京东面试：mysql深度分页 严重影响性能？根本原因是什么？如何优化？

原创 尼恩架构团队 技术自由圈 2025年05月11日 10:52 湖北

尼恩说在前面：

在40岁老架构师 尼恩的**读者交流群**(50+)中，最近有小伙伴拿到了一线互联网企业如得物、阿里、滴滴、极兔、有赞、shein 希音、shopee、百度、网易的面试资格，遇到很多很重要的面试题：

> mysql 深度翻页太慢，如何解决？
>
> 深度分页为什么会影响性能？有什么优化方案？
>
> mysql深度分页 严重影响性能？根本原因是什么？如何优化？

前几天 小伙伴面试 京东，遇到了这个问题。但是由于 没有回答好，导致面试挂了。

小伙伴面试完了之后，来求助尼恩。

遇到mysql 深度翻页这个问题，该如何才能回答得很漂亮，才能 让面试官刮目相看、口水直流。

所以，尼恩给大家做一下系统化、体系化的梳理，使得大家内力猛增，可以充分展示一下大家雄厚的 “技术肌肉”，**让面试官爱到 “不能自已、口水直流”**，然后实现”offer直提”。

当然，这道面试题，以及参考答案，也会收入咱们的 《尼恩Java面试宝典》V145版本PDF集群，供后面的小伙伴参考，提升大家的 3高 架构、设计、开发水平。

> 最新《尼恩 架构笔记》《尼恩高并发三部曲》《尼恩Java面试宝典》的PDF，请关注本公众号【技术自由圈】获取，后台回复：领电子书

单表场景，limit深度分页存在的严重性能问题

大家的业务接口，常常是分页接口。

这样的接口，如果碰到深度分页，都会有慢sql性能问题， 而且会引发 非常严重的 线上 故障。

一个小伙伴反馈，他们公司今年3月份时候，线上发生一次大事故：

- 第一阶段：后端服务器发生宕机，所有接口超时。
- 第二阶段：宕机半小时后，又自动恢复正常。
- 第三阶段：但是过了2小时，又再次发生宕机。

通过接口日志定位，发现MySQL数据库无法响应服务器，出现 大量的 深度翻页的 慢sql：

- 当 接口 数据 被刷到7000多页，偏移量（offset）高达20w多。
- 每当这条SQL执行时，数据库CPU直接打满。
- 查询时间超过1分钟才有响应。

总之，由于慢查询导致数据库CPU使用率爆满，其他业务的数据库请求无法得到及时响应，两个结果：

- 接口超时。
- 最后，拖垮主服务器。

所以说，老架构师尼恩提示大家一定当心：limit深度分页存在的严重性能问题。

MySQL Limit 语法格式

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 。

先来看看 MySQL Limit 语法格式。

分页查询时，我们会在 LIMIT 后面传两个参数，一个是偏移量（offset），一个是获取的条数（rows）。

```
SELECT column1, column2, ...
FROM table_name
WHERE condition
LIMIT rows  OFFSET offset;
```

或者

```
SELECT * FROM table LIMIT [offset,] rows | rows OFFSET offset
SELECT column1, column2, ...
FROM table_name
WHERE condition
LIMIT [offset,] rows;
```

下面有两个例子：

- `select * from xxx limit M,N`

- `select * from xxx limit N offset M`

两个例子代表的意思是一样的：返回从第 M 开始（不包括这一行）之后的 N 行数据。

当偏移量很小时，查询速度很快，但是当 `offset` 很大时，查询速度就会变慢。

假设有一张 300w 条数据的表，对其进行分页查询， 下面是大概的 速度对比：

```
select * from user  where sex = 'm' order by age  limit 1, 10  // 32.8ms
select * from user  where sex = 'm' order by age  limit 10, 10 // 34.2ms
select * from user  where sex = 'm' order by age  limit 100, 10 // 35.4ms
select * from user  where sex = 'm' order by age  limit 1000, 10 // 39.6ms
select * from user  where sex = 'm' order by age  limit 10000, 10 // 5660ms
select * from user  where sex = 'm' order by age  limit 100000, 10 // 61.4 秒
select * from user  where sex = 'm' order by age  limit 1000000, 10 // 273 秒
```

