# 16 order by是怎么工作的

表定义是这样的

CREATE TABLE `t` (

`id` int(11) NOT NULL,

`city` varchar(16) NOT NULL,

`name` varchar(16) NOT NULL,

`age` int(11) NOT NULL,

`addr` varchar(128) DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `city` (`city`)

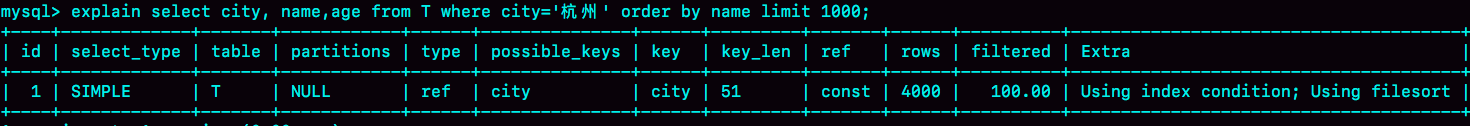
) ENGINE=InnoDB;

SQL语句这么写

select city,name,age from t where city='杭州' order by name limit 1000 ;

全字段排序

为了避免全表扫描，需要在city字段上加索引



Extra 这个字段中的“Use 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”.

索引示意图

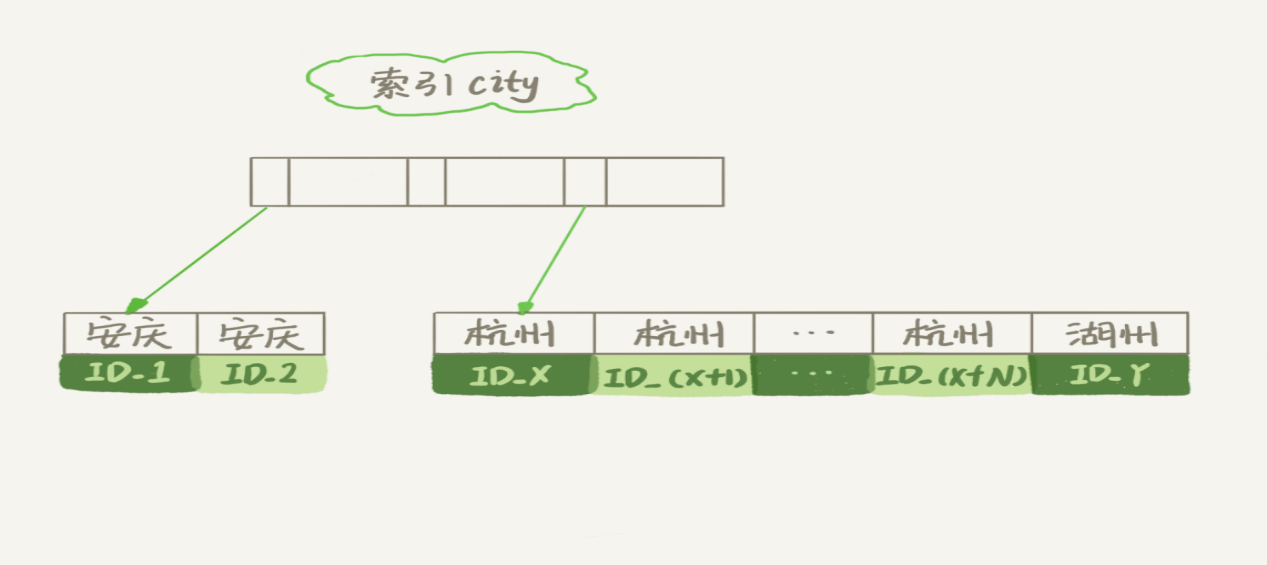


图2 city 字段的索引示意图

从图中可以看到，满足city=’杭州’条件的行，是从ID\_X到ID\_(X+N)的这些记录

通常情况，

1. 初始化sort\_buffer,确定放入name,city,age三个字段
2. 从索引city找到第一个满足city=’杭州’的主键id,也就是图中的ID\_X
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ID\_Y
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

