连续几次出现的问题

> 实质上就是某个唯一标识或者实体在一个表中，在某个计算维度连续出现三次的题目

连续出现次数问题

```
Logs 表：
+----+-----+
| Id | Num |
+----+-----+
| 1  | 1   |
| 2  | 1   |
| 3  | 1   |
| 4  | 2   |
| 5  | 1   |
| 6  | 2   |
| 7  | 2   |
+----+-----+

Result 表：
+-----------------+
| ConsecutiveNums |
+-----------------+
| 1               |
+-----------------+
1 是唯一连续出现至少三次的数字。
```

```sql
select distinct Num as ConsecutiveNums from
(select
    Num,
    (
        row_number() OVER (ORDER BY id ASC) -
        row_number() OVER (partition by NUM order By id asc)
    ) as series_id
    from Logs
) tmp
group by Num ,series_id
HAVING count(1)>=3;
```

用户连续N日登录

```sql
#  涉及表字段user_id,以及登陆时间
SELECT
    user_id,
    count(1) cnt
FROM
    (
        SELECT
```

```
            user_id,
            login_date,
            row_number () over (PARTITION BY user_id ORDER BY login_date ) rn
        FROM
            Login
    ) a
GROUP BY a.user_id,date_sub(a.login_date, INTERVAL rn day)
HAVING count(1) >= 3;
```

延伸

```
#最大连续登录天数
select
    uid,
    max(count)
from
    (
        select
            uid,
            logindate,
            data_sub(logindate,rank) as series,
            count(1) as count --连续登陆天数
        from
            (
                select
                    uid,
                    logindate,
                    row_number() over(partition by uid order by logindate desc)
as rank
                from
                    user_login
            )a
        group by
            uid,
            data_sub(logindate,rank) -- 连续登陆的id，这个差值是相同的
    )
group by uid
```

体育馆的人流量问题

#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。visit_date 升序排列

601. 体育馆的人流量

难度 困难 191

SQL架构 >

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

```
--连续三行或者多行 意味着每行记录有共同点 转化成数字是否连续的问题，那么根据100条件筛选后，对id做排序 求与序号的差值，如果id连续，那么差值必然相等。而后对差值相等的记录做一个累加计数，计数后数量大于等于3的即为结果

select id,visit_date,people from (select id,visit_date,people,count(*) over(partition by cz) cnt from (select id,visit_date,id - (row_number() over(order by id asc)) cz,people from Stadium where people >= 100) t0) t1 where cnt>=3 order by visit_date ;
```

连续空余座位

603. 连续空余座位

难度 简单 65

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

```
select
    seat_id
from
    (
        select
            seat_id,
            count(*) over(partition by cz) as cnt
        from
            (
                select seat_id,(seat_id - row_number() over(order by seat_id))
as cz from cinema where free = 1
            )t0
    )t1
where cnt >= 2
```

某个组织内的topN

> 典型案例
>
> 1、某个班内分数top3的学生分数

> 2、某个学生所有科目分数中的top3分数
>
> 3、某个部门薪水top3的人

每个人最大分数的科目

姓名	语文	数学
A	70	85
B	90	99
C	88	77
D	66	95
…	23	86
Z	98	78

结果如下:	
姓名	最大分数科目
A	数学
B	数学
C	数学
D	数学
E	语文

```
select name,case when shuxue-yuwen > 0 then '数学' else '语文' end from score;
```

每个人分数最高的科目以及分数

姓名	语文	数学	英文
A	70	85	88
B	90	99	66
C	88	77	99
D	66	95	77
Z	98	78	76

结果如下：		
姓名	最大科目分数	科目
A	88	英文
B	99	数学
C	99	英文
D	95	数学
E	86	语文

思路

> 类似班级内求最高分数的同学，或者某个部门最高薪水的员工，可用原生sql写，也可用窗口函数实现
>
> 1、列转行，union all
>
> 2、name分组内排序
>
> 3、取出组内第一的分数和科目

```
#基础写法
SELECT
    name,class,grade
FROM
    (
            SELECT
                name,
                grade,
                class,
                if(
                    @name = name,
                    if(
                        @grade = grade,@rank := @rank,@rank := @rank +1
                    ),
                    @rank :=1
                ) as rank,
                @name := name,
                @grade := grade
            FROM
            (
                    SELECT
                        @rank :=0,@name :=NULL,@grade :=NULL,name,class,grade
                    FROM (
                                SELECT a.name,a.shuxue as grade ,'数学' as class
FROM score a
                            UNION ALL
                            SELECT b.name,b.yingyu as grade ,'英语' as class FROM
score b
                            UNION ALL
                            SELECT c.name,c.yuwen as grade ,'语文' as class FROM
score c
                            ) t0
                    ORDER BY
                        name asc , grade desc
```

```
            )t1
    )t2
WHERE rank = 1
GROUP BY name;
#窗口函数
#首先要列转行而后使用窗口函数做排序
SELECT name,row_number(),class over(partition by name order by grade) AS rank
FROM score WHERE rank = 1;
```