可以看到，随着偏移量（`offset`）的增加，查询时间变得越长。

上例的数据：

- 当偏移的起始位置超过10万时，分页查询的时间超过61秒。
- 当偏移量超过100万时，查询时间竟然长达273秒。

对于普通的业务而言，超过1秒的查询是绝对不可以忍受的。

从上例中，我们可以总结出：LIMIT分页查询的时间与偏移量值成正比。

当偏移量越大时，查询时间越长。

这种情况，会随着业务的增加，数据的增多，会越发的明显。

问题: 为什么 mysql 深度分页会很慢？

mysql 服务端架构 包括两层，server层和存储引擎层

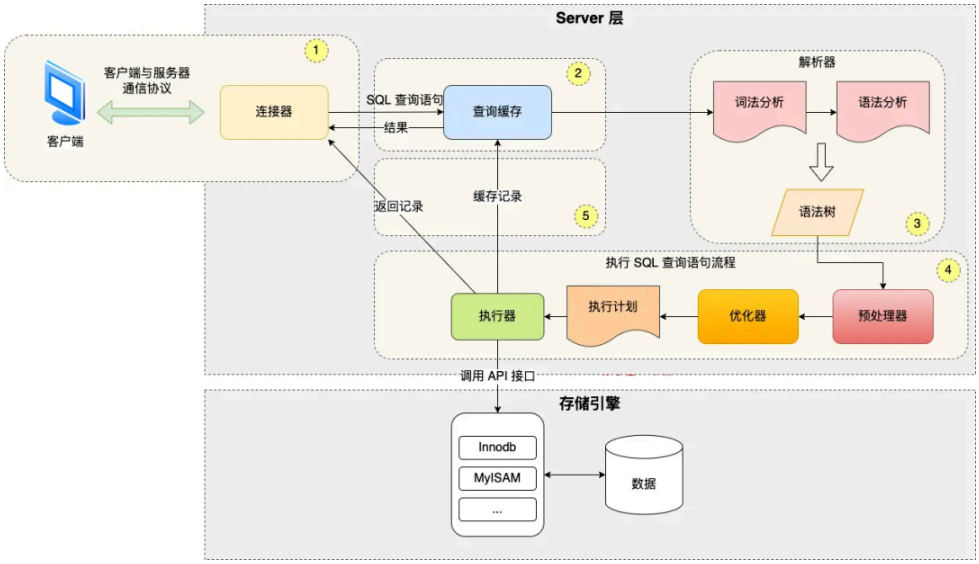
server层：查询缓存，解析sql语句生成语法树，执行sql。

在执行sql中包括预处理器，优化器和执行器。

- 预处理器：将查询字段展开（如select * 展开为具体字段）并检查字段是否合法
- 优化器：指定sql执行计划，如选择合适的索引
- 执行器：与存储引擎层交互，执行sql语句

Engine层：存储引擎层 如InnoDB和MyISAM。

以InnoDB为例，访问B+树数据结构获取记录（聚簇索引，二级索引等的访问都在存储引擎层）



limit在深度翻页场景下变成了： 全表扫描+ 文件排序 filesort

limit 是执行在server 层，而不是innodb层。

也就是说， 在server层 需要获取到 全部的 limit_cout 的结果，在发送给客户端时，才会进行limit操作。

比如，如下sql

```
select * from user  where sex = 'm' order by age  limit 1000000, 10 // 273 秒
```

server 层 在执行器调存储引擎api获取到一条数据时，会查看数据是否是第1000000 以后条数据，如果不是，就不会发送到客户端，只进行limit_cout 计数。

server 层 直到10001才会发送到客户端。也就是说，执行 limit m n语句的场景下， Engine层 实际上也会访问前m条数据，然后返回后n条数据。

正是因为 Engine层 limit会扫描每条记录，因此如果我们查询的字段需要回表扫描，每一次查询都会拿着age列的二级索引查到的主键值去回表，limit 1000000 就会回表1000000 次，效率极低。