暂时称这个流程为“全字段排序”，执行流程图如下

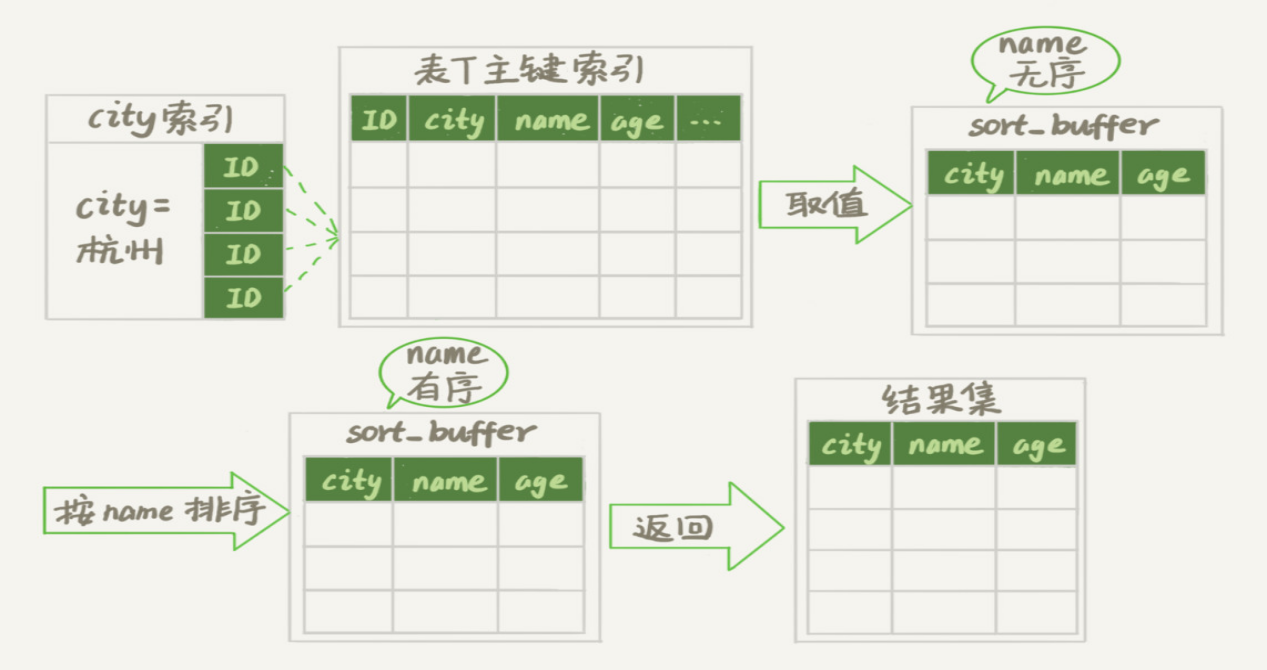


图3 全字段排序

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。如果排序的数量小于sort\_buffer\_size,排序就在内存中完成。但如果排序数量太大，内存放不下，则不得不利用磁盘临时文件辅助排序。

下列方法可以确认是否用了临时排序

/\* 打开 optimizer\_trace，只对本线程有效 \*/

SET optimizer\_trace='enabled=on'; 、

/\* @a 保存 Innodb\_rows\_read 的初始值 \*/

select VARIABLE\_VALUE into @a from performance\_schema.session\_status where variable\_name = 'Innodb\_rows\_read';

/\* 执行语句 \*/

select city, name,age from t where city='杭州' order by name limit 1000;

/\* 查看 OPTIMIZER\_TRACE 输出 \*/

SELECT \* FROM `information\_schema`.`OPTIMIZER\_TRACE`\G

/\* @b 保存 Innodb\_rows\_read 的当前值 \*/

select VARIABLE\_VALUE into @b from performance\_schema.session\_status where variable\_name = 'Innodb\_rows\_read';

/\* 计算 Innodb\_rows\_read 差值 \*/

select @b-@a;

这个方法可以通过查看OPTIMIZER\_TRACE的结果来确认，可以从number\_of\_tmp\_files中看到是否使用了临时文件

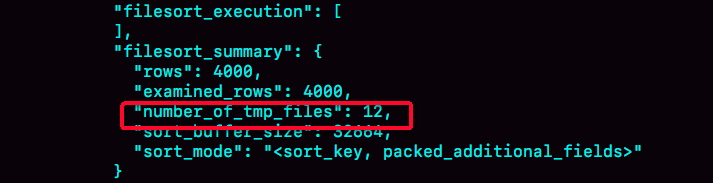


图4 全字段排序OPTIMIZER\_TRACE部分结果

Number\_of\_tmp\_files表示的是，排序过程中使用的临时文件。为什么要12个文件呢？内存放不下时，就需要使用外部排序，外部排序一般使用归并排序算法，可以简单理解，**MySQL将需要排序的数据分成12份，每一份单独排序后存在这些临时文件中。然后把这个12个有序文件在合并到一个有序的大文件。**

