# 力扣高频 SQL 50 题阶段总结（二）

本文总结的是 **LeetCode 高频 SQL 50 题**中一组**非常典型、面试出现率极高**的题目：

> **620、1251、1075、1633、1211、1193、1174、550**

这一组题的共同特点是：

- 大量出现 **比例 / 平均值 / 留存率 / 转化率**
- 强依赖 **GROUP BY + JOIN + 条件统计**
- 极易在「统计粒度」和「NULL 处理」上出错

如果说前面的 SQL 基础题是在考语法，那么这一组题 **考的是 SQL 的“业务建模能力”**。

---

## 一、620. 有趣的电影（Not Boring Movies）

### 题目描述

给定一张 `Cinema` 表：

| id | movie | description | rating |

要求：

- 找出 `id` 为 **奇数** 的电影
- `description` **不等于 'boring'**
- 按 `rating` **降序排序**

---

### 解题思路（详细）

这是一道**SQL 入门但非常“规范性”的题**，核心在于理解：

> **SQL 中数据筛选的顺序和职责划分**

我们逐条拆解题意：

**1️⃣ id 为奇数**

- 这是一个数值判断问题
- 在 SQL 中，判断奇偶最直接的方式是取模运算
- `id % 2 = 1` 表示奇数

**2️⃣ description ≠ 'boring'**

- 字符串比较一定要注意：
  - 使用 `<>` 或 `!=`
  - 区分大小写与否由数据库决定（MySQL 默认不区分）

**3️⃣ 排序条件**

- 排序一定发生在：
  - 数据已经筛选完之后
- `ORDER BY` 永远是 SQL 执行逻辑中的最后一步

这道题的意义在于训练：

- 多条件 `WHERE` 的**并列组合能力**
- 不要把过滤逻辑写进 `ORDER BY` 或子查询中

---

## 解法

代码块

```sql
SELECT id, movie, description, rating
FROM Cinema
WHERE id % 2 = 1
  AND description <> 'boring'
ORDER BY rating DESC;
```

---

## 解题思路

这是一道**纯筛选 + 排序题**，但它的意义在于：

> **让你意识到 WHERE 子句中的条件是可以并列组合的**

关键点：

1. 奇数判断：`id % 2 = 1`
2. 字符串比较：`<> 'boring'`
3. 排序一定要放在最后

---

## 解法

代码块

```sql
SELECT id, movie, description, rating
FROM Cinema
WHERE id % 2 = 1
  AND description <> 'boring'
ORDER BY rating DESC;
```

---

## 二、1251. 平均售价（Average Selling Price）

### 题目描述

给定两张表：

- `Prices(product_id, start_date, end_date, price)`
- `UnitsSold(product_id, purchase_date, units)`

