天天写 order by，你知道Mysql底层执行流程吗?

原创 不才陈某 码猿技术专栏 4月12日

点击蓝色字关注我们

前言

- 在实际的开发中一定会碰到根据某个字段进行排序后来显示结果的需求，但是你真的理解 `order by` 在 Mysql 底层是如何执行的吗?
- 假设你要查询城市是 苏州 的所有人名字，并且按照姓名进行排序返回前 1000 个人的姓名、年龄，这条 sql 语句应该如何写?
- 首先创建一张用户表，sql 语句如下:

```
CREATE TABLE user (
  id int(11) NOT NULL,
  city varchar(16) NOT NULL,
  name varchar(16) NOT NULL,
  age int(11) NOT NULL,
  PRIMARY KEY (id),
  KEY city (city)
) ENGINE=InnoDB;
```

- 则上述需求的 sql 查询语句如下:

```
select city,name,age from user where city='苏州' order by name limit 1000;
```

- 这条 sql 查询语句相信大家都能写出来，但是你了解它在 Mysql 底层的执行流程吗? 今天陈某来大家聊一聊这条 sql 语句是如何执行的以及有什么参数会影响执行的流程。
- 本篇文章分为如下几个部分进行详细的阐述:

1. **全字段排序**
2. **rowid 排序**
3. **全字段排序 VS rowid 排序**
4. **如何避免排序**

全字段排序

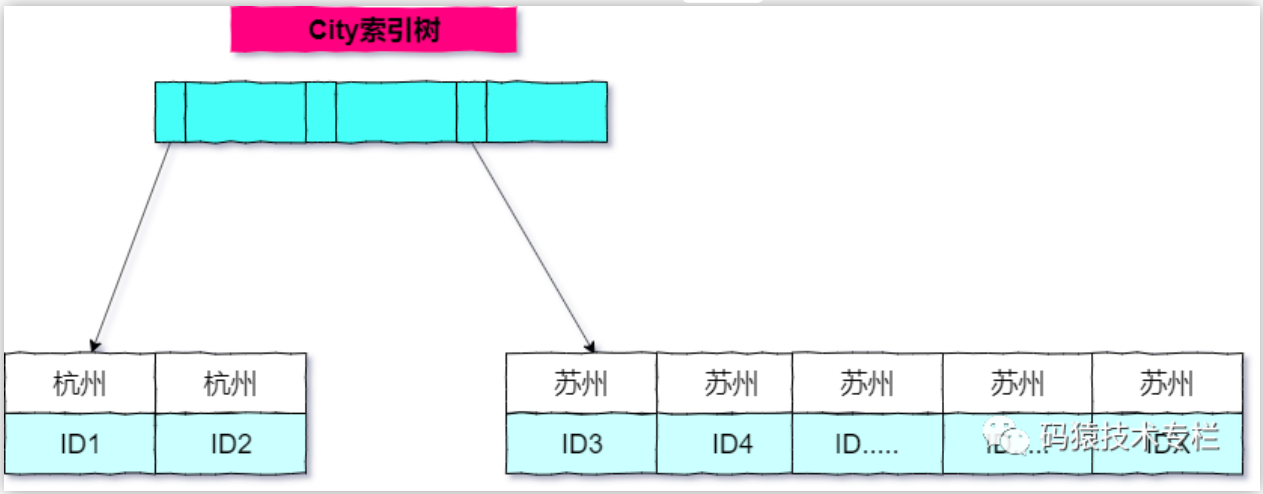
- 前面聊过索引能够避免全表扫描，因此我们给 `city` 这个字段上添加了索引，当然城市的字段很小，不用考虑字符串的索引问题，之前有写过一篇关于如何给字符串的加索引的文章，有不了解朋友看一下这篇文章:**Mysql 性能优化：如何给字符串加索引?**
- 此时用 `Explain` 来分析一下的这条查询语句的执行情况，结果如下图：

```
1  EXPLAIN select city,name,age from user where city='苏州' order by name limit 1000;
```

