

告别排队，Trae订阅上线，首月Pro仅\$3，开发无忧

立即体验

理解 MySQL 的分组机制：GROUP BY、SELECT、HAVING 及索引优化

Asthenian 2025-03-30 141 阅读5分钟

关注

告别排队，Trae订阅上线，首月Pro仅\$3，开发无忧

理解 MySQL 的分组机制：GROUP BY、SELECT、HAVING 及索引优化

MySQL 的 **GROUP BY** 是 SQL 中一个核心功能，用于分组统计数据。你可能已经对它的基本用法有所了解，但一些细节，比如 **SELECT** 中非聚合列的限制，或者 **HAVING** 的作用，可能还让人困惑。今天我们不仅会拆解这些机制，还会深入探讨一个更实际的问题：在 **HAVING** 中使用函数是否影响索引，以及如何优化。

一、GROUP BY 到底是怎么分组的？

简单来说，**GROUP BY** 按指定列的值将数据分成组，然后对每组应用聚合操作。就像整理一堆学生成绩单，按班级分成几组，再计算每组的平均分。

示例表：学生成绩

假设有表 **scores**：

▼ CSS 体验AI代码助手 复制代码

1	student_id class score
2	----- ----- -----
3	1 A 80
4	2 A 90
5	3 B 85

6	4	B	95	
7	5	A	70	

查询：

▼ sql

 体验AI代码助手 

复制代码

```
1 SELECT class, AVG(score)
2 FROM scores
3 GROUP BY class;
```

结果：

▼ CSS

 体验AI代码助手 

复制代码

1	class AVG(score)
2	----- -----
3	A 80
4	B 90

分组过程

1. 按 **class** 分组：数据分成 A 和 B 两组。
2. 聚合计算：对每组的 **score** 计算平均值。
3. 返回结果：每组一行，显示分组列和聚合结果。

二、为什么 **SELECT** 外的非聚合列必须分组？

如果查询写成：

▼ sql

 体验AI代码助手 

复制代码

```
1 SELECT student_id, class, AVG(score)
2 FROM scores
3 GROUP BY class;
```



在严格模式下会报错，因为 **student_id** 不是分组依据，也没有聚合函数处理。分组后每组只有一行，但 **student_id** 在 A 组有多个值（1、2、5），MySQL 无法决定显示哪个值。SQL 标准要求：**SELECT** 中非聚合列必须出现在 **GROUP BY** 中。

解决方法

- 用聚合函数： `SELECT MAX(student_id), class, AVG(score) GROUP BY class;`
- 调整分组： `GROUP BY student_id, class;` （但可能改变业务逻辑）。



三、HAVING 的作用及常见用法

HAVING 是分组后的条件过滤器。比如：

▼ sql  体验AI代码助手  复制代码

```
1 SELECT class, AVG(score)
2 FROM scores
3 GROUP BY class
4 HAVING AVG(score) > 85;
```



结果只显示平均分大于 85 的班级：

▼ objectivec  体验AI代码助手  复制代码

1	class	AVG(score)
2	-----	-----
3	B	90



互联网场景用法

1. 活跃用户：

▼ sql  体验AI代码助手  复制代码

```
1 SELECT user_id, COUNT(*) as login_count
2 FROM user_logins
3 GROUP BY user_id
4 HAVING COUNT(*) > 5;
```

2. 高消费用户：

▼ sql  体验AI代码助手  复制代码

```
1 SELECT user_id, SUM(order_amount)
2 FROM orders
```

```
3 GROUP BY user_id
4 HAVING SUM(order_amount) > 1000;
```

3. 异常检测：

▼ sql

 体验AI代码助手 

复制代码

```
1 SELECT ip_address, COUNT(*)
2 FROM api_logs
3 GROUP BY ip_address
4 HAVING COUNT(*) > 1000;
```

四、HAVING 中使用函数会影响索引吗？

你可能注意到，上面例子中 HAVING 用到了 COUNT(*) 或 SUM(order_amount) 这类聚合函数。这引发了一个关键问题：在 HAVING 中使用函数会不会导致索引失效？

索引的影响

答案是：是的，HAVING 中的聚合函数通常无法直接利用索引。原因如下：

- 聚合是计算结果：COUNT、SUM 等函数是对分组后的数据进行计算，索引只能加速数据的查找和分组（GROUP BY 部分），但无法直接优化聚合结果的过滤。
- 执行顺序：MySQL 的查询执行顺序是 FROM -> WHERE -> GROUP BY -> HAVING -> SELECT -> ORDER BY。HAVING 在分组和聚合之后执行，此时索引的作用已经局限于前面的步骤（如 WHERE 过滤或 GROUP BY 排序）。

例如：

▼ sql

 体验AI代码助手 

复制代码

```
1 SELECT user_id, SUM(order_amount)
2 FROM orders
3 WHERE order_date > '2025-01-01'
4 GROUP BY user_id
5 HAVING SUM(order_amount) > 1000;
```

- 如果 order_date 有索引，WHERE 可以利用它快速过滤数据。
- 如果 user_id 有索引，GROUP BY 可能利用它加速分组。
- 但 HAVING SUM(order_amount) > 1000 是基于聚合结果的条件，无法直接用索引优化。

验证索引使用

可以用 **EXPLAIN** 检查：

▼ sql

 体验AI代码助手 

复制代码

```
1 EXPLAIN SELECT user_id, SUM(order_amount)
2 FROM orders
3 GROUP BY user_id
4 HAVING SUM(order_amount) > 1000;
```

