力扣高频 SQL 50 题阶段总结（四）

> 这一组题目整体非常有代表性，覆盖了 **去重统计、连续性判断、历史状态查询、前缀和、分组后过滤** 等 LeetCode SQL 中最核心、最容易反复考察的模型。本文不只给出解法，而是重点拆解 **“为什么要这样写”**，以及 **如果写错通常错在哪里**。

一、1713 – 得到连续的 id 区间（连续性问题）

题目描述

给定一张只包含整数 id 的表（可能无序、但 id 不重复），要求找出 **所有连续的 id 区间**，并返回每个区间的起始 id 和结束 id。

难点分析

- SQL 中没有 for 循环
- 无法像程序一样“逐个判断是否连续”
- 必须把“连续”转化成 **可分组的数学特征**

解题核心思路（非常关键）

如果一组数字是连续递增的，那么：

> `id - 行号（row_number）` 的结果是一个常数

这是因为：

- id 每次 +1
- row_number 也每次 +1
- 二者的差保持不变

解题步骤拆解

1. 先按 id 排序
2. 使用 `ROW_NUMBER()` 给每一行编号
3. 计算 `id - row_number` 作为分组标识
4. 按该标识分组，取最小和最大 id

参考解法（MySQL 8.0）

```
代码块
SELECT
    MIN(id) AS start_id,
    MAX(id) AS end_id
FROM (
    SELECT id, id - ROW_NUMBER() OVER (ORDER BY id) AS grp
    FROM Logs
) t
GROUP BY grp;
```

为什么这样一定正确?

- 每一个 grp 对应一段连续区间
- 一旦 id 中断，row_number 仍连续，差值立刻变化

二、1789 – 员工在职区间（状态区间问题）

题目描述

给定员工的进入日期和离开日期，要求找出员工 **连续在职的时间区间**。

本题本质

这是一个典型的 **时间轴 + 状态变化** 问题：

- 某个时间点状态从“不在职”变为“在职”
- 某个时间点再从“在职”变为“不在职”

解题关键思想

SQL 不擅长“状态切换”，但擅长：

- 排序
- 累加
- 窗口函数

因此需要把问题转化为：

> 在时间轴上，哪些区间内员工状态为“在职”

思路拆解（逻辑层面）

1. 将进入视为 +1 事件
2. 将离开视为 -1 事件
3. 按时间排序

4. 对事件做累加，判断状态是否 > 0

代码块

```sql
select employee_id, department_id
from Employee
where primary_flag = 'Y'
or employee_id
in
(
    select employee_id
    from Employee
    group by employee_id
    having count(employee_id) = 1
);
```

核心认知

- 区间问题 ≠ JOIN
- 区间问题 = **时间排序 + 状态累计**

三、610 – 判断三角形（条件判断）

题目描述

给定三条边长 x、y、z，判断是否能构成三角形。

数学基础

三角形成立的充分必要条件是：

- 任意两边之和 **严格大于** 第三边

即需要同时满足三条不等式。

SQL 解题思路

- SQL 中的 IF / CASE 非常适合表达这种 **多条件判断**
- 关键是：
  - 三个条件必须 **同时成立**

参考实现

代码块

```sql
SELECT
    x, y, z,
    IF(x + y > z AND x + z > y AND y + z > x, 'Yes', 'No') AS triangle
FROM Triangle;
```

常见错误

- 漏掉某一个不等式
- 使用 OR 而不是 AND

四、180 – 连续出现的数字

题目描述

找出在日志表中 **连续出现至少三次** 的数字。

难点分析

- 要求“连续”，而不是出现次数 ⩾ 3
- 必须考虑行与行之间的关系

解题核心模型

> **比较当前行与前几行是否相同**

解题思路

1. 按 id 排序
2. 使用 `LAG` 函数获取前 1 行、前 2 行的值
3. 判断三行是否相等

参考解法

代码块

```sql
SELECT DISTINCT num AS ConsecutiveNums
FROM (
    SELECT
        num,
        LAG(num,1) OVER (ORDER BY id) AS p1,
        LAG(num,2) OVER (ORDER BY id) AS p2
    FROM Logs
) t
```

```sql
WHERE num = p1 AND num = p2;
```

为什么不能只用 COUNT?

- COUNT 只能统计数量
- 无法保证“顺序上的连续”

五、1164 – 指定日期的商品价格（历史状态）

题目描述

查询每个商品在指定日期之前（含）最近一次变价后的价格。

本题模型

> **Latest Before Date（历史快照查询）**

解题思路拆解

1. 只保留 `change_date <= 指定日期` 的记录
2. 对每个 product_id 找到最大的 change_date
3. 通过 (product_id, change_date) 回表取 price
4. 对没有历史记录的商品，返回 NULL

关键点说明

- 不能直接 `GROUP BY product_id` 并取 price
- 必须先确定“代表行”，再取行上的字段

示例解法

代码块

```sql
SELECT p.product_id, t.price
FROM (SELECT DISTINCT product_id FROM Products) p
LEFT JOIN (
    SELECT product_id, price
    FROM Products
    WHERE (product_id, change_date) IN (
        SELECT product_id, MAX(change_date)
        FROM Products
        WHERE change_date <= '2019-08-16'
        GROUP BY product_id
```

```
    )
) t
ON p.product_id = t.product_id;
```

六、1204 – 最后一个能上车的人（前缀和）

题目描述

乘客按 turn 顺序上车，每人有体重 weight，车辆最大承重为 1000，求最后一个能上车的人。

难点分析

- 需要“累加到当前行”的概念
- SQL 无法用变量逐行累加

核心模型

> **Running Sum（前缀和）**

解题思路

1. 按 turn 排序
2. 使用窗口函数计算从第一行到当前行的体重和
3. 过滤出累计体重 ⩽ 1000 的记录
4. 取累计体重最大的那一行

推荐解法

代码块

```sql
SELECT person_name
FROM (
    SELECT person_name, SUM(weight) OVER (ORDER BY turn) AS total
    FROM Queue
) t
WHERE total <= 1000
ORDER BY total DESC
LIMIT 1;
```

关键认知

- JOIN 不能表达“过程”

- 窗口函数可以一次性看到历史

七、1907 – 统计访问次数（条件聚合）

题目描述

统计每个账号在不同状态下（如 open / closed）的访问次数。

本题模型

> **条件聚合（Conditional Aggregation）**

解题思路

- 先按 account_id 分组
- 在同一次扫描中，用 CASE 区分不同状态
- 对满足条件的行累加计数

示例解法

代码块

```sql
SELECT
    account_id,
    SUM(CASE WHEN status = 'open' THEN 1 ELSE 0 END) AS open_count,
    SUM(CASE WHEN status = 'closed' THEN 1 ELSE 0 END) AS closed_count
FROM LogInfo
GROUP BY account_id;
```

为什么不用多次查询?

- 多次查询效率低
- SQL 更擅长一次扫描完成多条件统计