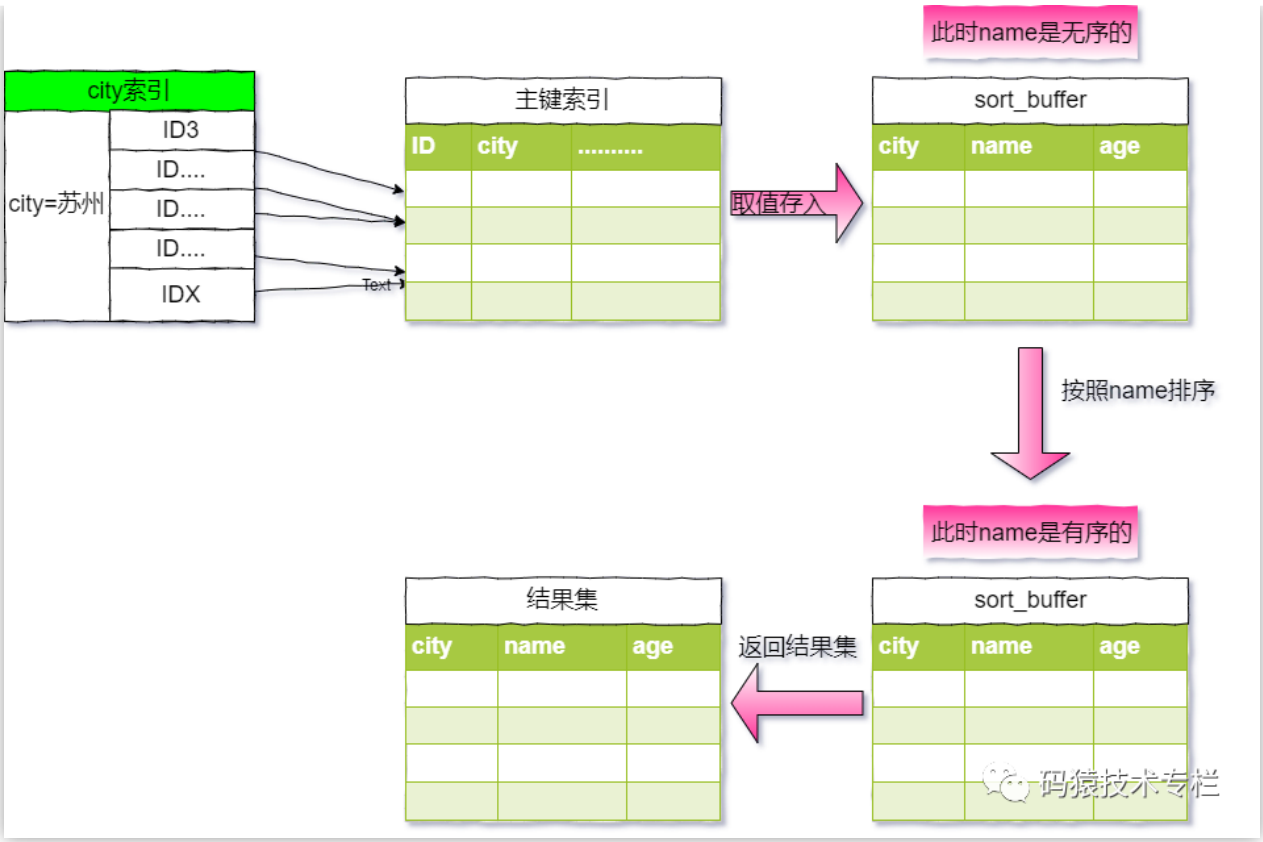
信息 | 结果 1 | 剖析 | 状态

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	user	ref	city	city	66	const	1	Using index condition; Using where; Using filesort

- `Extra` 这个字段中的 `Using filesort` 表示的就是需要排序，MySQL 会给每个线程分配一块内存用于排序，称为 `sort_buffer` 。
- 既然使用了索引进行查询，我们来简单的画一下 `city` 这棵索引树的结构，如下图：