结果中通常不会显示 **HAVING** 使用索引，因为它是后置过滤。

五、索引优化的解决策略

既然 **HAVING** 中的函数会导致性能瓶颈，从索引优化的角度，我们可以采取以下方法：

1. 提前过滤（用 **WHERE** 替代部分 **HAVING**）

尽量把条件前移到 **WHERE**，减少分组的数据量。比如：

▼ sql

 体验AI代码助手 

复制代码

```
1 SELECT user_id, COUNT(*) as login_count
2 FROM user_logins
3 WHERE login_time > '2025-01-01'
4 GROUP BY user_id
5 HAVING COUNT(*) > 5;
```

- **WHERE login_time > '2025-01-01'** 可以用 **login_time** 索引，减少扫描行数。
- **HAVING** 只处理剩下的聚合结果。

2. 创建覆盖索引

为 **GROUP BY** 和 **WHERE** 涉及的列创建复合索引。例如：

▼ sql

 体验AI代码助手 

复制代码

```
1 CREATE INDEX idx_orders_user_date ON orders (user_id, order_date);
```

这可以加速 `GROUP BY user_id` 和 `WHERE order_date > '2025-01-01'`，间接减少 `HAVING` 的负担。

3. 物化中间结果

对于复杂查询，可以用子查询或临时表先计算聚合结果，再过滤：

▼ sql

 体验AI代码助手 

复制代码

```
1 SELECT user_id, total_amount
2 FROM (
3     SELECT user_id, SUM(order_amount) as total_amount
4     FROM orders
5     GROUP BY user_id
6 ) AS temp
7 WHERE total_amount > 1000;
```

- 子查询先完成分组和聚合。
- `WHERE` 替代 `HAVING`，可能利用物化表的索引（如果数据库支持）。

4. 避免不必要的聚合

如果业务允许，可以简化查询逻辑。比如，如果只需要知道哪些用户消费超过 1000，不一定非要用 `SUM`：

▼ sql

 体验AI代码助手 

复制代码

```
1 SELECT DISTINCT user_id
2 FROM orders
3 WHERE order_amount > 1000;
```

这避免了分组和 `HAVING`，直接用索引（如果 `order_amount` 有索引）。

5. 分区表或分片

在互联网场景下，数据量巨大时，可以按时间（如 `order_date`）或 `user_id` 分区，分而治之，减少单次查询的计算量。

六、总结

- **GROUP BY**：按列值分组，聚合统计。
- **SELECT 限制**：非聚合列需在 **GROUP BY** 中，确保结果明确。
- **HAVING**：分组后过滤，常用于统计分析。
- **索引与 HAVING**：聚合函数无法直接用索引，但可以通过提前过滤、覆盖索引、物化结果等优化。

标签： 后端

本文收录于以下专栏



MYSQL面试

专栏目录

MYSQL面试

12 订阅 · 66 篇文章

订阅

上一篇 SQL执行顺序与ON vs WHERE...

下一篇 如何为这条sql语句建立索引： ...

评论 0



登录 / 注册 即可发表评论!

暂无评论数据

目录

[收起 ^](#)

理解 MySQL 的分组机制：GROUP BY、SELECT、HAVING 及索引优化

一、GROUP BY 到底是怎么分组的？

示例表：学生成绩

分组过程

二、为什么 SELECT 外的非聚合列必须分组？

解决方法

三、HAVING 的作用及常见用法

互联网场景用法

四、HAVING 中使用函数会影响索引吗？

索引的影响

验证索引使用

五、索引优化的解决策略

1. 提前过滤（用 WHERE 替代部分 HAVING）

2. 创建覆盖索引

3. 物化中间结果

4. 避免不必要的聚合

5. 分区表或分片

六、总结

相关推荐

大数据学习（一）：HDFS

68阅读 · 0点赞

外企也半夜发布上线吗？

75阅读 · 0点赞

Java 源码 - 本地变量ThreadLocal

47阅读 · 0点赞

使用Spring Boot对接印度股票数据源：实战指南

75阅读 · 0点赞

JVM字节码详解

27阅读 · 0点赞

为你推荐

Mysql：第06章_DQL-Mysql内置函数--分组函数和分页查询

花粥之间 3年前  526  2  评论

MySQL 后端

MySQL的聚合函数该如何使用？

快乐大队长 2年前  1.3k  4  1

MySQL 后端

MySQL索引怎么用？4个点让你秒懂！

Java小叮当 4年前  794  7  评论

Java

MySQL中这些关键字的用法，佬们get到了嘛

小威要向诸佬学习呀 1年前  761  3  评论

后端

【MySQL】MySQL索引及调优

Kimizu0 1年前  141  点赞  评论

后端

SQL优化_优化分组

一只小码农正在路过 4年前  327  2  评论

MySQL

MySQL索引（六）索引优化补充，分页查询、多表查询、统计查询

鳄鱼儿 1年前  487  5  评论

数据库 MySQL 搜索引擎

简单易懂的MySQL覆盖索引、前缀索引、索引下推

沸羊羊 3年前  2.7k  13  评论

MySQL

Pandas DataFrame 实战分析：分组、合并、查询、索引与缺失值处理

Asthenian 26天前  72  点赞  评论

后端

MySQL高级进阶：索引优化

智多星云 1年前  326  1  1

后端

MySQL | GROUP BY子句使用详解

Andya 4月前  165  1  评论

后端 MySQL SQL

SQL 查询的执行顺序

emanjusaka 1年前  316  1  评论

数据库 MySQL

《MySQL技术内幕--InnoDB存储引擎》笔记--索引篇

MySQL 索引深入解析及优化策略

仰望星空下的自己

2年前

299

2

评论

后端

深入 MySQL 索引：从数据结构到具体使用

LBXX

3年前

1.1k

5

评论

MySQL

后端