<https://mp.weixin.qq.com/s/uuavfy3O6UiKLfCesddCUA>

https://mp.weixin.qq.com/s/jJt5jotnYGX-XlZD9d-gUw

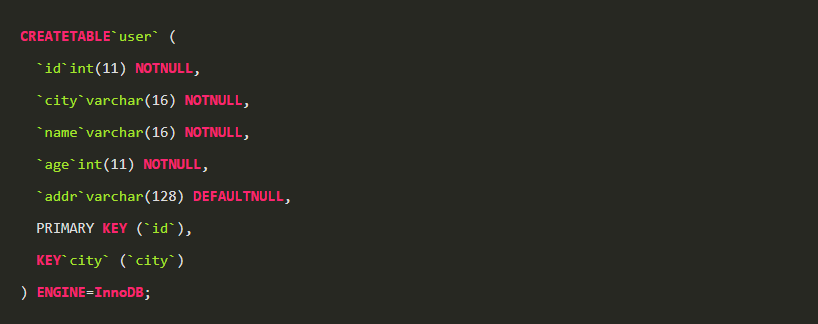
# 索引和order By 函数

# 第一部分 ORDER BY详解

order by 是我们常用的 SQL 函数之一，如果我们需要返回结果集为有序的，则我们需要使用 order by 函数。

假设我们现在有一个需求，查询用户表中来自江西的用户，并且按照名字排序返回前 20 位，面对这种需求我们就需要利用 order by 函数来帮我们实现。

用户表定义如下：



我们的 SQL 语句就可以这样写：

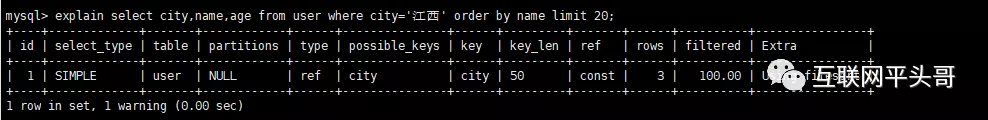


这样返回来的数据就是有序的，order by 函数背后是如何运行的？帮助我们完成结果集排序。

order by 函数要满足结果集排序要求，在 MySQL 数据库中，有两种方式来实现：**利用索引满足和使用文件排序**。

## **使用文件排序**

先使用 explain 查看上述语句的执行情况



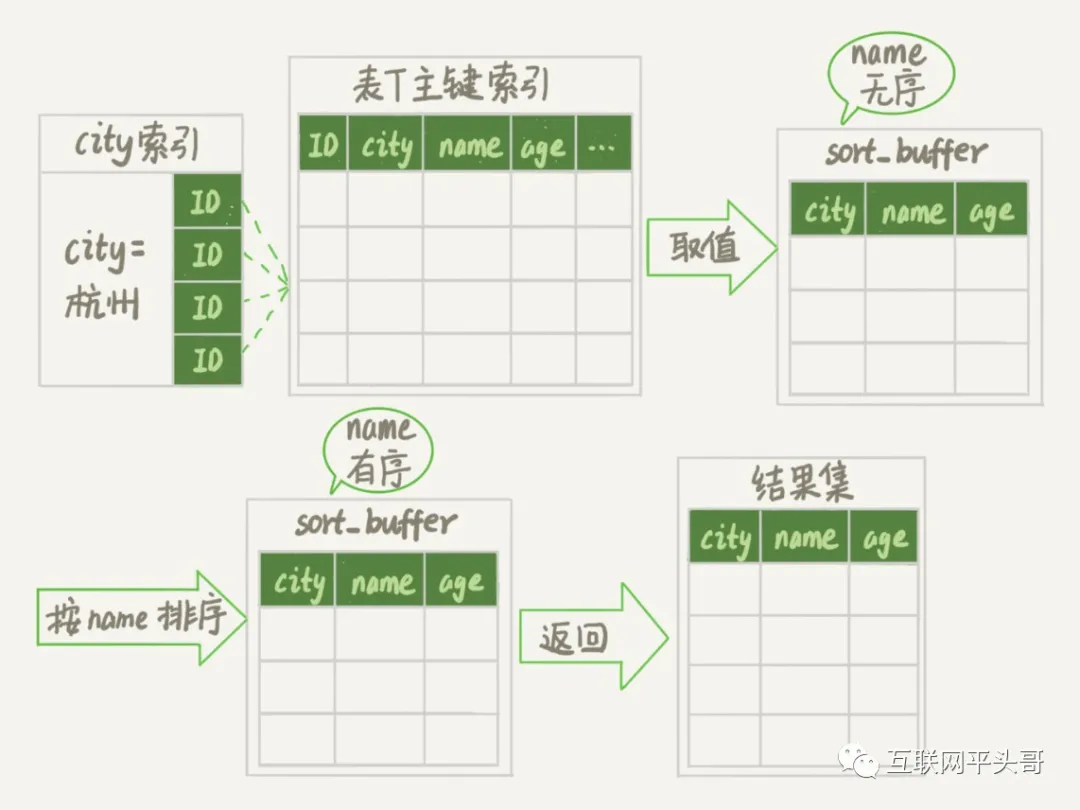
可以看到 Extra 这一行，显示为 Using filesort，说明使用了文件排序。对于文件排序 MySQL 数据库会给每个线程分配一块排序内存（sort\_buffer）。sort\_buffer 的大小可以通过 sort\_buffer\_size 来控制，默认值为 262144。