要求：

> 计算每个商品的 **平均售价**。

平均售价定义为：

```
代码块
1  总销售额 / 总销量
```

如果某个商品 **没有任何销量**，则平均售价为 `0`。

---

### 解题思路（详细，重点题）

这道题是 SQL 中**非常经典的业务型统计题**，本质是：

> **时间区间 JOIN + 加权平均**

我们一步步拆解。

#### ① 明确“统计单位”

最终结果是：

```
代码块
1  每个 product_id 一行
```

所以：

- `GROUP BY product_id` 是必然的

---

### ② 正确匹配“售价”与“销量”

一个商品在不同时间段：

- 可能有 **不同的价格**
- 而销量发生在某一天

因此：

> 只有当 `purchase_date` 落在某个价格区间内，
> 这个价格才对这次销量生效

这一步决定了：

- 必须使用 **非等值 JOIN（BETWEEN）**

---

### ③ 为什么不能直接 AVG(price)？

因为：

- 不同价格对应的销量不同
- 这是一个 **加权平均问题**

正确公式是：

```
代码块
1   SUM(price * units) / SUM(units)
```

---

### ④ 为什么要 LEFT JOIN + IFNULL？

- 有些商品可能：
  - 在 `Prices` 中存在
  - 但在 `UnitsSold` 中完全没有记录

如果用 INNER JOIN：

- 这些商品会 **直接消失**

但题目要求它们的平均售价是 `0`，
因此：

- 必须保留 `Prices` 中的商品
- 用 `LEFT JOIN`
- 再用 `IFNULL` 兜底

---

## 解法

代码块

```sql
SELECT
  p.product_id,
  IFNULL(ROUND(SUM(p.price * u.units) / SUM(u.units), 2), 0) AS average_price
FROM Prices p
LEFT JOIN UnitsSold u
  ON p.product_id = u.product_id
 AND u.purchase_date BETWEEN p.start_date AND p.end_date
GROUP BY p.product_id;
```

---

## 解题思路（重点）

这道题是**典型的“时间区间 JOIN + 加权平均”问题**。

核心拆解：

1. `UnitsSold.purchase_date` 必须落在 `Prices.start_date ~ end_date`
2. 实际销售额 = `price * units`
3. 平均售价 = `SUM(price * units) / SUM(units)`
4. 要防止除 0 → `IFNULL`

---

## 解法

代码块

```sql
SELECT
  p.product_id,
  IFNULL(ROUND(SUM(p.price * u.units) / SUM(u.units), 2), 0) AS average_price
FROM Prices p
LEFT JOIN UnitsSold u
  ON p.product_id = u.product_id
 AND u.purchase_date BETWEEN p.start_date AND p.end_date
GROUP BY p.product_id;
```

## 三、1075. 项目员工 I（Project Employees I）

### 题目描述

给定两张表：

- `Project(project_id, employee_id)`
- `Employee(employee_id, experience_years)`

要求：

> 计算每个项目中员工的 **平均工龄**。

---

### 解题思路（详细）

这是一道**标准的 JOIN + GROUP BY + AVG 模板题**，但非常适合作为：

> 判断你是否真正理解了“统计维度”

#### ① 统计维度是谁？

题目要求的是：

```
每个 project_id 一行
```

因此：

- `project_id` 一定出现在 `GROUP BY`

---

#### ② 统计指标来自哪里？

- 员工的工龄存储在 `Employee` 表
- 项目和员工的关系在 `Project` 表

所以：

- 必须先 JOIN，再统计

---

#### ③ 为什么可以直接 AVG？

因为：

- 每个员工在一个项目中只算一次
- 不存在权重问题

这是**最纯粹的平均值模型**

---

## 解法

代码块

```
SELECT
  p.project_id,
  ROUND(AVG(e.experience_years), 2) AS average_years
FROM Project p
JOIN Employee e
  ON p.employee_id = e.employee_id
GROUP BY p.project_id;
```

---

## 解题思路

这是**最标准的 JOIN + GROUP BY + AVG** 模型。

关键在于：

- 平均值是 **项目维度**
- 工龄来自 `Employee`

---

## 解法

代码块

```
SELECT
  p.project_id,
  ROUND(AVG(e.experience_years), 2) AS average_years
FROM Project p
JOIN Employee e
  ON p.employee_id = e.employee_id
GROUP BY p.project_id;
```

---

## 四、1633. 各赛事的用户注册率（Percentage of Users Attended a Contest）

### 题目描述

- `Users(user_id, user_name)`
- `Register(contest_id, user_id)`

要求：

> 对每个比赛，计算：

```
代码块
1   注册人数 / 用户总人数
```

并按：

1. 百分比降序
2. contest_id 升序

---

### 解题思路（高频易错）

这是**比例类 SQL 的模板题**。

关键点：

1. 分子：某 contest 注册的 **去重用户数**
2. 分母：`Users` 表的 **总用户数**（不随 contest 变化）
3. 分母必须是一个「常量子查询」

---

### 解法

```
代码块
1   SELECT
2     r.contest_id,
3     ROUND(COUNT(DISTINCT r.user_id) / (
4       SELECT COUNT(*) FROM Users
5     ), 2) AS percentage
6   FROM Register r
7   GROUP BY r.contest_id
8   ORDER BY percentage DESC, r.contest_id;
```

