`company`，`orders`。
输出所有表 `salesperson` 中，没有向公司 'RED' 销售任何东西的销售员。

示例：
输入

表：`salesperson`

```
+----------+------+--------+-----------------+-----------+
| sales_id | name | salary | commission_rate | hire_date |
+----------+------+--------+-----------------+-----------+
|   1      | John | 100000 |     6           | 4/1/2006  |
|   2      | Amy  | 120000 |     5           | 5/1/2010  |
|   3      | Mark | 65000  |     12          | 12/25/2008|
|   4      | Pam  | 25000  |     25          | 1/1/2005  |
|   5      | Alex | 50000  |     10          | 2/3/2007  |
+----------+------+--------+-----------------+-----------+
```

表 `salesperson` 存储了所有销售员的信息。每个销售员都有一个销售员编号 **sales_id** 和他的名字 **name** 。

表： `company`

```
+---------+--------+------------+
| com_id  |  name  |    city    |
+---------+--------+------------+
|   1     |  RED   |   Boston   |
|   2     | ORANGE |   New York |
|   3     | YELLOW |   Boston   |
|   4     | GREEN  |   Austin   |
+---------+--------+------------+
```

表 `company` 存储了所有公司的信息。每个公司都有一个公司编号 **com_id** 和它的名字 **name** 。

表： `orders`

```
+----------+------------+---------+----------+--------+
| order_id | order_date | com_id  | sales_id | amount |
+----------+------------+---------+----------+--------+
| 1        |   1/1/2014 |    3    |    4     | 100000 |
| 2        |   2/1/2014 |    4    |    5     | 5000   |
| 3        |   3/1/2014 |    1    |    1     | 50000  |
| 4        |   4/1/2014 |    1    |    4     | 25000  |
+----------+----------+---------+----------+--------+
```

表 `orders` 存储了所有的销售数据，包括销售员编号 **sales_id** 和公司编号 **com_id** 。

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。

通过次数4,062 提交次数6,095

我的解法

用一个子查询，然后用 `not in` 的方法：

```sql
select salesperson.name as name
from salesperson
where salesperson.sales_id not in(
    select orders.sales_id
    from company, orders
    where company.com_id=orders.com_id and company.name='RED'
);
```

提交结果为

```
执行用时：1025 ms，在所有 MySQL 提交中击败了28.36%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

1294 不同国家的天气类型

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

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。
```

通过次数2,199 提交次数3,331

我的解法

这里**需要注意在SQL语句中条件语句的使用方法。**

```
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, weather
where datediff(day, '2019-11-01')>=0 and datediff(day, '2019-11-30')<=0 and
Countries.country_id=Weather.country_id
group by country_name;
```

提交结果为

```
执行用时：424 ms，在所有 MySQL 提交中击败了61.41%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

网友的解法

下面是一个网友的解法，他的**时间处理要比我的更地道一些**

```
SELECT
    C.country_name,
    CASE
        WHEN AVG(W.weather_state) <= 15 THEN 'Cold'
        WHEN AVG(W.weather_state) >= 25 THEN 'Hot'
        ELSE 'Warm'
    END AS weather_type
FROM
    Countries AS C
    INNER JOIN Weather AS W
    ON C.country_id = W.country_id
WHERE
    YEAR(W.day) = 2019
    AND
    MONTH(W.day) = 11
GROUP BY
    C.country_id;
```

610 判断三角形

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

一个小学生 Tim 的作业是判断三条线段是否能形成一个三角形。

然而，这个作业非常繁重，因为有几百组线段需要判断。

假设表 `triangle` 保存了所有三条线段的三元组 x, y, z ，你能帮 Tim 写一个查询语句，来判断每个三元组是否可以组成一个三角形吗？

```
| x  | y  | z  |
|----|----|----|
| 13 | 15 | 30 |
| 10 | 20 | 15 |
```

对于如上样例数据，你的查询语句应该返回如下结果：

```
| x  | y  | z  | triangle |
|----|----|----|----------|
| 13 | 15 | 30 | No       |
| 10 | 20 | 15 | Yes      |
```

通过次数3,611 提交次数5,478

我的解法

只需要判断每一条边是否小于另外两条边之和就可以了，这里仍然需要用到**SQL语句中的条件判断语句**

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

提交结果为

```
执行用时：199 ms, 在所有 MySQL 提交中击败了80.84%的用户
内存消耗：0B, 在所有 MySQL 提交中击败了100.00%的用户
```

网友解法

也可以用 ** `IF` 语句** ：

```
Select *,IF(x+y>z and x+z>y and y+z>x, "Yes", "No") AS triangle
FROM triangle
```

1075 项目员工 I

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

项目表 `Project`：

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| project_id  | int     |
| employee_id | int     |
+-------------+---------+
主键为 (project_id, employee_id)。
employee_id 是员工表 Employee 表的外键。
```

员工表 `Employee`：

```
+------------------+---------+
| Column Name      | Type    |
+------------------+---------+
| employee_id      | int     |
| name             | varchar |
| experience_years | int     |
+------------------+---------+
主键是 employee_id。
```

请写一个 SQL 语句，查询每一个项目中员工的 **平均** 工作年限，**精确到小数点后两位。**

查询结果的格式如下：

```
Project 表：
+-------------+-------------+
| project_id  | employee_id |
+-------------+-------------+
| 1           | 1           |
| 1           | 2           |
| 1           | 3           |
| 2           | 1           |
| 2           | 4           |
+-------------+-------------+

Employee 表：
+-------------+--------+------------------+
| employee_id | name   | experience_years |
+-------------+--------+------------------+
| 1           | Khaled | 3                |
| 2           | Ali    | 2                |
| 3           | John   | 1                |
| 4           | Doe    | 2                |
+-------------+--------+------------------+

Result 表：
+-------------+---------------+
| project_id  | average_years |
+-------------+---------------+
| 1           | 2.00          |
| 2           | 2.50          |
+-------------+---------------+
第一个项目中，员工的平均工作年限是 (3 + 2 + 1) / 3 = 2.00；第二个项目中，员工的平均工作年限是 (3 + 2) / 2 = 2.50
```