如果sort\_buffer\_size超过了需要排序的的数量大小，number\_of\_tmp\_files就是0，表示排序在内存中完成。

否则就需要在临时文件中排序，sort\_buffer\_size越小，需要分成的份数越多，number\_of\_temp\_files的值就越大。

接下来，解释图4中其他两个值的意思

示例表中有4000条满足city=’杭州’的记录，所以看到examined=4000,表示参与排序的行数是4000行。

Sort\_mode里面的packed\_additional\_fields的意思是，排序过程对字符串做了“紧凑”处理，即使name字段的定义是varchar(16),在排序过程中还是按照实际长度分配空间的。

同时，最后一个查询语句select @b-@a的返回结果是4000，表示整个执行扫描过程只扫描了4000行

**注意** 为了避免对结论造成干扰，把internal\_tmp\_disk\_storage\_engine设置成MyISAM。否则，select@b-@a结果会是4001

这是因为查询OPTIMIZER\_TRACE这个表时，需要用到临时表，而internal\_tmp\_disk\_storage\_engine的默认值是InnoDB。如果使用InnoDB把数据从临时表取出来的时候，会让Innodb\_rows\_read的值加1

**Rowid 排序**

上面的算法，只对原表的数据读了一遍，剩下的操作都是sort\_buffer和临时文件中执行。但这个算法有个问题，如果查询要返回的字段很多的话。那么sort\_buffer里面要放的字段太多，这样内存同时放下的行数很少，要分成多个临时文件，排序性能会很差。

如果单行很大，这个方法效果不好

**如果MySQL认为排序的单行长度太大会怎么做呢？**

让MySQL采用另外一种算法

SET max\_length\_for\_sort\_data = 16;

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这个三个字段的总长度是36，把max\_length\_for\_sort\_data设置为16，看看那就算过程有什么变化。

新的算法放入sort\_buffer的字段，只有排序的列（即name字段）和主键id

排序的结果少了city和age字段的值，不能直接返回了，整个执行流程如下

1. 初始化sort\_buffer,确定放入两个字段，name和id
2. 从索引city找到第一个满足city=’杭州’条件的主键id,也就是图中的ID\_X
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直到不满足city=’杭州’的条件为止，也就是图中的ID\_Y
6. 对sort\_buffer中的数据按照name进行排序
7. 遍历排序结果，取前100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