在文件排序中根据 max\_length\_for\_sort\_data 情况，MySQL 提供了两种算法，如果我们返回的字段长度总和小于 max\_length\_for\_sort\_data 的值，这会将所有的字段都放入到 sort\_buffer 中，我们称之为**全字段排序**。

如果返回的字段长度总和大于 max\_length\_for\_sort\_data 设置的值，则只会将 id，排序字段放入到 sort\_buffer 中，最后排序后再回主键索引中查询所有数据，这种称之为 **rowid 排序**。

**全字段排序和 rowid排序 的区别在于数据在 sort\_buffer 排序内存中排序后，全字段排序算法在sort\_buffer 的临时表中已经有所有的返回字段，不需要再回主键索引中查询信息，可以直接返回给用户，而 rowid排序算法在sort\_buffer 中只有部分字段，需要再回主键索引中查询数据，才能返回给用户。**

可能看了还是有点懵逼，可以参考下面这张图，加深理解：



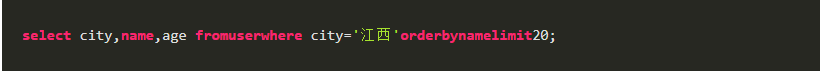
文件排序优先在 sort\_buffer 内存块中完成，如果需要排序的数据量大于  sort\_buffer\_size 的值，那么就需要**借助磁盘临时文件来完成排序**，那么性能就回有所下降，**所以在内存允许的情况下，可以适当的增大  sort\_buffer\_size 的值，来减少使用磁盘文件排序的机率**。

对于全字段排序和rowid排序，优选选择使用全字段排序，因为rowid排序需要回表查询，会造成磁盘读，不会被 MySQL 优选使用。在 MySQL8 中好像已经取消了。