求表中每个同学的总分占班级的占比

姓名	数学	语文	**英语**
A	63	77	80
B	74	78	62
C	78	78	78
D	84	62	62
E	92	94	**64**

结果如下:	
姓名	占比
A	20%
B	19%
C	21%
D	18%
E	22%

sql

```
select
    name,CONCAT(CAST(ROUND((p_total/a_total)*100,0) AS CHAR),'%') as rate
from
    (select name,shuxue+yuwen+yingyu as p_total from score) t0,
    (select SUM(shuxue+yuwen+yingyu) as a_total from score) t1;
```

求表中不重复人员的数量

单据	姓名
A111062	A
A182934	A
A131608	B
A195334	C
A116524	B
A125031	A
A132999	D
A174581	C
A116905	E
A113802	A
A176447	F
A102222	S
A174531	A
A154632	S
A100784	C

结果如下：	
姓名	数量
A	5
B	2
C	3
D	1
E	1
F	1
S	2

sql

```
select count(distinct 单据) from table
```

行程和用户

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。

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)。

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：

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

Result 表：

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

```
select
    Request_at as Day,
    round(count(case when status != 'completed' THEN 1 else NULL
END)/count(status),2)  as Cancellation_Rate
from Trips
    where
        Client_id  in (select Users_id from Users where Banned = 'No')
    AND Driver_id  in (select Users_id from Users where Banned = 'No')
    AND Request_at between '2013-10-01' AND '2013-10-03'
    group by Request_at
```

登录天数累加

Activity table:

```
Create table If Not Exists Activity (player_id int, device_id int, event_date
date, games_played int);
Truncate table Activity;
insert into Activity (player_id, device_id, event_date, games_played) values
('1', '2', '2016-03-01', '5');
insert into Activity (player_id, device_id, event_date, games_played) values
('1', '2', '2016-03-02', '6');
insert into Activity (player_id, device_id, event_date, games_played) values
('2', '3', '2017-06-25', '1');
insert into Activity (player_id, device_id, event_date, games_played) values
('3', '1', '2016-03-02', '0');
insert into Activity (player_id, device_id, event_date, games_played) values
('3', '4', '2018-07-03', '5');
```

Result table:

```
+-----------+------------+---------------------+
| player_id | event_date | games_played_so_far |
+-----------+------------+---------------------+
| 1         | 2016-03-01 | 5                   |
| 1         | 2016-05-02 | 11                  |
| 1         | 2017-06-25 | 12                  |
| 3         | 2016-03-02 | 0                   |
| 3         | 2018-07-03 | 5                   |
+-----------+------------+---------------------+
```

```
# 窗口函数的解法
# select player_id ,event_date , sum(games_played ) over (partition by player_id
order by event_date ) as games_played_so_far from Activity ;
#基础解法
SELECT
    player_id,
    event_date,
    num as games_played_so_far
FROM
```

```
    (
        SELECT
            player_id,
            event_date,
            if(
                @player = player_id,
                @num := @num + games_played,
                @num := games_played
            ) as num,
            @player := player_id
        FROM
            (SELECT *,@num :=NULL,@player :=NULL FROM Activity ) t0
    ) t1
```

用户三日留存率

Activity table:

```
Create table If Not Exists Activity (player_id int, device_id int, event_date
date, games_played int);
Truncate table Activity;
insert into Activity (player_id, device_id, event_date, games_played) values
('1', '2', '2016-03-01', '5');
insert into Activity (player_id, device_id, event_date, games_played) values
('1', '2', '2016-03-02', '6');
insert into Activity (player_id, device_id, event_date, games_played) values
('2', '3', '2017-06-25', '1');
insert into Activity (player_id, device_id, event_date, games_played) values
('3', '1', '2016-03-02', '0');
insert into Activity (player_id, device_id, event_date, games_played) values
('3', '4', '2018-07-03', '5');
```