- 从上图可以看出，满足 `city='苏州'` 是从 `ID3` 到 `IDX` 这些记录。
- 通常情况下，此条 sql 语句执行流程如下：
  1. 初始化 sort_buffer，确定放入 name、city、age 这三个字段。
  2. 从索引 city 找到第一个满足 `city='苏州'` 条件的 `主键id` ，也就是图中的 `ID3` 。
  3. 到 `主键id索引` 取出整行，取 `name` 、 `city` 、 `age` 三个字段的值，存入 `sort_buffer` 中。
  4. 从索引 `city` 取下一个记录的主键 id。
  5. 重复步骤 3、4 直到 city 的值不满足查询条件为止，对应的主键 id 也就是图中的 `IDX` 。
  6. 对 `sort_buffer` 中的数据按照字段 `name` 做快速排序。
  7. 按照排序结果取前 1000 行返回给客户端。
- 我们称这个排序过程为 `全字段排序` ，执行的流程图如下：



- 图中 按name排序 这个动作，可能在内存中完成，也可能需要使用外部排序，这取决于排序所需的内存和参数 sort_buffer_size 。
- sort_buffer_size ：就是 MySQL 为排序开辟的内存（sort_buffer）的大小。如果要排序的数据量小于 sort_buffer_size，排序就在内存中完成。但如果排序数据量太大，内存放不下，则不得不利用 磁盘临时文件 辅助排序。

rowid 排序

- 在上面这个算法过程里面，只对原表的数据读了一遍，剩下的操作都是在 sort_buffer 和 临时文件 中执行的。**但这个算法有一个问题，就是如果查询要返回的字段很多的话，那么 sort_buffer 里面要放的字段数太多，这样内存里能够同时放下的行数很少，要分成很多个临时文件，排序的性能会很差。**
- 所以如果单行很大，这个方法效率不够好。
- 我们可以修改一个 max_length_for_sort_data 这个参数使其使用另外一种算法。max_length_for_sort_data，是 MySQL 中专门控制用于排序的行数据的长度的一个参数。它的意思是，如果单行的长度超过这个值，MySQL 就认为单行太大，要换一个算法。
- city 、 name 、 age 这三个字段的定义总长度是 36 ，我把 max_length_for_sort_data 设置为 16，我们再来看看计算过程有什么改变。设置的 sql 语句如下：

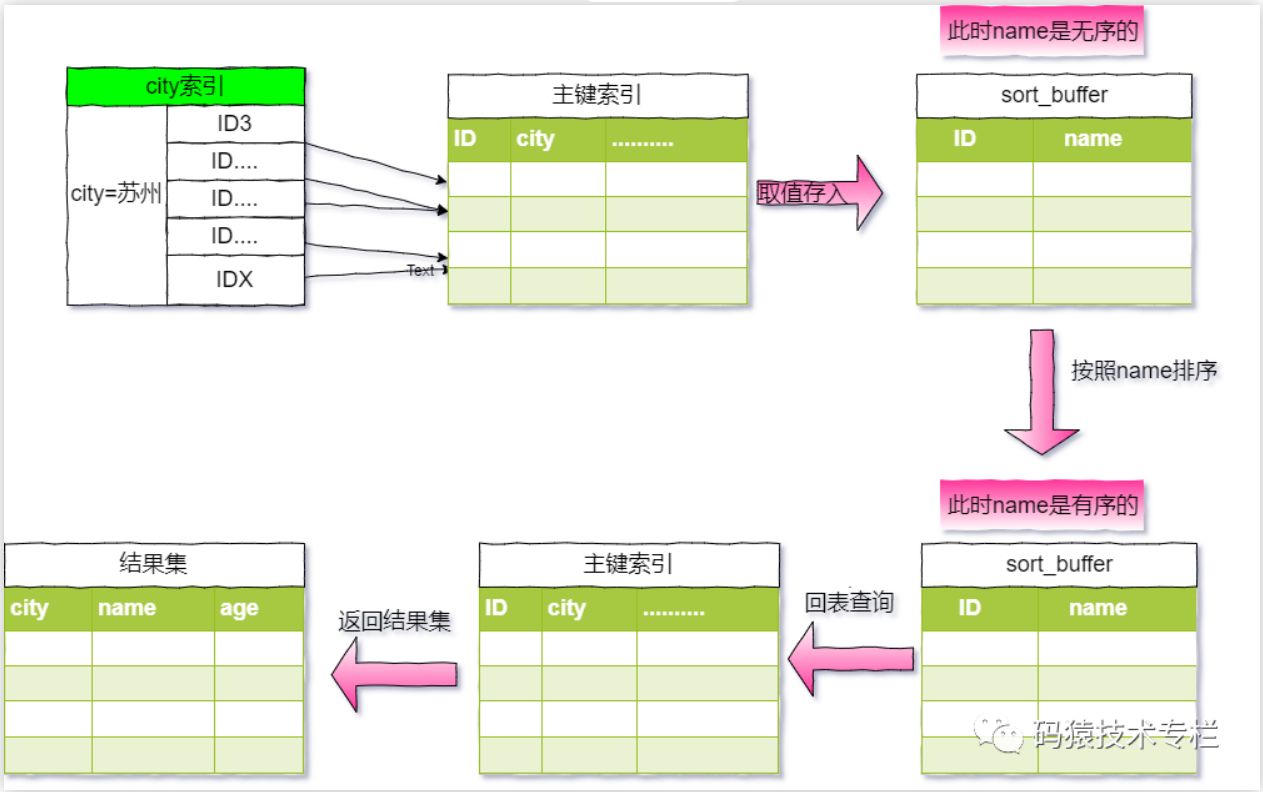
```
SET max_length_for_sort_data = 16;
```

