06 让order by、group by查询更快

更新时间：2019-08-06 10:03:57

> 古之立大事者，不唯有超世之才，亦必有坚韧不拔之志。
>
> ——苏轼

在工作中，我们应该经常会遇到需要对查询的结果进行排序或者分组的情况。你是否会在意这两类 SQL 的执行效率呢？这篇文稿就一起讨论下如何优化 order by 和 group by 语句。

1 order by 原理

在优化 order by 语句之前，需要先了解 MySQL 中排序的相关知识点和原理，为了方便讲解过程举例说明，首先创建一张测试表，建表及数据写入语句如下：

```
use muke;             /* 使用muke这个database */

drop table if exists t1;  /* 如果表t1存在则删除表t1 */

CREATE TABLE `t1` (       /* 创建表t1 */
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `a` int(20) DEFAULT NULL,
  `b` int(20) DEFAULT NULL,
  `c` int(20) DEFAULT NULL,
  `d` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  KEY `idx_a_b` (`a`,`b`),
  KEY `idx_c` (`c`)
) ENGINE=InnoDB CHARSET=utf8mb4 ;

drop procedure if exists insert_t1; /* 如果存在存储过程insert_t1，则删除 */
delimiter ;;
create procedure insert_t1()       /* 创建存储过程insert_t1 */
begin
  declare i int;                  /* 声明变量i */
  set i=1;                        /* 设置i的初始值为1 */
  while(i<=10000)do             /* 对满足i<=10000的值进行while循环 */
    insert into t1(a,b,c) values(i,i,i); /* 写入表t1中a、b两个字段，值都为i当前的值 */
    set i=i+1;                       /* 将i加1 */
  end while;
end;;
delimiter ;
call insert_t1();             /* 运行存储过程insert_t1 */

update t1 set a=1000 where id >9000;    /* 将id大于9000的行的a字段更新为1000 */
```

下面一起来研究下 MySQL 的排序原理：

1.1 MySQL 的排序方式

按照排序原理分，MySQL 排序方式分两种：

- 通过有序索引直接返回有序数据
- 通过 Filesort 进行的排序

> **怎么确定某条排序的 SQL 所使用的排序方式？**
>
> 使用 explain 来查看该排序 SQL 的执行计划，重点关注 Extra 字段：
>
> 如果该字段里显示是 Using index，则表示是通过有序索引直接返回有序数据。比如：

```
explain select id,c from t1 order by c;
```

```
mysql> explain select id,c from t1 order by c;
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | index | NULL          | idx_c | 5       | NULL | 10236 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

> 如果该字段里显示是 Using filesort，则表示该 SQL 是通过 Filesort 进行的排序，比如：