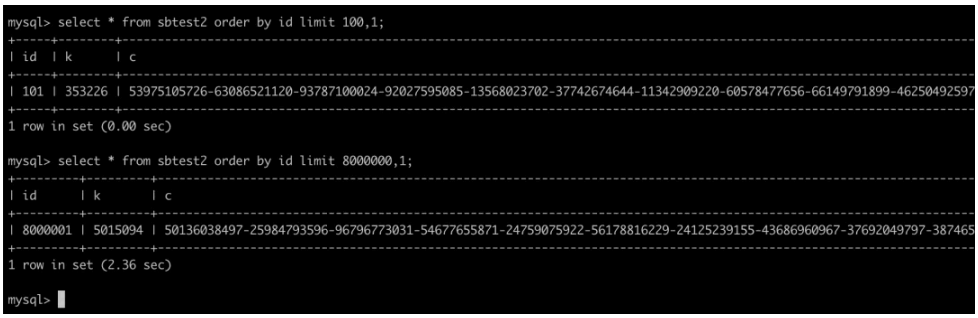
所以如果我们使用explain查看查询计划：

```
explain select * from user  where sex = 'm' order by age  limit 1000000, 10
```

其往往不会走age索引，而是全表扫描+filesort，为什么？

因为 server 层优化器认为选择age索引的效率，甚至不如全表扫描+文件排序filesort。

下面是来自网络的一个小伙伴的对比， 先看一下实际的效果，看看这个“性能的陷阱”是什么。



两个语句的内容都非常简单，差别只在 limit 的部分，第一个语句跳过的行数很少，第二个语句跳过的行数很多，结果是两个语句的执行时间差了至少 200 倍。

可以看到，当跳过的行数大幅度增长时，SQL 语句的执行时间也会快速增长，原因其实比较简单：在处理 limit M,N 的时候，MySQL 会先拿到 M+N 行结果数据，然后再丢弃 M 行数据，展示之后剩下的 N 行数据。

