MySQL 用 limit 为什么会影响性能？有什么优化方案？

Java后端编程 2025年03月29日 09:04 福建

来源：toutiao.com/article/7279396210157371904

> **微信搜索公众号：架构师指南，回复：架构师 领取资料 。**

- Mysql 的索引结构
- Mysql 的查询过程
- Mysql 的 Limit 性能问题
- 那么，有没有办法优化这个问题呢？

Limit 是一种常用的分页查询语句，它可以指定返回记录行的偏移量和最大数目。例如，下面的语句表示从 test 表中查询 val 等于4的记录，并返回第300001到第300005条记录：

```
select * from test where val=4 limit 300000,5;
```

这样的语句看起来很简单，但是在实际使用中，可能会出现性能问题。为什么呢？我们需要从 Mysql 的索引结构和查询过程来分析。

Mysql 的索引结构

Mysql 支持多种类型的索引，其中最常用的是 B+ 树索引。B+ 树索引是一种平衡多路查找树，它有以下特点：

- 树中的每个节点最多包含 m 个子节点，m 被称为 B+ 树的阶。
- 树中的每个节点最少包含 m/2（向上取整）个子节点，除了根节点和叶子节点。
- 树中的所有叶子节点都位于同一层，并且通过指针相连。
- 树中的所有非叶子节点只存储键值（索引列）和指向子节点的指针。
- 树中的所有叶子节点存储键值（索引列）和指向数据记录（聚簇索引）或者数据记录地址（非聚簇索引）的指针。

下图是一个 B+ 树索引的示例：

图片

在 Mysql 中，有两种常见的 B+ 树索引：聚簇索引和非聚簇索引。

聚簇索引是一种特殊的 B+ 树索引，它将数据记录和索引放在一起存储，也就是说，叶子节点就是数据记录。在 Mysql 中，每张表只能有一个聚簇索引，通常是主键或者唯一非空键。如果没有定义这样的键，Mysql 会自动生成一个隐藏的聚簇索引。

非聚簇索引是一种普通的 B+ 树索引，它将数据记录和索引分开存储，也就是说，叶子节点只存储键值和指向数据记录地址的指针。在 Mysql 中，每张表可以有多个非聚簇索引，通常是普通键或者唯一键。

下图是一个聚簇索引和非聚簇索引的对比：

聚簇索引

非聚簇索引

Mysql 的查询过程

当我们执行一个 SQL 查询语句时，Mysql 会根据优化器的选择，使用不同的执行计划来执行。其中，最常见的执行计划有以下几种：

- **全表扫描：** 顾名思义，就是扫描整张表的所有数据记录，逐条检查是否满足条件。这种执行计划通常在没有合适的索引或者条件过于复杂时使用。
- **索引扫描：** 也称为范围扫描，就是根据条件在索引上进行查找，并返回满足条件的记录。这种执行计划通常在有合适的索引且条件较为简单时使用。
- **索引覆盖扫描：** 也称为索引只扫描，就是根据条件在索引上进行查找，并返回满足条件的记录，但是不需要再访问数据记录，因为查询所需的所有字段都在索引中。这种执行计划通常在有合适的索引且查询字段较少时使用。
- **回表查询：** 也称为索引查找，就是根据条件在索引上进行查找，并返回满足条件的记录，然后再根据索引指针去访问数据记录，获取查询所需的其他字段。这种执行计划通常在有合适的索引但查询字段较多时使用。

下图是一个回表查询的示例：

图片

Mysql 的 Limit 性能问题

回到我们最开始的问题，Mysql 的 Limit 会影响性能吗？为什么？

答案是：会影响性能，因为 Limit 会导致 Mysql 扫描过多的数据记录或者索引记录，而且大部分扫描到的记录都是无用的。

我们以一个非聚簇索引为例，来分析一下 Limit 的影响。假设我们有一张表 test ，它有两个字段 id 和 val ，其中 id 是主键，val 是非唯一非聚簇索引。表中有 500 万条数据，val 的值从 1 到 10 随机分布。我们执行以下语句：

```
select * from test where val=4 limit 300000,5;
```