```
explain select id,d from t1 order by d;
```

```
mysql> explain select id,d from t1 order by d;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 10236 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
1 row in set, 1 warning (0.00 sec)
```

那么 Filesort 是否都是在磁盘中完成排序操作的呢？下面我们再看看 Filesort 排序的详情。

1.2 Filesort 是在内存中还是在磁盘中完成排序的？

MySQL 中的 Filesort 并不一定是在磁盘文件中进行排序的，也有可能在内存中排序，内存排序还是磁盘排序取决于排序的数据大小和 sort_buffer_size 配置的大小。

- 如果 “排序的数据大小” < sort_buffer_size: 内存排序
- 如果 “排序的数据大小” > sort_buffer_size: 磁盘排序

> **怎么确定使用 Filesort 排序的 SQL 是在内存还是在磁盘中进行的排序操作？**
>
> 此时就可以使用 trace 进行分析（trace 的使用方法可以复习第 1 节中 2.3 小节-trace 分析 SQL 优化器）重点关注 number_of_tmp_files，如果等于 0，则表示排序过程没使用临时文件，在内存中就能完成排序；如果大于0，则表示排序过程中使用了临时文件。
>
> 如下图，因为 number_of_tmp_files 等于 0，表示未使用临时文件进行排序，所以是内存排序。
>
> ```
> "filesort_summary": {
>   "rows": 10000,
>   "examined_rows": 10000,
>   "number_of_tmp_files": 0,
>   "sort_buffer_size": 262136,
>   "sort_mode": "<sort_key, additional_fields>"
> } /* filesort_summary */
> ```
>
> 这里解释一下上面一些参数的含义：
>
> - rows：预计扫描的行数
> - examined_rows：参与排序的行
> - number_of_tmp_files：使用临时文件的个数
> - sort_buffer_size：sort_buffer 的大小
> - sort_mode：排序模式
>
> 再看一个用到临时文件的例子，如下图，因为 number_of_tmp_files 等于 7，所以表示使用的是磁盘排序。对于 number_of_tmp_files 等于 7 表示该 SQL 将需要排序的数据分为 7 份，然后每份单独排序，再存放在 7 个临时文件中，最后把 7 个临时文件合并成一个大的有序文件。
>
> ```
> "filesort_summary": {
>   "rows": 10000,
>   "examined_rows": 10000,
>   "number_of_tmp_files": 7,
>   "sort_buffer_size": 32760,
>   "sort_mode": "<sort_key, additional_fields>"
> } /* filesort_summary */
> ```

下面再重点介绍 sort_mode。

1.3 Filesort 下的排序模式

Filesort 下的排序模式有三种，具体介绍如下：（参考《MySQL 5.7 Reference Manual》8.2.1.14 ORDER BY Optimization）

- < sort_key, rowid >双路排序（又叫回表排序模式）：是首先根据相应的条件取出相应的排序字段和可以直接定位行数据的行 ID，然后在 sort buffer 中进行排序，排序完后需要再次取回其它需要的字段；
- < sort_key, additional_fields >单路排序：是一次性取出满足条件行的所有字段，然后在sort buffer中进行排序；
- < sort_key, packed_additional_fields >打包数据排序模式：与单路排序相似，区别是将 char 和 varchar 字段存到 sort buffer 中时，更加紧缩。

因为打包数据排序模式是单路排序的一种升级模式，因此重点探讨双路排序和单路排序的区别。MySQL 通过比较系统变量 max_length_for_sort_data 的大小和需要查询的字段总大小来判断使用哪种排序模式。

- 如果 max_length_for_sort_data 比查询字段的总长度大，那么使用 < sort_key, additional_fields >排序模式；
- 如果 max_length_for_sort_data 比查询字段的总长度小，那么使用 <sort_key, rowid> 排序模式。

下面一起来通过实验验证参数 max_length_for_sort_data 对排序模式的影响：

```
set session optimizer_trace="enabled=on",end_markers_in_json=on;

SET max_length_for_sort_data = 20;

select a,d from t1 order by d;    /* 查询表t1的id、a、d三个字段的值，按照字段d进行排序  */

SELECT * FROM information_schema.OPTIMIZER_TRACE\G
```

OPTIMIZER_TRACE 结果中排序信息如下图：

```
            "filesort_summary": {
              "rows": 10000,
              "examined_rows": 10000,
              "number_of_tmp_files": 9,
              "sort_buffer_size": 32752,
              "sort_mode": "<sort_key, additional_fields>"
            } /* filesort_summary */
```

发现使用的排序模式是 < sort_key, additional_fields >

怎么让这条 SQL 的排序模式变成 <sort_key, rowid> 呢？下面我们来试验下：

因为 a、d 两个字段的总长度为 12，可以尝试把 max_length_for_sort_data 改为小于 12 的值，看排序模式是否有改变。

> 知识扩展：
>
> MySQL 常见字段类型及所占字节：
>
> | 字段类型 | 字节 |
> | --- | --- |
> | INT | 4 |
> | BIGINT | 8 |
> | DECIMAL(M,D) | M+2 |
> | DATETIME | 8 |

字段类型	字节
TIMESTAMP	4
CHAR(M)	M
VARCHAR(M)	M

```
set session optimizer_trace="enabled=on",end_markers_in_json=on;

set max_length_for_sort_data = 4;

select a,d from t1 order by d;

SELECT * FROM information_schema.OPTIMIZER_TRACE\G
```

OPTIMIZER_TRACE 结果中排序信息如下图：

