京东二面：分库分表后翻页100万条，怎么设计？答对这题直接给P7！

原创 小北架构师团队 程序员江小北 2025年03月21日 08:30

一、引言

如果你已经有过几轮面试经验，肯定会遇到过类似的问题。

> **面试官会问你：“如果数据量特别大的时候，分页查询慢该怎么办？”**

我们都知道，数据库分页查询很常见，尤其是展示列表的功能，基本上每个系统都有。

但是，如果你做过大流量、高并发的系统，就会知道，**当数据量上来后，分页查询的性能问题**真的是一大痛点。

数据量小的时候，分页查询没问题，但是当表里有上千万条数据时，翻一翻第几页，查询就开始卡了，慢得让人想砸电脑。

那面试官提这个问题的时候，肯定希望听到你对于**分页查询慢**的根本原因是什么，知道你是怎么解决这个问题的，以及**分库分表后深度分页的问题该如何解决**？

所以，小北今天就用最简单的语言，带大家逐步分析这个问题，看看怎么从根本上解决这个痛点。

最重要的是，**让面试官觉得你是个能打硬仗的技术高手**，给你点个赞，甚至直接发出offer！

> **文末附有让面试官狂赞的答案！**

二、问题分析

> **为什么分页查询随着翻页的深入，会变得越来越慢。**

其实，问题的根本就在于:
第一**数据量太大**
第二**数据库处理分页的方法太笨**

你以为LIMIT 100000,10是直接跳过后10万条？太天真了！

数据库的真实操作：

- 第一步： 把整张表的数据全捞出来（全表扫描），按年龄排好序（文件排序）。

- 第二步： 吭哧吭哧数到第100010条，再给你返回最后10条。

> **相当于：让你从新华字典第1页开始翻，翻到第1000页才找到字，谁能不炸？**

最坑爹环节：回表查数据
如果用了普通索引（比如按年龄建的索引）：

- **先查索引**：按年龄找到对应的主键ID（快速）
- **再回表**：用ID去主键索引里捞完整数据（慢！）

> **10万次回表 = 10万次IO操作，不卡你卡谁？**

再说另一个常见的情况——**排序**。

大多数时候，分页查询都会带有排序，比如按时间、按ID排序。

数据库不仅要查数据，还得根据你的排序要求重新排一次，特别是在数据量大的时候，排序的开销就变得非常大。

所以，翻越几百页的时候，你的查询可能就开始慢得像蜗牛。

单表场景 limit 深度分页 的优化方法

核心思路： 绕过全表扫描，直接定位到目标数据！

方案一：子查询分页

```
-- 先查索引定位ID，再捞数据（54毫秒搞定！）
SELECT * FROM user
WHERE id >= (SELECT id FROM user ORDER BY age LIMIT 100000, 1)
LIMIT 10;
```

原理： 用覆盖索引快速找到第100000条的ID，直接从这个ID开始拿数据，跳过前面10万次回表。

缺点是，不适用于结果集不以ID连续自增的分页场景。

在复杂分页场景，往往需要通过过滤条件，筛选到符合条件的ID，此时的ID是离散且不连续的。如果使用上述的方式，并不能筛选出目标数据。

方案二：JOIN联表黑科技

```
SELECT * FROM user t1
JOIN (SELECT id FROM user ORDER BY age LIMIT 100000,10) t2
ON t1.id = t2.id;
```

原理： 先用索引快速拿到10个目标ID，再一次性联表查完整数据，减少回表次数。

方案三：索引覆盖

索引覆盖（Index Covering）是指一个查询可以完全通过索引来执行，而无需通过回表来查询其他字段数据。

例如：

```
ALTER TABLE user ADD INDEX idx_age_name(age, name);  -- 查询+排序全走索引
SELECT age, name FROM user ORDER BY age LIMIT 100000,10;  -- 0.1秒！
```

精髓： 索引里直接存了所有要查的字段，不用回表，直接起飞！

四、分库分表后，翻页为什么更慢了？

1. 分库分表的翻页逻辑

假设订单表分了3个库，每个库分了2张表（共6张表），按用户ID分片。
当你执行：

```
SELECT * FROM orders ORDER BY create_time DESC LIMIT 1000000, 10;
```

你以为数据库的操作：
智能跳过100万条，从6张表各拿10条，合并完事？

实际上的操作：

1. **每张表都老老实实查100万+10条数据**（共600万+60条）。
2. **把所有数据汇总到内存，重新排序**（600万条数据排序，内存直接炸穿）。
3. **最后忍痛扔掉前100万条，给你10条结果**。

结果： 查一次耗时10秒+，数据库CPU 100%！

---

2. 分库分表翻页的存在的3个问题

- **1：数据分散，全局排序难**
  各分片数据独立排序，合并后可能乱序，必须全量捞数据重排。
- **2：深分页=分片全量扫描**
  每张表都要查 `offset + limit` 条数据，性能随分片数量指数级下降。
- **3：内存归并压力大**
  100万条数据 × 6个分片 = 600万条数据在内存排序，分分钟OOM！