- 新的算法放入 sort_buffer 的字段，只有要排序的列（即 name 字段）和主键 id。

- 但这时，排序的结果就因为少了 city 和 age 字段的值，不能直接返回了，整个执行流程就变成如下所示的样子：

  1. 初始化 `sort_buffer` ，确定放入两个字段，即 `name` 和 `id` 。
  2. 从索引 city 找到第一个满足 `city='苏州'` 条件的 `主键id` ，也就是图中的 `ID3` 。
  3. 到 `主键id索引` 取出整行，取 name、id 这两个字段，存入 sort_buffer 中。
  4. 从索引 `city` 取下一个记录的主键 id。
  5. 重复步骤 3、4 直到 city 的值不满足查询条件为止，对应的主键 id 也就是图中的 `IDX` 。
  6. 对 `sort_buffer` 中的数据按照字段 `name` 做快速排序。
  7. 遍历排序结果，取前 1000 行，并按照 id 的值回到原表中取出 city、name 和 age 三个字段返回给客户端。

- 这个执行流程的示意图如下，我把它称为 `rowid排序` 。



- **对比 `全字段排序` ， `rowid排序` 多了一次 `回表查询` ，即是多了 `第7步` 的查询主键索引树。**

全字段排序 VS rowid 排序

- 如果 MySQL 实在是担心排序内存太小，会影响排序效率，才会采用 rowid 排序算法，这样排序过程中一次可以排序更多行，但是需要再回到原表去取数据。
- 如果 MySQL 认为内存足够大，会优先选择全字段排序，把需要的字段都放到 sort_buffer 中，这样排序后就会直接从内存里面返回查询结果了，不用再回到原表去取数据。