所以上图的第二个语句实际上扫描了 800 多万行数据，然后丢弃了 800 万行数据，只展示之后的 1 行结果。

所以，当翻页靠后时，查询会变得很慢，因为随着偏移量的增加，我们需要排序和扫描的非目标行数据也会越来越多，这些数据再扫描后都会被丢弃。

而 server 层优化器认为选择age索引的效率，甚至不如全表扫描 + 文件排序filesort。 而 全表扫描 ， 本身就是最慢的。

> 45岁老架构师 尼恩提示： 全表扫描 ， 本身就是最慢的。

为什么呢？需要从 Mysql 的索引结构和查询过程来分析。

MYSQL 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：

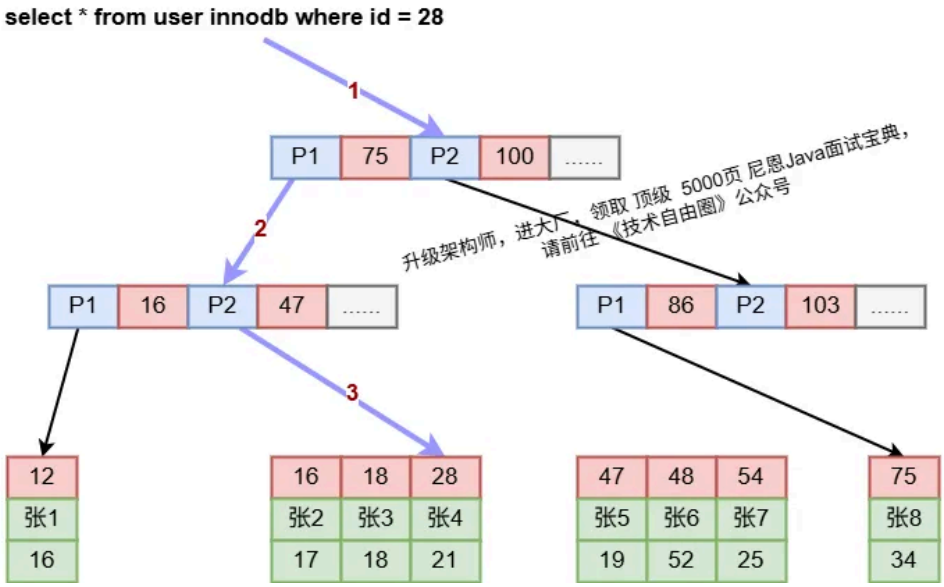
聚族索引

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。

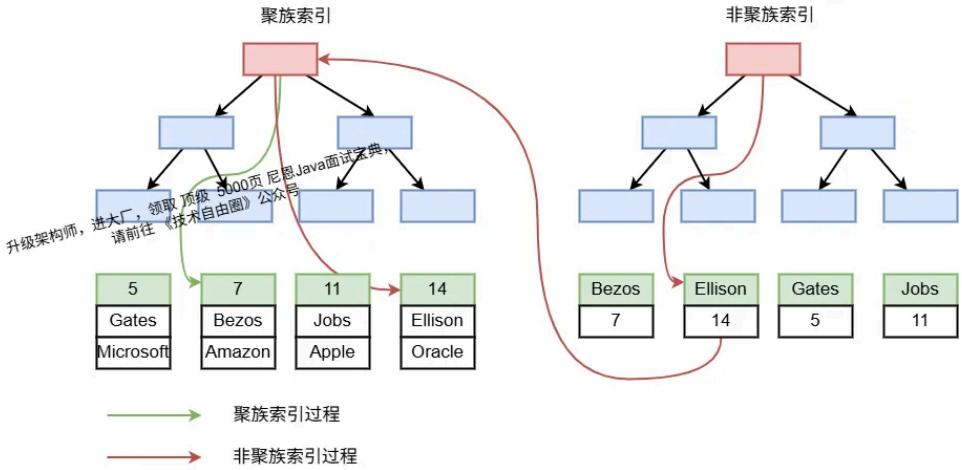
下图是一个聚簇索引：



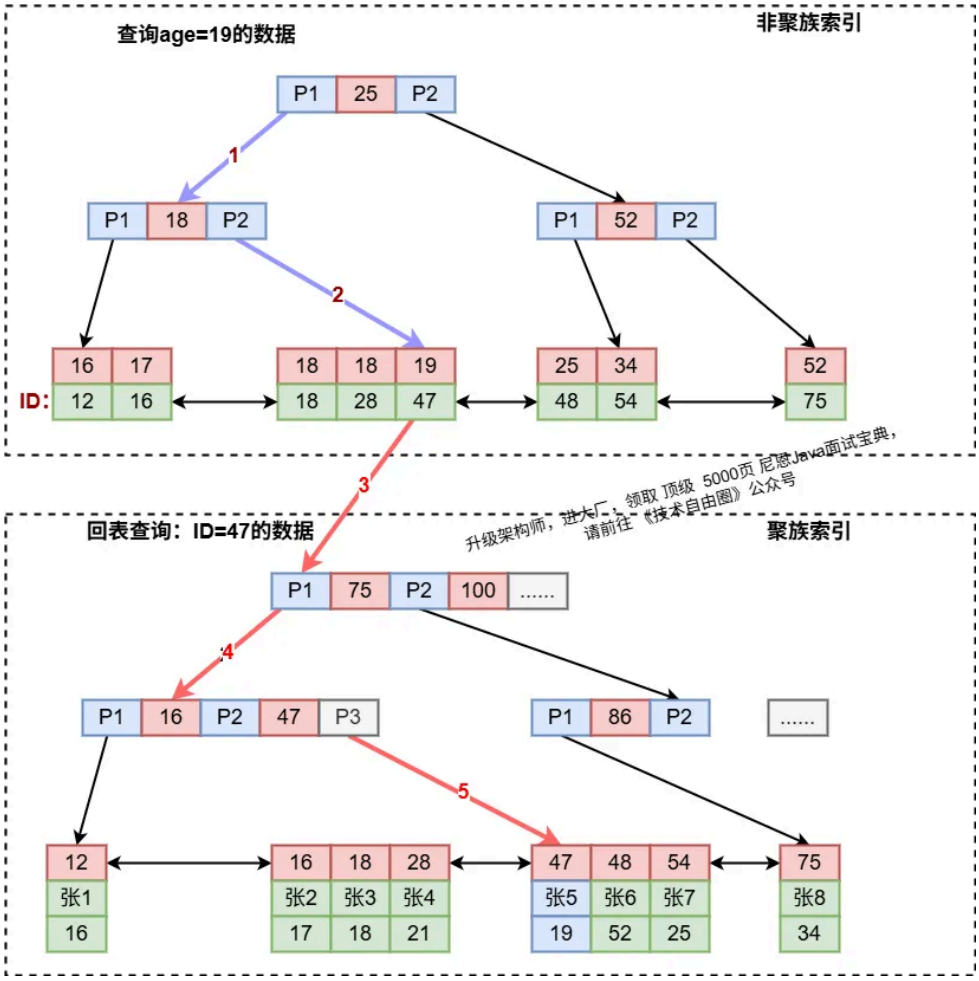
非聚族索引

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。



下图是使用非聚族索引查询的案例



全表扫描、 索引扫描、 回表查询、索引覆盖扫描的对比

当执行一个 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
- 索引扫描
- 回表查询
- 索引覆盖扫描
- **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　时使用。

```
SELECT * FROM user WHERE phone='13812345678';
```