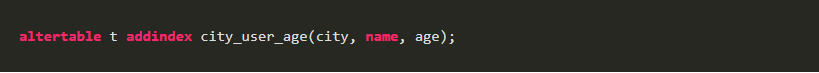
## **利用索引排序**

我们还可以利用联合索引来满足排序要求，因为索引在存储的时候就是有序的，所以在读取的时候自然就有序了。

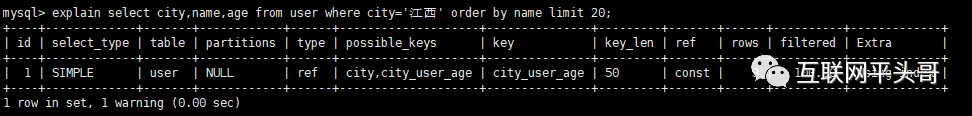
对于这条 SQL 语句：



我们可以创建一个city、name、age的联合索引，添加索引的SQL 语句为：



再次使用 explain 查看 SQL 语句执行过程：



还是看 Extra 这一行，不再是 Using filesort，而是 Using index了，说明使用联合索引。

此时这条语句的执行流程大概如下：

* 1、从索引 (city,name,age) 找到第一个满足 city='江西’条件的记录，取出其中的 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，同样取出这三个字段的值，作为结果集的一部分直接返回；
* 3、重复执行步骤 2，直到查到第 1000 条记录，或者是不满足 city='江西’条件时循环结束。



使用索引来满足排序性能要好不少，但是维护索引需要不少的代价，任何一种技术没有最好，只有最合适。

# 第二部分 联合索引

不啰嗦，直接入正题。问题是这样的。请问下面的sql语句，要想加快查询速度，该怎么创建索引？以下，以mysql数据库为准。

**select** \* **from** test **where** a=? **and** b>? **order** **by** c **limit** 0,100

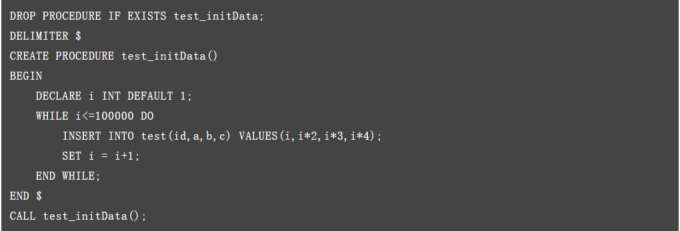
结果可能会出乎你的意料。我们首先准备一下运行环境，然后按照最左前缀原则和explain关键字来进行验证。结果真是颠覆了xjjdog多年的认知。

# **准备阶段**

为了进行验证，我们创建一个简单的数据表。里面有a、b、c三个简单的int字段。



接下来，写一个简单的存储过程，来插入10w条数据。等待大约1分钟，数据插入完毕。



由于mysql有最左前缀原则，我们对abc三列进行了全排列，创建了6个索引。这6个索引涵盖了所有的根据abc查询的情况。

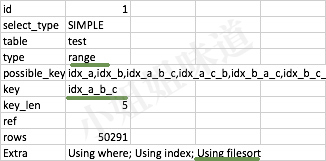


# **使用Explain进行验证**

## **1、自动选用索引**

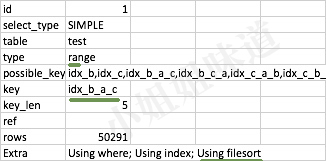
explain select \* from test where a>10 and b >10 order by c

首先，我们拿上面的sql语句进行验证。结果发现，查询使用了索引idx\_a\_b\_c，只用到了前缀a，b。而extra部分，则用到了filesort，也就是性能非常差的方式。



我们尝试换一下查询参数的位置。

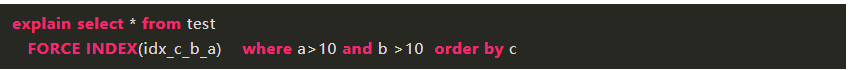
explain select \* from test where c>10 and b >10 order by a



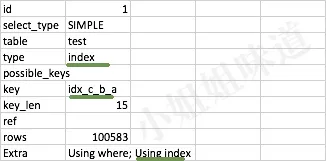
这次索引自动选择了idx\_b\_a\_c，但依然使用的filesort，查询效果是一样的。按照上面的逻辑，不是应该选择idx\_b\_c\_a么？

## **2、指定索引**

接下来使用force index方式，强制指定索引。  
这里直接给出结果，就是下面的sql。



结果如下。

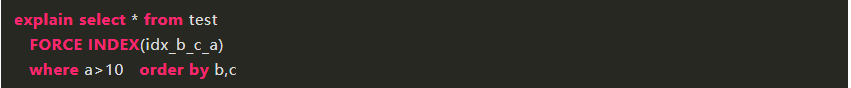


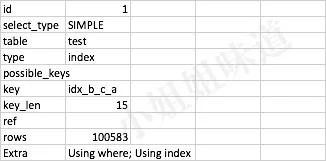
我们使用force index来指定使用的索引。这次效果非常好，显示使用了index，使用了where，只在索引上就完成了操作。但扫描的行数却增加了。