一句话总结： 分库分表后，翻页越深，死得越惨！

二、3种解决分库分表深度翻页方案

方案1：禁止跳页（青铜方案）

核心思想： 别让用户随便跳页，只能一页一页翻！

实现方法：

1. **第一页查询：**

```
-- 按时间倒序，拿前10条
SELECT * FROM orders
WHERE user_id = 123
ORDER BY create_time DESC
LIMIT 10;
```

1. **翻下一页：**

```
-- 记住上一页最后一条的时间
SELECT * FROM orders
WHERE user_id = 123
AND create_time < '2023-10-01 12:00:00'  -- 上一页最后一条的时间
ORDER BY create_time DESC
LIMIT 10;
```

优点：

- 性能：每页查询只扫索引的10条，0回表。
- 内存：无需全量排序。

缺点：

- 用户不能跳页（比如从第1页直接跳到第100页）。
- 适合Feed流场景（如朋友圈、抖音），不适合后台管理系统。

方案2：二次查询法（黄金方案）

核心思想： 把分库分表的“大海捞针”，变成“精准狙击”！

实现步骤：

1. **第一轮查询：每张分片查缩小范围的数据**

```
-- 每张分片查 (offset / 分片数量) + limit 条
SELECT create_time FROM orders
ORDER BY create_time DESC
LIMIT 33334, 10;  -- 假设总offset=100万，分6个分片：100万/6 ≈ 166666
```

1. **确定全局最小时间戳**：
   从所有分片结果中，找到最小的 `create_time` （比如 `2023-09-20 08:00:00` ）。
2. **第二轮查询：根据最小时间戳查全量数据**

```
SELECT * FROM orders
WHERE create_time >= '2023-09-20 08:00:00'
ORDER BY create_time DESC
LIMIT 10;
```

优点：

- 避免全量数据排序，性能提升10倍+。
- 支持跳页查询（如直接从100万页开始查）。

缺点：

- 需要两次查询，逻辑复杂。
- 极端情况下可能有误差（需业务容忍）。

方案3：ES+HBase核弹方案（王者方案）

核心思想： 让专业的人干专业的事！

- **ES**：负责海量数据搜索+分页（倒排索引碾压数据库）。
- **HBase**：负责存储原始数据（高并发读取无压力）。

架构图：

实现步骤：

1. **写入时**：订单数据同时写MySQL（分库分表）、ES、HBase。
2. **查询时**：

```
GET /orders/_search
{
  "query": { "match_all": {} },
  "sort": [{"create_time": "desc"}],
  "from": 1000000,
  "size": 10
}
```

```
List<Order> orders = es.search(...); // 从ES拿到10个ID
List<Order> details = hbase.batchGet(orders); // 从HBase拿详情
```

- **Step2：用ES返回的ID，去HBase批量查数据**。
- **Step1：用ES查分页**（只查ID和排序字段）。

优点：

- 分页性能碾压数据库，百万级数据毫秒响应。
- 支持复杂搜索条件（ES的强项）。

缺点：

- 架构复杂度高，成本飙升（ES集群要钱，HBase要运维）。
- 数据一致性难保证（延迟可能秒级）。

三、面试怎么答？

1. 面试官要什么？

- **原理理解**：知道分库分表后翻页的痛点（数据分散、归并排序）。
- **方案灵活**：根据场景选方案（禁止跳页、二次查询、ES+HBase）。
- **实战经验**：遇到过真实问题，用过二次查询或ES。

2. 标准答案模板

“分库分表后深度分页的难点在于全局排序和内存压力。
我们有三种方案：

1. **禁止跳页**：适合C端Feed流，用连续查询代替跳页。
2. **二次查询法**：通过两次查询缩小范围，适合管理后台。
3. **ES+HBase**：扛住亿级数据分页，适合高并发大厂场景。
   在实际的场景中，订单查询需要支持搜索条件，我们最终用ES+HBase，性能从10秒降到50毫秒。”

加分的骚操作：

- 画架构图（分库分表+ES+HBase数据流向）。
- 给性能对比数据（ES分页 vs 数据库分页）。
- 提一致性解决方案（监听MySQL Binlog同步到ES）。

五、总结

分库分表后的深度分页，本质是 **“分布式数据排序”** 的难题。

- **百万以内数据**：二次查询法性价比最高。
- **高并发大厂场景**：ES+HBase是唯一选择。
- **千万别硬刚**： `LIMIT 1000000,10` 就是自杀式操作！

最后一句忠告：
面试被问分页，先拍桌子喊出“禁止跳页”，再掏出ES，面试官绝对眼前一亮！

近期技术热文

count(*)、count(1)哪个更快？面试必问：通宵整理的十道经典MySQL必问面试题

腾讯三面：40亿个QQ号，如何用1GB内存处理？

第3版：互联网大厂面试题

包括 Java 集合、JVM、多线程、并发编程、设计模式、算法调优、Spring全家桶、Java、MyBatis、ZooKeeper、Dubbo、Elasticsearch、Memcached、MongoDB、Redis、MySQL、RabbitMQ、Kafka、Linux、Netty、Tomcat、Python、HTML、CSS、Vue、React、JavaScript、Android 大数据、阿里巴巴等大厂面试题等、等技术栈！

阅读原文： 高清 7701页大厂面试题 PDF

阅读原文