Result table:

```
+-----------+
| fraction  |
+-----------+
| 0.33      |
+-----------+
```

```
#三日留存 diff = 2，四日 = 3，五日 =4....
SELECT
    round(
        count(
            CASE
            WHEN diff = 2 THEN
                1
            ELSE
                NULL
            END
        ) / count(DISTINCT player_id),
        2
```

```
    ) AS fraction
FROM
    (
        SELECT
            player_id,
            event_date,
            dateDiff(
                event_date,
                min(event_date) over (
                    PARTITION BY player_id
                    ORDER BY
                        event_date ASC
                )
            ) AS diff
        FROM
            Activity
    ) t0;
```

员工薪水中位数

```
#建表
Create table If Not Exists Employee (Id int, Company varchar(255), Salary int);
Truncate table Employee;
insert into Employee (Id, Company, Salary) values ('1', 'A', '2341');
insert into Employee (Id, Company, Salary) values ('2', 'A', '341');
insert into Employee (Id, Company, Salary) values ('3', 'A', '15');
insert into Employee (Id, Company, Salary) values ('4', 'A', '15314');
insert into Employee (Id, Company, Salary) values ('5', 'A', '451');
insert into Employee (Id, Company, Salary) values ('6', 'A', '513');
insert into Employee (Id, Company, Salary) values ('7', 'B', '15');
insert into Employee (Id, Company, Salary) values ('8', 'B', '13');
insert into Employee (Id, Company, Salary) values ('9', 'B', '1154');
insert into Employee (Id, Company, Salary) values ('10', 'B', '1345');
insert into Employee (Id, Company, Salary) values ('11', 'B', '1221');
insert into Employee (Id, Company, Salary) values ('12', 'B', '234');
insert into Employee (Id, Company, Salary) values ('13', 'C', '2345');
insert into Employee (Id, Company, Salary) values ('14', 'C', '2645');
insert into Employee (Id, Company, Salary) values ('15', 'C', '2645');
insert into Employee (Id, Company, Salary) values ('16', 'C', '2652');
insert into Employee (Id, Company, Salary) values ('17', 'C', '65');
```

请编写SQL查询来查找每个公司的薪水中位数。挑战点：你是否可以在不使用任何内置的SQL函数的情况下解决此问题。

Id	Company	Salary
5	A	451
6	A	513
12	B	234
9	B	1154
14	C	2645

```
    select
        Id,
        Company,
        Salary
    from
        (select
            Id,
            Company,
            Salary,
            if(
                @Company = Company,
                @rank :=@rank +1,
                @rank := 1
            ) as rn,
            @Company :=Company,
            @Salary :=Salary
        from
            (
                select *,@Company :=null,@Salary :=null,@rank :=null from
Employee ORDER BY Company  asc , Salary asc
            ) t0
        ) t1
        INNER JOIN
    (SELECT
        COUNT(*) AS totalcount, Company AS name
    FROM
        Employee e2
    GROUP BY e2.Company) companycount ON companycount.name = t1.Company
    where rn = floor((totalcount+1)/2) or rn = floor((totalcount+2)/2)
```

给定数字的频率查询中位数

```
Create table If Not Exists Numbers (Number int, Frequency int);
Truncate table Numbers;
insert into Numbers (Number, Frequency) values ('0', '7');
insert into Numbers (Number, Frequency) values ('1', '1');
insert into Numbers (Number, Frequency) values ('2', '3');
insert into Numbers (Number, Frequency) values ('3', '1');
```

在此表中，数字为 0, 0, 0, 0, 0, 0, 0, 1, 2, 2, 2, 3，所以中位数是 (0 + 0) / 2 = 0。

+--------+
| median |
+--------|
| 0.0000 |
+--------+
请编写一个查询来查找所有数字的中位数并将结果命名为 median 。

```

```

查询员工的每三个月累计薪水

```
#表结构及数据
Create table If Not Exists Employee (Id int, Month int, Salary int);
Truncate table Employee;
```

```
insert into Employee (Id, Month, Salary) values ('1', '1', '20');
insert into Employee (Id, Month, Salary) values ('2', '1', '20');
insert into Employee (Id, Month, Salary) values ('1', '2', '30');
insert into Employee (Id, Month, Salary) values ('2', '2', '30');
insert into Employee (Id, Month, Salary) values ('3', '2', '40');
insert into Employee (Id, Month, Salary) values ('1', '3', '40');
insert into Employee (Id, Month, Salary) values ('3', '3', '60');
insert into Employee (Id, Month, Salary) values ('1', '4', '60');
insert into Employee (Id, Month, Salary) values ('3', '4', '70');
insert into Employee (Id, Month, Salary) values ('1', '7', '90');
insert into Employee (Id, Month, Salary) values ('1', '8', '90');
```