但是，这与我们的经验是相悖的。idx\_c\_b\_a的索引，是在字段(c,b,a)上创建的。按照最左原则，支持的搜索条件有：c,cb,cba。在这个例子中，order by后面的参数，却被当作了前缀的头部信息。

我们删掉其他索引，只留下idx\_c\_b\_a，然后去掉force index部分。结果发现，mysql现在能够自动的选择索引了。

再看另外一种情况，order by上有两个参数。





结果如上，使用idx\_b\_c\_a，不走filesort。其他索引都不是最优。

## **3、explain部分返回值意义**

我们得出上面的结论，是根据mysql自己提供的explain工具。这个工具能够输出一些有用的信息。下面是相关的部分返回值的意义。

**select\_type**  
表示SELECT的类型，常见的取值有：

SIMPLE    简单表，不使用表连接或子查询。  
PRIMARY    主查询，即外层的查询。  
UNION    UNION中的第二个或者后面的查询语句。  
SUBQUERY    子查询中的第一个。

**type**  
表示MySQL在表中找到所需行的方式，或者叫访问类型。常见访问类型如下，从下到上，性能越来越差。

system,const 表只有一行记录（等于系统表），这是const类型的特列。  
eq\_ref 唯一性索引扫描，对于每个索引键，表中只有一条记录与之匹配。  
ref  非唯一性索引扫描，返回匹配某个单独值的所有行，本质上也是一种索引访问，它返回所有匹配某个单独值的行，然而，它可能会找到多个符合条件的行，所以他应该属于查找和扫描的混合体。  
range 只检索给定范围的行，使用一个索引来选择行，key列显示使用了哪个索引。这种范围扫描索引比全表扫描要好，因为它只需要开始于索引的某一点，而结束于另一点，不用扫描全部索引。  
index Full Index Scan，Index与All区别为index类型只遍历索引树。这通常比ALL快，因为索引文件通常比数据文件小。  
all 全表扫描，性能最差

**Extra**  
using index  
表示相应的select操作中使用了覆盖索引，避免访问了表的数据行，效率不错。如果同时出现using where，表明索引被用来执行索引键值的查找；如果没有同时出现using where，表明索引用来读取数据而非执行查找动作。

using filesort  
说明mysql会对数据使用一个外部的索引排序，而不是按照表内的索引顺序进行读取。MySQL中无法利用索引完成的排序操作称为“文件排序”。

using temporary  
使用了用临时表保存中间结果，mysql在对查询结果排序时使用临时表。常见于排序order by和分组查询group by。

# **End**

可以看到，在我们创建了多个索引的时候，mysql的查询优化，并不一定能够进行智能的解析、用到最优的方式，需要使用force index指定索引。

mysql中的索引，主要就用在where条件中和排序动作中。分两种情况。

1、**先过滤，再排序**，会用到过滤条件中的索引参数，但是排序会使用较慢的外部排序。因为这个结果集是经过过滤的，并没有什么索引参与。

2、**先排序，再过滤**，可以使用同一个索引，排序的优先级高于过滤的优先级。选择合适的索引，在过滤的同时就把这个事给办了。但是扫描的行数会增加。

我想，mysql并不能够了解到这两个过程，到底谁快谁慢，于是选了一个最通用的方式，直接选用了第一种。甚至在索引非常多的时候，直接晕菜了。**索引建多了，你可能间接把mysql给害了。**这是现象，至于深层次的原因，欢迎读过mysql相关源码的给解释一下。

这对经常变换字段进行排序的代码来说，并不是一个好的信号。考虑到程序的稳定性，我想应该要尽量减少where条件过滤后的结果集。这种情况下，创建一个（a，b）的联合索引，或许是一个折衷的方式。