若没有 对phone 字段建立索引，数据库会逐行扫描全部用户数据，检查每一行的手机号是否符合条件。

- **索引扫描：**

就是根据条件， 在索引上进行查找，并返回满足条件的记录。

简单的说：就是 通过 **索引树结构** 快速定位目标数据。

索引扫描 比 全表扫描性能高，为啥呢？

B+树是一种高度平衡的多叉树结构，其**树高通常仅为3-4层**， 通过B+树 快速定位目标路径。

例如查询age=20 从根节点出发，逐层比较节点键值，最终定位到叶子节点中的年龄=20的记录 。仅需访问3-4个磁盘块（对应树高），无需遍历全部数据

全表扫描 的话， 需读取所有数据页（如用户表中所有用户记录的物理存储块），数据量大时I/O次数剧增 。

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。

```
SELECT * FROM user WHERE phone='13812345678';
```

若 对phone 字段建立索引，数据库会逐行 phone 字段 索引的 B+树 进行 快速定位 扫描 ，而不需要 全表扫描 了。

- **回表查询： **

首先 根据条件在 二级索引 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 主键索引（聚族索引） 访问数据记录，获取查询所需的其他字段。

原因：SELECT的 字段 存在 非索引列， 而 二级索引叶子节点存储的是主键值而非数据地址， 所以先 通过二级索引找到主键后，**需回到 主键索引树 查找完整数据**。

假设执行

```
SELECT name, phone FROM user WHERE age=20
```

若 对age字段建立索引， 而 name, phone 没有索引， 就需要回表查询。

回表查询 降低性能， 因为 多一次索引树查询。 需要通过下面的 索引覆盖 来进行优化。

- **索引覆盖扫描： **

根据条件在索引上进行查找，并通过 索引直接 返回满足条件的记录，但是不需要再访问 主键索引树 ，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。

假设执行

```
SELECT name, phone FROM user WHERE age=20
```

优化方案：建立(age, name, phone)联合索引 → 实现索引覆盖扫描。

全表扫描、 索引扫描、索引覆盖扫描、 回表查询 的性能对比：

类型	扫描对象	是否需要回表	性能
全表扫描	整张表数据	否	最差（大数据量时）
索引扫描	索引树	视SELECT字段而定	中等
索引覆盖扫描	索引树	否	最优
回表查询	索引树 + 主键索引树	是	较差