要求

> 编写 SQL 语句，对于每个员工，查询他除最近一个月（即最大月）之外，剩下每个月的近三个月的累计薪水（不足三个月也要计算）。结果请按 Id 升序，然后按 Month 降序显示。

结果

Id	Month	Salary
1	3	90
1	2	50
1	1	20
2	1	20
3	3	100
3	2	40

分析

> 两点要求
>
> 1、排除数据中当前月的记录，不参与计算
>
> 2、每三个月做累加，如果数据中每三个月数据不全，就有多少加多少。比如：如果数据中某个id最大月份为8，排除8月份后，仅有2，3，4，7月份的记录，那么，累加的窗口范围就是 2，3，4一组，7一组

```
SELECT
    B.Id AS id,
    B.Month AS month,
    (
        SELECT
            SUM(e2.Salary)
        FROM Employee e2
            WHERE
                e2.Id = B.Id AND
                -- 累加工资，要把e2表的当前记录控制在B表中当前记录近三个月即 如下条件
                e2.Month <= B.Month AND e2.Month > B.Month - 3
            ORDER BY
                e2.Month DESC  LIMIT 3
    ) AS Salary
FROM (
```

```
    --B表获取到每个员工参与累加的所有月份记录
    SELECT
        e1.Id, e1.Month, e1.Salary
    FROM
        Employee e1,
        --A表获取到每个员工当前月份为几月
        (SELECT e.Id, MAX(e.Month) AS Month FROM Employee e GROUP BY e.Id) A
        WHERE
            e1.Id = A.Id AND e1.Month < A.Month
        ORDER BY
            e1.Id ASC, e1.Month DESC
) B

#关于窗口函数的延伸 窗口函数无法实现对月份的窗口控制，即ROWS BETWEEN 不能在月份层面做控制，所以不符合题目要求，但思想可以借鉴
select Id,Month,
sum(Salary) over(partition by Id order by Month ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) Salary
from
(
    # 排除最大的一个月份，lead获取到当前月对应的下个月份，且升序排序。排到表中数据最大月份时，其对应的下个月份为0，这个时候即可排除最大月份(当前月)的记录
    select Id,Month,Salary,
    lead(Month,1,0) over(partition by Id order by Month) lm
    from Employee
)t1
where lm != 0
order by  Id,Month desc
```

统计各专业人数

```
CREATE TABLE IF NOT EXISTS student (student_id INT,student_name VARCHAR(45), gender VARCHAR(6), dept_id INT);
CREATE TABLE IF NOT EXISTS department (dept_id INT, dept_name VARCHAR(255));
Truncate table student;
insert into student (student_id, student_name, gender, dept_id) values ('1', 'Jack', 'M', '1');
insert into student (student_id, student_name, gender, dept_id) values ('2', 'Jane', 'F', '1');
insert into student (student_id, student_name, gender, dept_id) values ('3', 'Mark', 'M', '2');
Truncate table department;
insert into department (dept_id, dept_name) values ('1', 'Engineering');
insert into department (dept_id, dept_name) values ('2', 'Science');
insert into department (dept_id, dept_name) values ('3', 'Law');
```

580. 统计各专业学生人数

难度 中等 31

SQL架构 >

一所大学有 2 个数据表，分别是 ***student*** 和 ***department***，这两个表保存着每个专业的学生数据和院系数据。

写一个查询语句，查询 ***department*** 表中每个专业的学生人数（即使没有学生的专业也需列出）。

将你的查询结果按照学生人数降序排列。 如果有两个或两个以上专业有相同的学生数目，将这些部门按照部门名字的字典序从小到大排列。

student 表格如下：

```
| Column Name  | Type      |
|--------------|-----------|
| student_id   | Integer   |
| student_name | String    |
| gender       | Character |
| dept_id      | Integer   |
```

其中， student_id 是学生的学号， student_name 是学生的姓名， gender 是学生的性别， dept_id 是学生所属专业的专业编号。

department 表格如下：

```
| Column Name | Type    |
|-------------|---------|
| dept_id     | Integer |
| dept_name   | String  |
```

dept_id 是专业编号， dept_name 是专业名字。

```
| dept_name   | student_number |
|-------------|----------------|
| Engineering | 2              |
| Science     | 1              |
| Law         | 0              |
```

```
-- 1、专业学生人数 -> 专业 分组，组内计数
-- 2、组内人数降序排列，专业相同，专业名字排列
-- 3、没有学生的专业也要列出



SELECT
    dept_name,
    count(s.student_id) as student_number
FROM
    department d
```

```
LEFT JOIN
    student s
    ON d.dept_id = s.dept_id
group by
    d.dept_name
ORDER BY
    student_number DESC ,
    d.dept_name;
```