流程执行图如下，称为rowid 排序

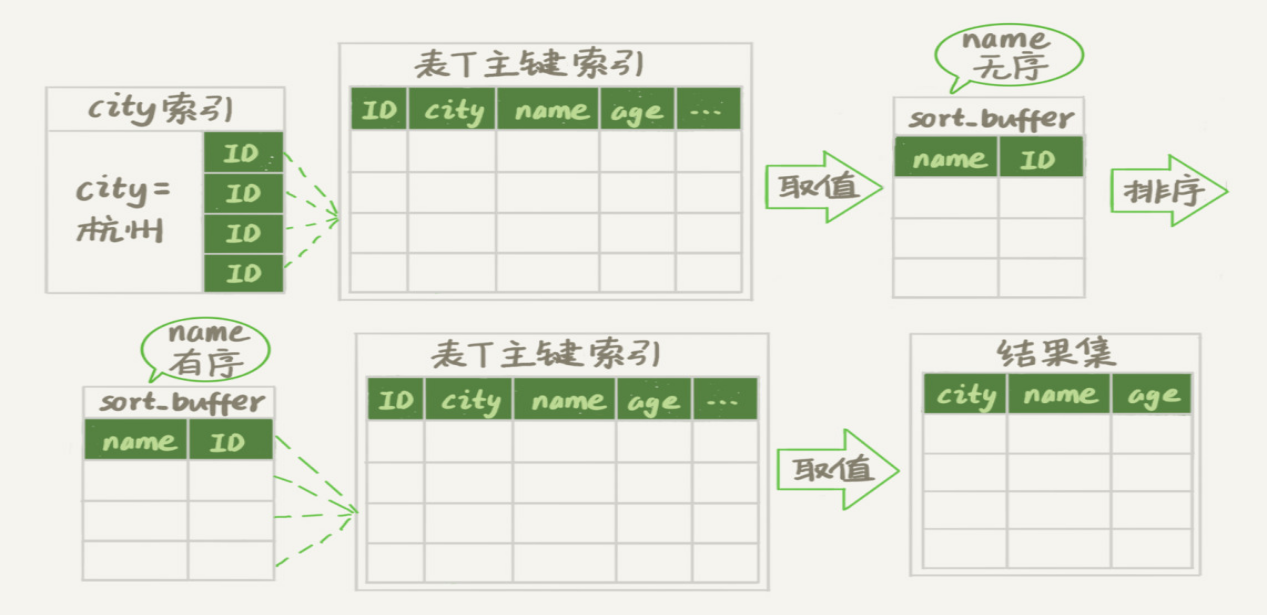


图5 rowid 排序

对比图3的，rowid多访问了一次表t的主键索引，就是步骤7

需要说明的是，最后“结果集”是一个逻辑概念，实际上MySQL服务端从排序后的sort\_buffer中依次取出id,然后回到原表查到city,name和age这三个字段的结果，不需要在服务端在消耗内存存储结果，是直接返回给客户端。

根据这个过程和图示，这个时候中select@b-@a,看看结果又什么不同

首先，图中examined\_rows的值还是4000，表示用于排序的数据是4000行，但是select@b-@a这个语句的值变成了5000

因为除了排序外，在排序完成后，还要根据id去原表取值，由于语句limit1000,因此会多读1000行。

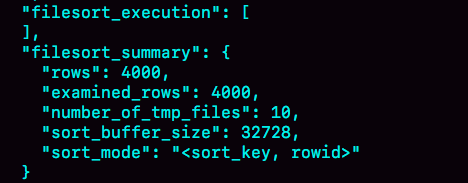


图6 rowid 排序OPTIMIZER\_TRACE部分输出

从OPTIMIZER\_TRACE的结果中，还能看到另外两个信息也变了

·sort\_mode变成<sort\_key,rowid>，表示参数与排序的只有name和id两个字段

·number\_of\_tmp\_files变成10了，因为这时候参与排序的行数仍然是4000行，但是每一行都变小了，因此需要排序的总数据量就变小了，需要的临时文件也相应的变少了。

**全字段排序 VS Rowid排序**

如果MySQL实在是担心内存太小，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，这样排序后会直接从内存里面返回查询结果，不在回到原表去取数据。

这体现了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来说，rowid排序要求回表多造成磁盘读，因此不会被优先选择。

并不是所有的Order by，都需要排序，从上面的分析的执行过程，MySQL之所以生成临时表，并在临时表上左排序操作，**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，是不是就不用再排序了呢？

我们在市民表上创建一个city和name的联合索引，对应的SQL

alter table t add index city\_user(city, name);