通过次数3,170 提交次数4,837

我的解法

```sql
select project_id, round(avg(experience_years), 2) as average_years
from Project, Employee
where Project.employee_id=Employee.employee_id
group by project_id;
```

提交结果为

```
执行用时：665 ms，在所有 MySQL 提交中击败了39.22%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

1211 查询结果的质量和占比

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

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
```

通过次数2,141 提交次数3,356

我的解法

写得比较复杂了

```sql
select D.query_name, D.quality, case when C.poor_query_percentage is NULL then
round(0, 2) else C.poor_query_percentage end as poor_query_percentage
from
(select B.query_name, round(A.poor_number/B.total_name*100, 2) as
poor_query_percentage
from
(select query_name, count(*) as poor_number
from Queries
where rating<3
group by query_name) as A right join (
    select query_name, count(*) as total_name
    from Queries
    group by query_name
) as B
on A.query_name=B.query_name) as C, (
    select query_name, round(sum(rating/position)/count(*), 2) as quality
    from Queries
    group by query_name
) as D
where C.query_name=D.query_name;
```

提交结果为

```
执行用时：574 ms，在所有 MySQL 提交中击败了27.18%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

官方解法

这道题官方给出的答案比较灵活地运用了MySQL中的聚合函数，代码要比我的简单清楚不少

```sql
SELECT
    query_name,
    ROUND(AVG(rating/position), 2) quality,
    ROUND(SUM(IF(rating < 3, 1, 0)) * 100 / COUNT(*), 2) poor_query_percentage
FROM Queries
GROUP BY query_name
```

1241 每个帖子的评论数

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

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。

查询结果格式如下例所示：

```
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          |
+---------+------------+

结果表：
+---------+--------------------+
| post_id | number_of_comments |
+---------+--------------------+
| 1       | 3                  |
| 2       | 2                  |
| 12      | 0                  |
+---------+--------------------+

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。
```

通过次数2,156 提交次数3,627

我的解法

查看评论关系需要用到外连接，而要保证没有被评论的文章也统计进去，就需要用到左外连接。为了避免没有被评论的文章统计为1，就要用到 `IF` 语句，因而，MySQL实现如下

```
select A.sub_id as post_id, sum(If(B.sub_id is not NULL, 1, 0)) as
number_of_comments
from (select distinct(sub_id) from Submissions where parent_id is NULL) as A
left join
(select distinct(sub_id), parent_id from Submissions) as B
on A.sub_id=B.parent_id
group by A.sub_id
order by post_id
```

提交结果为

```
执行用时：1217 ms，在所有 MySQL 提交中击败了16.78%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

官方解法

官方给出解法还是要更加地道一些。

```
SELECT post_id, COUNT(sub_id) AS number_of_comments
FROM (
    SELECT DISTINCT post.sub_id AS post_id, sub.sub_id AS sub_id
    FROM Submissions post
    LEFT JOIN Submissions sub
    ON post.sub_id = sub.parent_id
    WHERE post.parent_id is null
) T
GROUP BY post_id
ORDER BY post_id ASC
```

1113 报告的记录

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

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）在昨天的报告数量。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。

通过次数2,161 提交次数3,991

我的解法

开始没有看明白例题中的 `spam` 为什么是1个，看了评论区之后才知道报告不能重复的意思是 `post_id` 不能重复，感觉这是题目没有讲清楚。这样MySQL代码为

```sql
select extra as report_reason, count(distinct(post_id)) as report_count
from Actions
where action_date='2019-07-04' and action='report' and extra is not NULL
group by extra
```

提交结果为

```
执行用时：507 ms，在所有 MySQL 提交中击败了30.00%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```

1084 销售分析III

```
@date: 2020-08-08
@difficulty: easy
```

SQL架构

Table: `Product`

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| product_id   | int     |
| product_name | varchar |
| unit_price   | int     |
+--------------+---------+
product_id 是这个表的主键
```

Table: `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.
```

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。

查询结果格式如下所示：

```
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         | 2          | 3        | 2019-06-02 | 1        | 800   |
| 3         | 3          | 4        | 2019-05-13 | 2        | 2800  |
+-----------+------------+----------+------------+----------+-------+

Result table:
+-------------+--------------+
| product_id  | product_name |
+-------------+--------------+
| 1           | S8           |
+-------------+--------------+
id为1的产品仅在2019年春季销售，其他两个产品在之后销售。
```

通过次数3,831 提交次数7,198

我的解法

注意审题，要查询的是仅在2019年第一季度出售的商品。所以要先查出在其他时间销售过的商品，然后从所有销售过的商品中去除这部分。

```sql
select distinct Product.product_id, product_name
from Product, Sales
where Product.product_id not in (
    select distinct(Product.product_id)
    from Product, Sales
    where Product.product_id=Sales.product_id and (datediff(sale_date, '2019-01-01')<0 or datediff(sale_date, '2019-03-31')>0)
) and Product.product_id=Sales.product_id;
```

提交结果为

```
执行用时：853 ms，在所有 MySQL 提交中击败了70.16%的用户
内存消耗：0B，在所有 MySQL 提交中击败了100.00%的用户
```