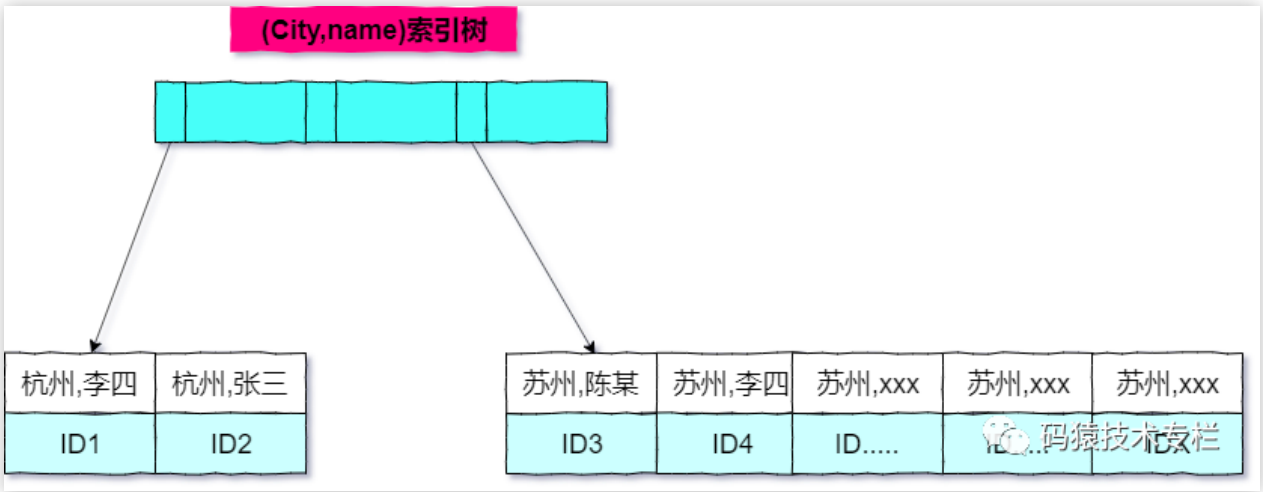
- 这也就体现了 MySQL 的一个设计思想：**如果内存够，就要多利用内存，尽量减少磁盘访问。**
- **对于 InnoDB 表来说，rowid 排序会要求回表多造成磁盘读，因此不会被优先选择。**

如何避免排序

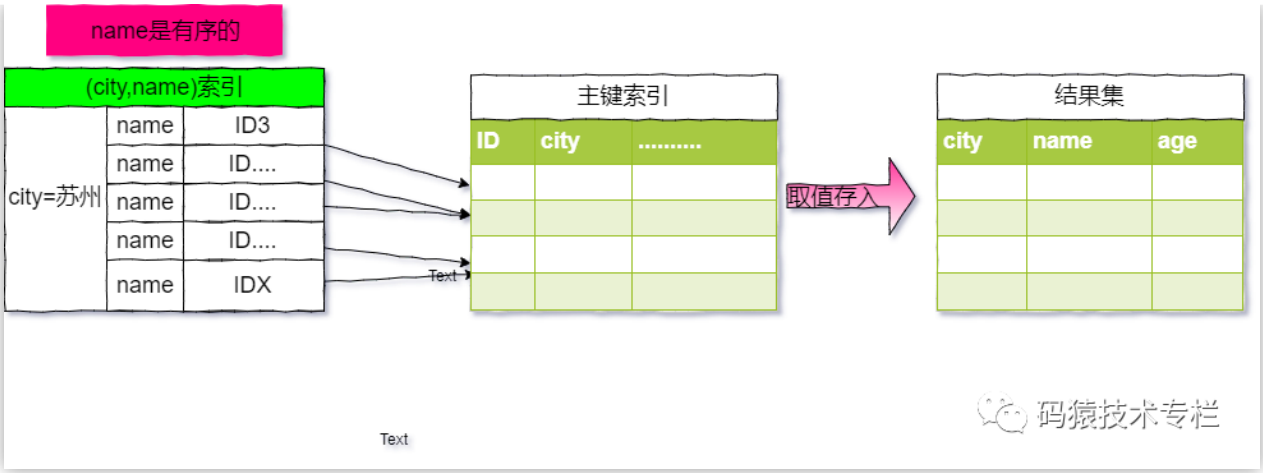
- 其实，并不是所有的 `order by` 语句，都需要排序操作的。从上面分析的执行过程，我们可以看到，MySQL 之所以需要生成临时表，并且在临时表上做排序操作，其原因是**原来的数据都是无序的。**
- 如果能够保证从 `city` 这个索引上取出来的行，天然就是按照 name 递增排序的话，是不是就可以不用再排序了呢?
- 因此想到了联合索引，创建 `(city,name)` 联合索引，sql 语句如下：

```
alter table user add index city_user(city, name);
```

- 此时的索引树如下：



- 在这个索引里面，我们依然可以用树搜索的方式定位到第一个满足 `city='苏州'` 的记录，并且额外确保了，接下来按顺序取“下一条记录”的遍历过程中，只要 city 的值是苏州，name 的值就一定是有序的。
- 按照上图，整个查询的流程如下：
  1. 从索引(city,name)找到第一个满足 city='苏州'条件的主键 id。
  2. 到主键 id 索引取出整行，取 name、city、age 三个字段的值，作为结果集的一部分直接返回。
  3. 从索引(city,name)取下一个记录主键 id。
  4. 重复步骤 2、3，直到查到第 1000 条记录，或者是不满足 city='苏州'条件时循环结束。
- 对应的流程图如下：