这条语句的意思是查询 val 等于 4 的记录，并返回第 300001 到第 300005 条记录。Mysql 会怎么执行呢？

首先，Mysql 会选择 val 索引作为执行计划，因为它可以缩小查询范围。然后，Mysql 会从 val 索引的根节点开始查找，沿着 B+ 树向下搜索，直到找到第一个 val 等于 4 的叶子节点。接着，Mysql 会沿着叶子节点的指针向右移动，扫描所有 val 等于 4 的叶子节点，并记录它们对应的 id 值和数据记录地址。

由于我们要返回第 300001 到第 300005 条记录，所以 Mysql 必须扫描至少 300005 个叶子节点，才能确定哪些是我们需要的。这就导致了大量的随机 I/O 操作，在磁盘上读取索引页。

接下来，Mysql 还要根据叶子节点指向的数据记录地址，去访问数据页，获取查询所需的所有字段。由于我们要返回所有字段（ `select *` ），所以 Mysql 必须访问至少 300005 次数据页，才能获取到完整的数据记录。这又导致了大量的随机 I/O 操作，在磁盘上读取数据页。

最后，Mysql 还要对扫描到的数据记录进行排序和过滤，抛弃前面 300000 条无用的记录，只保留后面 5 条有用的记录。这就导致了大量的 CPU 和内存消耗，在内存中进行排序和过滤。

综上所述，Mysql 在执行这条语句时，需要做以下操作：

- 扫描至少 300005 个索引页
- 访问至少 300005 次数据页
- 排序和过滤至少 300005 条数据记录

这些操作都是非常耗时和耗资源和时间的浪费。为了返回 5 条有用的记录，Mysql 不得不扫描和访问大量的无用的记录。这就是 Limit 会影响性能的原因。

那么，有没有办法优化这个问题呢？

答案是：有，但是需要根据具体的情况来选择合适的方法。下面，我们介绍几种常见的优化方法：

使用索引覆盖扫描。

如果我们只需要查询部分字段，而不是所有字段，我们可以尝试使用索引覆盖扫描，也就是让查询所需的所有字段都在索引中，这样就不需要再访问数据页，减少了随机 I/O 操作。

例如，如果我们只需要查询 id 和 val 字段，我们可以执行以下语句：

```
select id,val from test where val=4 limit 300000,5;
```

这样，Mysql 只需要扫描索引页，而不需要访问数据页，提高了查询效率。

使用子查询。

如果我们不能使用索引覆盖扫描，或者查询字段较多，我们可以尝试使用子查询，也就是先用一个子查询找出我们需要的记录的 id 值，然后再用一个主查询根据 id 值获取其他字段。

例如，我们可以执行以下语句：

```
select * from test where id in (select id from test where val=4 limit 300000,5);
```

这样，Mysql 先执行子查询，在 val 索引上进行范围扫描，并返回 5 个 id 值。然后，Mysql 再执行主查询，在 id 索引上进行点查找，并返回所有字段。这样，Mysql 只需要扫描 5 个数据页，而不是 300005 个数据页，提高了查询效率。

使用分区表。

如果我们的表非常大，或者数据分布不均匀，我们可以尝试使用分区表，也就是将一张大表分成多个小表，并按照某个字段或者范围进行划分。这样，Mysql 可以根据条件只访问部分分区表，而不是整张表，减少了扫描和访问的数据量。

例如，如果我们按照 val 字段将 test 表分成 10 个分区表（test_1 到 test_10），每个分区表只存储 val 等于某个值的记录，我们可以执行以下语句：

```
select * from test_4 limit 300000,5;
```

这样，Mysql 只需要访问 test_4 这个分区表，而不需要访问其他分区表，提高了查询效率。

.....END.....

资料链接

清华学姐自学的Linux笔记，天花板级别！
新版鸟哥Linux私房菜资料
阿里大佬总结的《图解Java》火了，完整版PDF开放下载！
Alibaba官方上线！SpringBoot+SpringCloud全彩指南
国内最强的SpringBoot+Vue全栈项目天花板，不接受反驳！

架构师指南
专注分享程序员架构师技术、Java后端、系统架构、微服务架构、分布式架...
1篇原创内容

公众号