作为索引city的对比，看看这个索引的示意图

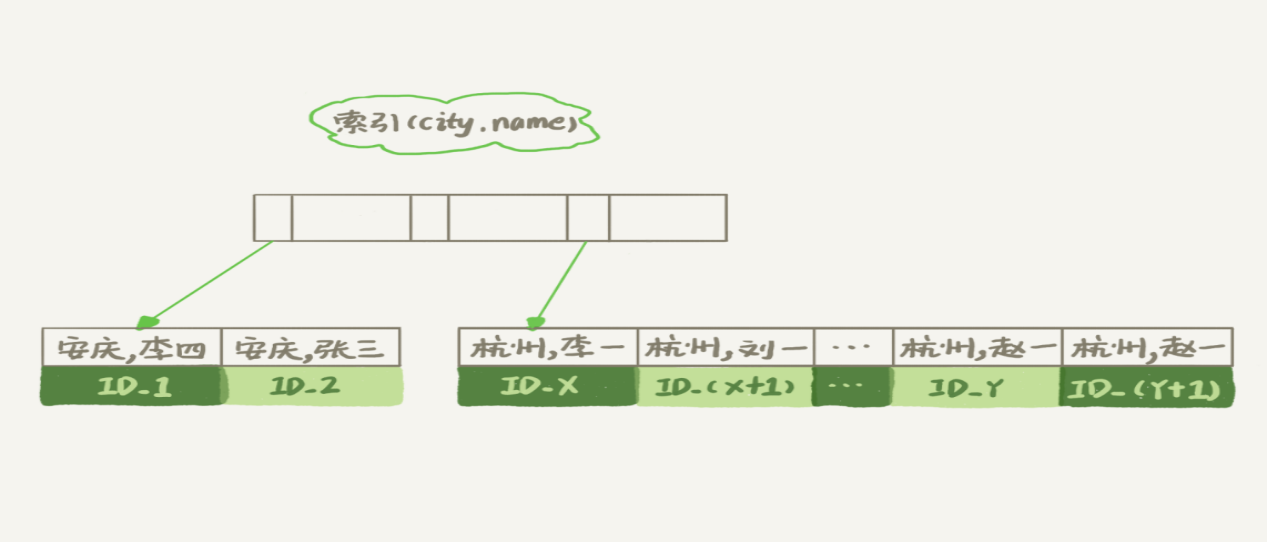


图7 city和name索引示意图

查询过程流程

1. 从索引（city,name）找到第一个city=’杭州’条件的主键
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，或者不满足city=’杭州’条件时循环结束



图8 引入（city,name）联合索引后，查询语句的执行计划

可以看到不需要临时表

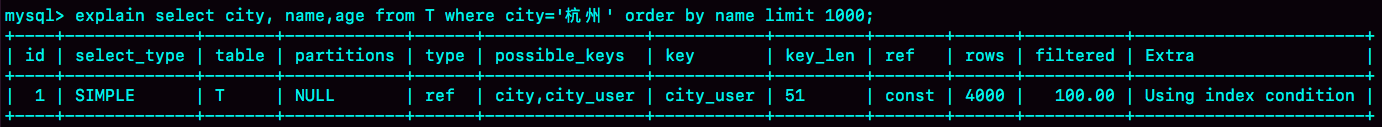


图 9 引入（city,name）联合索引后，查询语句执行计划

Extral 字段中没有 useing filesort了，也就不需要排序了，而且由于（city,name）这联合索引就是有序的，所以这个查询也不用把4000行全读一遍。只要找到满足条件的前1000条记录就可以退出。

**进一步优化**

**覆盖索引 ，索引上的信息满足查询的要求，不需要回到主键索引上去去数据**

可以创建一个city,name和age的联合索引，

alter table t add index city\_user\_age(city, name, age);

执行流程

1. 从索引（city,name,age）找到第一个满足city=’杭州’条件的记录，取出其中city,name和age这三个字段的值，作为结果集的一部分直接返回
2. 从索引（city,name,age）取下一个记录，同样取出这个三个字段的值，作为结果集的一部分直接返回
3. 重复步骤2，直到查到第1000条记录，或者不满足city=’杭州’的条件时循环结束



图10 引入（city,name,age）联合索引后，查询语句执行流程

看看explain

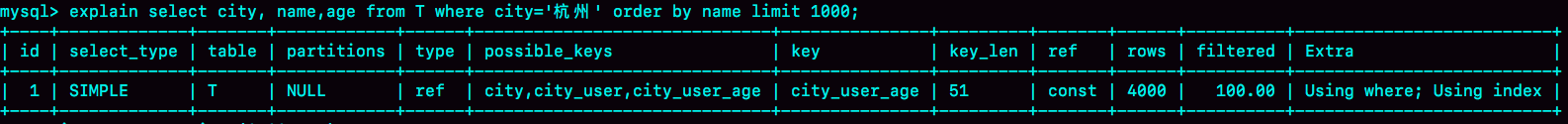


图 11 引入（city,name,age）联合索引后，查询执行计划

Extra 字段里面多了 “Using index”,表示用的是联合索引，性能上快了很多。

**思考题**

假设表里面有联合索引city\_name（city,name）这个联合索引，然后要查杭州和苏州两个城市中所有的市民的姓名，并且按名字排序，显示前100条记录，如果SQL这么写

mysql> select \* from t where city in ('杭州'," 苏州 ") order by name limit 100;

这语句执行时，会有排序过程吗，为什么？

如果从业务端需要实现一个不排序的方案，会怎么实现？

如果进一步有分页的需求，要显示第101页，也就是语句最后要改成‘limit 10000,100’

实现方法是什么？