投保人额度

```
CREATE TABLE IF NOT EXISTS insurance (PID INTEGER(11), TIV_2015 NUMERIC(15,2),
TIV_2016 NUMERIC(15,2), LAT NUMERIC(5,2), LON NUMERIC(5,2) );
Truncate table insurance;
insert into insurance (PID, TIV_2015, TIV_2016, LAT, LON) values ('1', '10',
'5', '10', '10');
insert into insurance (PID, TIV_2015, TIV_2016, LAT, LON) values ('2', '20',
'20', '20', '20');
insert into insurance (PID, TIV_2015, TIV_2016, LAT, LON) values ('3', '10',
'30', '20', '20');
insert into insurance (PID, TIV_2015, TIV_2016, LAT, LON) values ('4', '10',
'40', '40', '40');
```

样例输入

```
| PID | TIV_2015 | TIV_2016 | LAT | LON |
|-----|----------|----------|-----|-----|
| 1   | 10       | 5        | 10  | 10  |
| 2   | 20       | 20       | 20  | 20  |
| 3   | 10       | 30       | 20  | 20  |
| 4   | 10       | 40       | 40  | 40  |
```

样例输出

```
| TIV_2016 |
|----------|
| 45.00    |
```

```
-- 此题关键是如何保证经纬度以唯一，以及2015投资额不唯一


-- 开窗函数
-- 1、分组计数经纬度，计数2015投资额相同的数量
-- 2、根据条件筛选（经纬度唯一，2015投资额有相等），做sum

select
    SUM(TIV_2016) TIV_2016
from
    (
        select
            TIV_2016 ,
            count(*) over(partition by TIV_2015) as a,
            count(*) over(partition by LAT,LON) as b
```

```
        from
            insurance
    ) t0
where a > 1 and b = 1



--基础解法
-- 1、分组计数经纬度，计数2015投资额相同的数量
-- 2、子查询控制经纬度和2015投资额

select
    SUM(TIV_2016) TIV_2016
from
    insurance
where
    TIV_2015 in ( select TIV_2015 from insurance group by TIV_2015 having
count(*) > 1 )
    and
    (LAT,LON) not in (select LAT,LON from insurance group by LAT,LON having
count(*)=1 )
```

好友申请 II： 谁有最多的好友

602. 好友申请 II： 谁有最多的好友

难度 中等 34

SQL架构 >

在 Facebook 或者 Twitter 这样的社交应用中，人们经常会发好友申请也会收到其他人的好友申请。

表 `request_accepted` 存储了所有好友申请通过的数据记录，其中，
requester_id 和 **accepter_id** 都是用户的编号。

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

写一个查询语句，求出谁拥有最多的好友和他拥有的好友数目。对于上面的样例数据，结果为：

id	num
3	3

```sql
-- 1、如果a，b是好友，那么a，b的好友数都+1
-- 2、想办法列转行并union成新表，accepter_id，requester_id 值互相转换
-- 3、根据requeser_id分组排序取最大值

select
    requester_id as id,
    count(*) as num
from
    (
        select
            requester_id
        from request_accepted r
        union all
        select
            r2.accepter_id  as requester_id
        from request_accepted r2
    ) t0
group by
     requester_id
order by
    num desc
limit 1
```

树节点

608. 树节点

难度 中等 42

SQL架构 >

给定一个表 `tree`，**id** 是树节点的编号， **p_id** 是它父节点的 **id** 。

```
+----+------+
| id | p_id |
+----+------+
| 1  | null |
| 2  | 1    |
| 3  | 1    |
| 4  | 2    |
| 5  | 2    |
+----+------+
```