MYSQL 回表查询

关键是回表查询，那什么是回表查询呢？

什么是回表查询：

MySQL回表查询是指在使用非聚簇索引检索数据时，索引未能完全覆盖查询所需字段，需要根据索引记录的主键值回到聚簇索引（主键索引）中二次查询完整数据行的过程。

回表查询原因：

InnoDB中，聚簇索引的叶子节点存储数据行，而非聚簇索引叶子节点仅存储主键值。

若通过非聚簇索引查询非索引字段（如：索引为name，但查询SELECT *），需先扫描非聚簇索引获取主键，再通过主键查询聚簇索引获取完整数据。

回表查询影响：

回表增加磁盘I/O次数，降低查询效率，在大数据量或高并发场景下尤为明显。

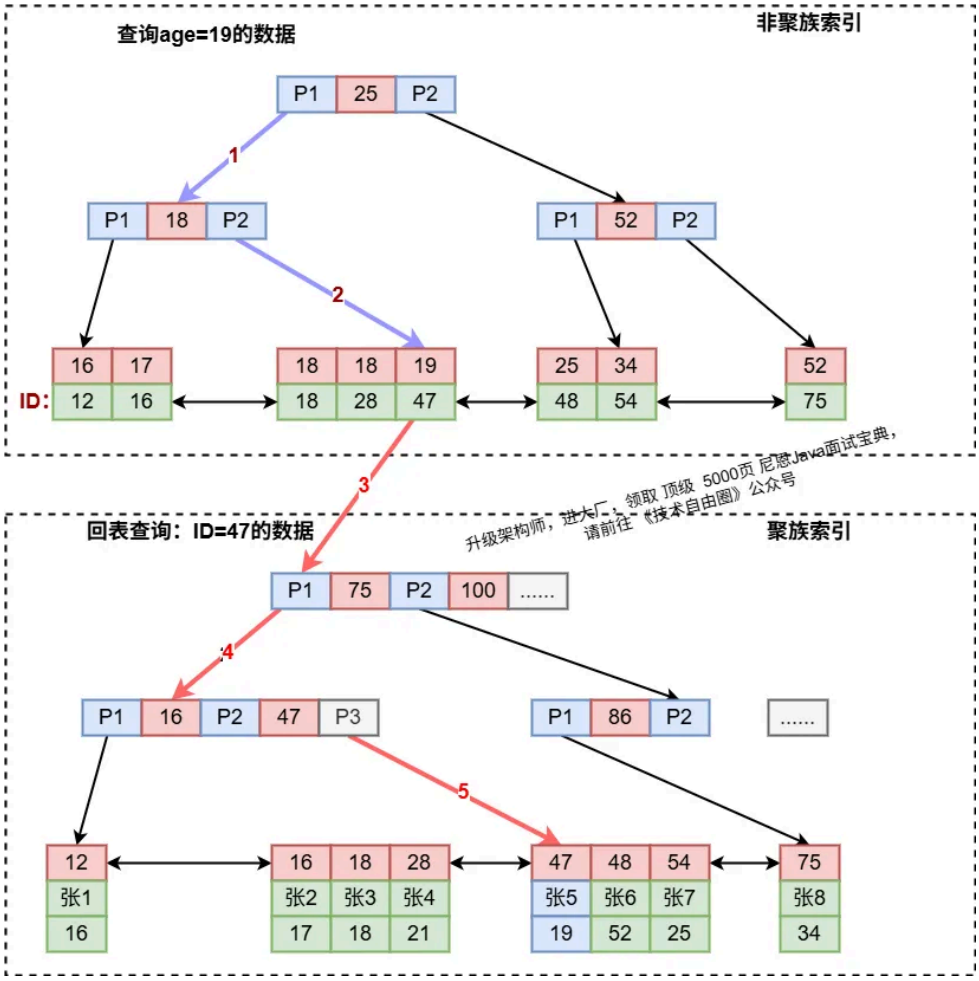
回表查询优化方式：

可通过覆盖索引（索引包含查询字段）避免回表。

例如，索引包含(name, age)时，查询SELECT age可直接从索引获取数据，无需回表。合理设计索引结构可显著提升查询性能。

通过一张图，解释下回表查询，下图：那么数据库server层 通过 Engine 利用 age字段的 非聚集索引，查到age=19的数据的id 为47。

server层 再通过 Engine 利用 id 上的聚集索引，快速地找到47 对应的员工recode记录"张5"，这就导致了回表查询。



上图案例磁盘IO数：非聚族索引3次+获取记录（聚族索引）回表3次

MYSQL Limit 深度分页 性能问题 与优化

回到最开始的问题，Mysql 的 Limit 会影响性能吗？为什么？

答案是 会影响性能，因为：

- 浅层次的维度： 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。
- 深层次的维度： 当 Limit 抛弃的数据量太大 的时候， server 层的 优化器认为 索引扫描 的效率 甚至不如全表扫描 + 文件排序filesort，于是 将 索引扫描优化为 全表扫描。 而一个大表的 全表扫描 ， 本身就是很慢的。

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，介绍几种常见的优化方法：

- 使用索引覆盖扫描
- 使用子查询
- 标签记录法

- 使用分区表

方法一：使用索引覆盖扫描

如果只需要查询部分字段，而不是所有字段，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，如果只需要查询 id 和 val 字段，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。

方法二：使用子查询

如果不能使用索引覆盖扫描，或者查询字段较多，可以尝试使用子查询。

也就是先用一个子查询找出需要的记录的 id 值，然后再用一个主查询根据 id 值获取其他字段。

例如，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。

方法三：标签记录法

就是标记一下上次查询到哪一条了，下次再来查的时候，从该条开始往下扫描。

就好像看书一样，上次看到哪里了，你就折叠一下或者夹个书签，下次来看的时候，直接就翻到了。

实现方案呢就是记录上次查询到的记录id，在这次查询时把id作为查询条件带入

```
select  id,name,balance FROM account where id > 100000 limit 10;
```