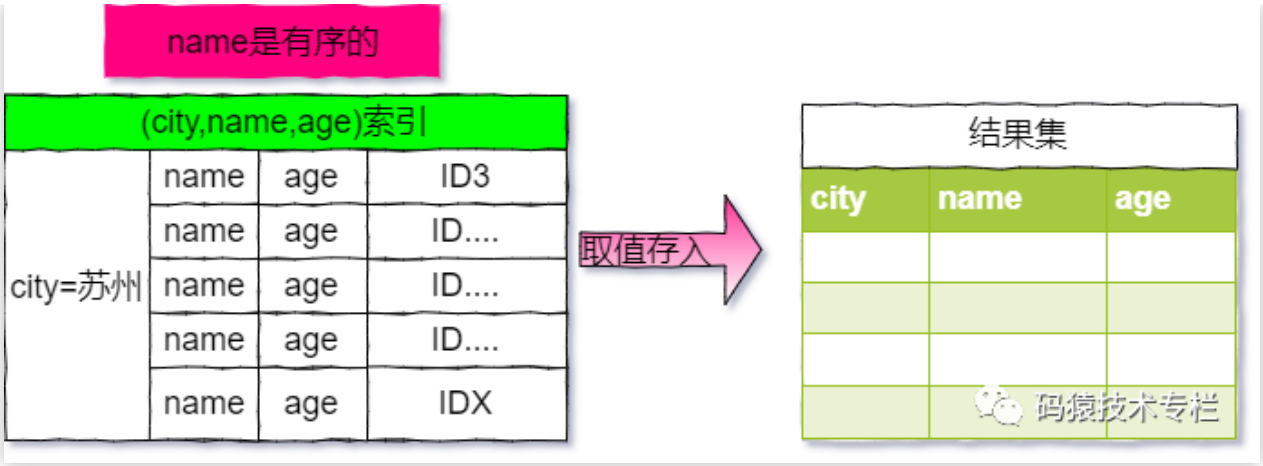
- 可以看到，这个查询过程不需要临时表，也不需要排序。接下来，我们用 explain 的结果来印证一下。

- 从图中可以看到， `Extra` 字段中没有 `Using filesort` 了，也就是不需要排序了。而且由于 `(city,name)` 这个联合索引本身有序，**所以这个查询也不用把 4000 行全都读一遍，只要找到满足条件的前 1000 条记录就可以退出了**。也就是说，在我们这个例子里，只需要扫描 1000 次。
- 难道仅仅这样就能满足了？此条查询语句是否能再优化呢?

- 朋友们还记得覆盖索引吗？覆盖索引的好处就是能够避免再次回表查询，不了解的朋友们可以看一下陈某之前写的文章：**Mysql 性能优化：如何使用覆盖索引？**。
- 我们创建 `(city,name,age)` 联合索引，这样在执行上面的查询语句就能使用覆盖索引了，避免了回表查询了，sql 语句如下：

```
alter table user add index city_user_age(city, name, age);
```

- 此时执行流程图如下：



- **当然，覆盖索引能够提升效率，但是维护索引也是需要代价的，因此还需要权衡使用。**

总结

- 今天这篇文章，我和你介绍了 MySQL 里面 `order by` 语句的几种算法流程。
- 在开发系统的时候，你总是不可避免地会使用到 order by 语句。心里要清楚每个语句的排序逻辑是怎么实现的，还要能够分析出在最坏情况下，每个语句的执行对系统资源的消耗，这样才能做到下笔如有神，不犯低级错误。

文章留言区

往期推荐

一条SQL查询语句是如何执行的?

Mysql性能优化：为什么要用覆盖索引?

Mysql性能优化：什么是索引下推?

Mysql中的三类锁，你知道吗?

Mysql性能优化：如何给字符串加索引?

Mysql性能优化：为什么count(*)这么慢?