树中每个节点属于以下三种类型之一：

- 叶子：如果这个节点没有任何孩子节点。
- 根：如果这个节点是整棵树的根，即没有父节点。
- 内部节点：如果这个节点既不是叶子节点也不是根节点。

写一个查询语句，输出所有节点的编号和节点的类型，并将结果按照节点编号排序。上面样例的结果为：

```
+----+------+
| id | Type |
+----+------+
| 1  | Root |
| 2  | Inner|
| 3  | Leaf |
| 4  | Leaf |
| 5  | Leaf |
+----+------+
```

```
-- 1、用case when 赋值 type，三个类型就需要满足三个条件
-- 2、Root条件：p_id 为 null，Inner条件：有父有子，Leaf：有父无子
-- 3、关键判断子，新增一列子字段

select
    id,
    case
        when p_id is null THEN 'Root'
        when p_id is not null and id in (select p_id from tree)             THEN
'Inner'
        else 'Leaf'
    end as Type
from
    tree;
```

```
SELECT
    atree.id,
    IF(ISNULL(atree.p_id),
        'Root',
        IF(atree.id IN (SELECT p_id FROM tree), 'Inner','Leaf')) Type
FROM
    tree atree
ORDER BY atree.id;
```

某个窗口期内 公司平均工资与部门平均工资的对比

615. 平均工资：部门与公司比较

难度 困难 31

SQL架构 >

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        |
```

```sql
-- 1、部门每月平均工资、公司每月平均工资
-- 2、分组条件为 工资发放日和部门
-- 3、注意涉及金额的比较，符号两边的值精度要一致

select
    pay_month,
    department_id,
```

```
    case when round(avg(amount),4)>totalavg then 'higher'
         when round(avg(amount),4) = totalavg then 'same'
         else 'lower' end comparison
from
    (
     select
        s.id,s.amount,
        DATE_FORMAT(s.pay_date,'%Y-%m') pay_month,
        e.department_id,
        -- 在每个月的窗口期内，求公司的平均工资
        round(avg(s.amount) over(partition by DATE_FORMAT(pay_date,'%Y-%m')),4)
totalavg
     from
        salary s
     join
        employee e
     on s.employee_id = e.employee_id
    )t0
group by department_id,pay_month
order by pay_month desc,department_id
```

学生地理报告--类似透视表行转列

618. 学生地理信息报告

难度 困难 48

SQL架构 >

一所美国大学有来自亚洲、欧洲和美洲的学生，他们的地理信息存放在如下 `student` 表中。

```
| name   | continent |
|--------|-----------|
| Jack   | America   |
| Pascal | Europe    |
| Xi     | Asia      |
| Jane   | America   |
```

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）。

对于样例输入，它的对应输出是：

```
| America | Asia | Europe |
|---------|------|--------|
| Jack    | Xi   | Pascal |
| Jane    |      |        |
```

```sql
## 关键是要找到三个条件下，可以连接起来的点，这里用了他们排序后的序号
select
    America,Asia,Europe
from
    (
        select row_number() over(order by name) as rn , name as America from
student where continent = 'America'
    ) a
    left join
    (
        select row_number() over(order by name) as rn , name as Europe from
student where continent = 'Europe'
    ) b on a.rn = b.rn
    left join
    (
        select row_number() over(order by name) as rn , name as Asia from
student where continent = 'Asia'
    ) c on a.rn = c.rn;
```

游戏用户分析 在线人数以及一日留存率

`Activity` 活动记录表

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
```

(player_id, event_date) 是此表的主键
这张表显示了某些游戏的玩家的活动情况
每一行表示一个玩家的记录，在某一天使用某个设备注销之前，登录并玩了很多游戏（可能是 0）

玩家的 **安装日期** 定义为该玩家的第一个登录日。

玩家的 **第一天留存率** 定义为：假定安装日期为 `X` 的玩家的数量为 `N` ，其中在 `X` 之后的一天重新登录的玩家数量为 `M` ， `M/N` 就是第一天留存率，**四舍五入到小数点后两位。**

编写一个 SQL 查询，报告所有安装日期、当天安装游戏的玩家数量和玩家的第一天留存率。

查询结果格式如下所示：

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

Result 表:
+------------+----------+----------------+
| install_dt | installs | Day1_retention |
+------------+----------+----------------+
| 2016-03-01 | 2        | 0.50           |
| 2017-06-25 | 1        | 0.00           |
+------------+----------+----------------+
```

```
# 先开窗求出首次安装时间install_dt,
# 再选出 event_date -install_dt =1 的就是1日留存率
# 当然求count的时候别忘记用distinct 吧play_id 去重
select
    install_dt,
    count(distinct player_id) installs,
    round(
        sum(
            if(
```

```
                datediff(event_date,install_dt)=1,
                1,
                0
            )
        )/count(
            distinct player_id
        ),
        2
    ) Day1_retention
from
    (
        select
            *,min(event_date) over(partition by player_id) install_dt
        from
            Activity
    ) t0
group  by install_dt
```