这样的话，后面无论翻多少页，性能都会不错的，因为命中了id索引。

但是这种方式有局限性：**需要一种类似连续自增的字段。**

方法四：使用分区表

如果表非常大，或者数据分布不均匀，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，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。

遇到问题，找老架构师取经

借助此文，尼恩给解密了一个高薪的 秘诀，大家可以 放手一试。**保证 屡试不爽，涨薪 100%-200%。**

后面，尼恩java面试宝典回录成视频， 给大家打造一套进大厂的塔尖视频。

通过这个问题的深度回答，可以充分展示一下大家雄厚的 “技术肌肉”，**让面试官爱到 “不能自已、口水直流”**，然后实现”offer直提”。

在面试之前，建议大家系统化的刷一波 5000页《尼恩Java面试宝典PDF》，里边有大量的大厂真题、面试难题、架构难题。

很多小伙伴刷完后， 吊打面试官， 大厂横着走。

在刷题过程中，如果有啥问题，大家可以来 找 40岁老架构师尼恩交流。

另外，如果没有面试机会，**可以找尼恩来改简历、做帮扶。**

遇到职业难题，找老架构取经， 可以省去太多的折腾，省去太多的弯路。

尼恩指导了大量的小伙伴上岸，前段时间，**刚指导一个 32岁 高中生，冲大厂成功 , 年薪 50W，逆天改命！！！。**

狠狠卷，实现 “offer自由” 很容易的， 前段时间一个武汉的跟着尼恩卷了2年的小伙伴， 在极度严寒/痛苦被裁的环境下， offer拿到手软， 实现真正的 “offer自由” 。

狠狠卷，实现 “offer自由” 很容易的， 前段时间一个武汉的跟着尼恩卷了2年的小伙伴， 在极度严寒/痛苦被裁的环境下， offer拿到手软， 实现真正的 “offer自由” 。

冲大厂 案例： 全网顶尖、高薪案例， 进大厂拿高薪， 实现薪酬腾飞、人生逆袭

阿里+美团offer：25岁 屡战屡败 绝望至极。找尼恩转架构升级，1个月拿到阿里+美团offer，逆天改命年薪 50W

阿里offer：6年一本 不想 混小厂了。狠卷1年 拿到 得物 + 阿里 offer ， 彻底上岸 ，逆天改命

字节 offer：3年经验 ， 足足折腾1年后绝望了，找尼恩陪跑， 2个月 逆天改命 ，拿到字节&携程 offer

小米offer：7年 普通小二本，冲 小米 成功，年薪60W， 吊打一大票 985/211，狠卷1年，足足涨了50%，人生逆袭

拼多多offer：2年经验60W破全网记录，顶尖案例， 2年经验年薪60W，3大厂offer， 双非一本秒杀985、秒杀211，逆天改命

美团offer：被裁后涨80%，大赚了。从小厂被裁，拿4大厂offer（申通、顺丰、携程、美团），大涨80%，26岁小伙被裁赚翻了

逆天大涨：暴涨200%，29岁/7年/双非一本 ， 从13K涨到 37K ，如何做到的？

逆天改命：27岁被裁2月，转P6降维攻击，2个月提 JD/PDD 两大offer，时来运转，人生翻盘!! 大逆袭!!

急救上岸：29岁（golang）被裁3月，转架构降维打击，收3个大厂offer， 年薪60W，逆天改命

大龄逆袭的案例： 大龄被裁，快速上岸的，远离没有 offer 的焦虑、恐慌

47岁超级大龄，被裁员后 找尼恩辅导收 2个offer，一个40多W。 35岁之后，只要 技术好，还是有饭吃，关键是找对方向，找对路子

大龄不难：39岁/15年老码农，15天时间40W上岸，管一个team，不用去 铁人三项了！

武汉收5个offer：34岁的CRUD小伙被裁， 尼恩陪跑12天， 收 5个offer，拿到30K，逆涨7K，逆天改命

上岸奇迹：中厂大龄34岁，被裁8月收一大厂offer， 年薪65W，转架构后逆天改命!

100W 年薪 天花板 案例 ，他们 如何 实现薪酬腾飞、人生逆袭？

年薪100W的底层逻辑： 大厂被裁，他们两个，如何实现年薪百万？

年薪100W：40岁小伙，被裁6个月，猛卷3月，100W逆袭 ，秘诀：升级首席架构/总架构

最新的100W案例：环境太糟，如何升 P8级，年入100W？