## 五、1211. 查询结果的质量（Queries Quality and Percentage）

### 题目描述

`Queries(query_name, result, position, rating)`

要求：

- quality = `AVG(rating / position)`
- poor_query_percentage = 评分 < 3 的比例

---

### 解题思路

这是**一题两指标的统计题**：

1. `rating / position` → 平均
2. 条件比例 → `SUM(CASE WHEN ... THEN 1 ELSE 0 END) / COUNT(*)`

---

### 解法

代码块

```sql
SELECT
  query_name,
  ROUND(AVG(rating / position), 2) AS quality,
  ROUND(SUM(CASE WHEN rating < 3 THEN 1 ELSE 0 END) / COUNT(*) * 100, 2)
    AS poor_query_percentage
FROM Queries
GROUP BY query_name;
```

---

## 六、1193. 每月交易 I（Monthly Transactions I）

### 题目描述

`Transactions(id, country, state, amount, trans_date)`

要求按：

- 月份 + 国家
  统计：
- 总交易数
- 成功交易数
- 总金额
- 成功金额

---

### 解题思路

这是**条件聚合的经典模板题**。

关键：

- 月份提取：`DATE_FORMAT(trans_date, '%Y-%m')`
- 条件统计：`SUM(CASE WHEN state='approved' THEN ... END)`

---

### 解法

代码块

```sql
SELECT
  DATE_FORMAT(trans_date, '%Y-%m') AS month,
  country,
  COUNT(*) AS trans_count,
  SUM(state = 'approved') AS approved_count,
  SUM(amount) AS trans_total_amount,
  SUM(CASE WHEN state = 'approved' THEN amount ELSE 0 END)
    AS approved_total_amount
FROM Transactions
GROUP BY month, country;
```

---

## 七、1174. 即时食物配送 II（Immediate Food Delivery II）

### 题目描述

`Delivery(customer_id, order_date, customer_pref_delivery_date)`

要求：

> 计算 **首次订单中是“即时配送”** 的客户比例

### 解题思路（非常重要）

这是**留存 / 转化类问题的典型变形**。

拆解：

1. 每个客户的 **第一单**
2. 判断第一单是否即时配送
3. 用客户维度算比例

---

### 解法

代码块

```sql
SELECT
  ROUND(
    SUM(order_date = customer_pref_delivery_date) / COUNT(*),
  2) AS immediate_percentage
FROM (
  SELECT
    customer_id,
    MIN(order_date) AS order_date,
    MIN(customer_pref_delivery_date) AS customer_pref_delivery_date
  FROM Delivery
  GROUP BY customer_id
) t;
```

---

## 八、550. 游戏玩法分析 IV（次日留存率）

### 题目描述

`Activity(player_id, event_date, games_played)`

要求：

> 计算：首日后 **次日仍登录的玩家比例**

---

### 解题思路（重点中的重点）

这道题的核心是：

> **统计单位必须是“玩家”，不是“登录行为”**

正确模型：

1. 找每个玩家的首日
2. 判断是否存在 `首日 + 1`
3. 对玩家取平均

---

## 解法

代码块

```sql
SELECT
  ROUND(
    COUNT(DISTINCT CASE
      WHEN a.event_date = DATE_ADD(f.first_day, INTERVAL 1 DAY)
      THEN f.player_id
    END) / COUNT(DISTINCT f.player_id),
  2) AS fraction
FROM (
  SELECT player_id, MIN(event_date) AS first_day
  FROM Activity
  GROUP BY player_id
) f
LEFT JOIN Activity a
ON f.player_id = a.player_id;
```

---

# 九、阶段总结（非常重要）

这一组题，你实际上已经接触到了 **SQL 面试的核心能力**：

- 条件聚合（SUM + CASE）
- 比例 / 平均 / 留存模型
- 统计粒度控制
- LEFT JOIN + NULL 语义

一句话总结：

> **SQL 真正的难点不是语法，而是“你在统计什么对象”。**

如果你愿意，下一步我可以帮你：

- 把这 8 题抽象成 **5 个通用 SQL 模型**

- 或继续写 **高频 SQL 50 后半段系统博客**