```
"filesort_summary": {
  "rows": 10000,
  "examined_rows": 10000,
  "number_of_tmp_files": 6,
  "sort_buffer_size": 32760,
  "sort_mode": "<sort_key, rowid>"
} /* filesort_summary */
```

发现使用的排序模式确实变成了 <sort_key, rowid>。

可能讲到这里，你会有个疑问，为什么要添加 max_length_for_sort_data 这个参数让排序使用不同的排序模式呢？限定只用一种排序模式不行吗？

接下来，我们一起分析下 max_length_for_sort_data 的重要性。比如下面这条 SQL：

```
select a,c,d from t1 where a=1000 order by d;
```

我们先看**单路排序**的详细过程：

1. 从索引 a 找到第一个满足 a = 1000 条件的主键 id
2. 根据主键 id 取出整行，**取出 a、c、d 三个字段的值，存入 sort_buffer** 中
3. 从索引 a 找到下一个满足 a = 1000 条件的主键 id
4. 重复步骤 2、3 直到不满足 a = 1000
5. 对 sort_buffer 中的数据按照字段 d 进行排序
6. 返回结果给客户端

我们再看下**双路排序**的详细过程：

1. 从索引 a 找到第一个满足 a = 1000 的主键 id
2. 根据主键 id 取出整行，**把排序字段 d 和主键 id 这两个字段放到 sort buffer** 中
3. 从索引 a 取下一个满足 a = 1000 记录的主键 id
4. 重复 3、4 直到不满足 a = 1000
5. 对 sort_buffer 中的字段 d 和主键 id 按照字段 d 进行排序
6. 遍历排序好的 id 和字段 d，按照 id 的值回到原表中取出 a、c、d 三个字段的值返回给客户端

其实对比两个排序模式，单路排序会把所有需要查询的字段都放到 sort buffer 中，而双路排序只会把主键和需要排序的字段放到 sort buffer 中进行排序，然后再通过主键回到原表查询需要的字段。

如果 MySQL 排序内存配置的比较小并且没有条件继续增加了，可以适当把 max_length_for_sort_data 配置小点，让优化器选择使用 rowid 排序算法，可以在 sort_buffer 中一次排序更多的行，只是需要再根据主键回到原表取数据。

如果 MySQL 排序内存有条件可以配置比较大，可以适当增大 max_length_for_sort_data 的值，让优化器优先选择全字段排序，把需要的字段放到 sort_buffer 中，这样排序后就会直接从内存里返回查询结果了。

所以 MySQL 通过 max_length_for_sort_data 这个参数来控制排序，在不同场景使用不同的排序模式，从而提升排序效率。

2 order by 优化

上面我们分析了 order by 的原理，小伙伴们应该会有些优化 order by 的思路了，下面我们就一起来总结 order by 的一些优化技巧。

2.1 添加合适索引

2.1.1 排序字段添加索引

首先我们看下对 d 字段（没有索引）进行排序的执行计划：

```
explain select d,id from t1 order by d;
```

```
mysql> explain select d,id from t1 order by d;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 10236 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
1 row in set, 1 warning (0.00 sec)
```

发现使用的是 filesort（关注 Extra 字段）。

再看些对 c 字段（有索引）进行排序的执行计划：

```
explain select c,id from t1 order by c;
```

```
mysql> explain select c,id from t1 order by c;
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key   | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | index | NULL          | idx_c | 5       | NULL | 10236 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+-------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

可以看到，根据有索引的字段排序，在 Extra 中显示的就为 Using index，表示使用的是索引排序，对应本节内容 1.1。如果数据量比较大，显然通过有序索引直接返回有序数据效率更高。

因此可以在排序字段上添加索引来优化排序语句。

2.1.2 多个字段排序优化

有时面对的需求是要对多个字段进行排序，而这种情况应该怎么优化或者设计索引呢？首先看下面例子：

对 a、c 两个字段进行排序的执行计划：

```
explain select id,a,c from t1 order by a,c;
```

```
mysql> explain select id,a,c from t1 order by a,c;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 10236 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
1 row in set, 1 warning (0.00 sec)
```

观察 Extra 字段，发现使用的是 filesort。

再看对 a、b（a、b 两个字段有联合索引）两个字段进行排序：

```
explain select id,a,b from t1 order by a,b;
```

```
mysql> explain select id,a,b from t1 order by a,b;
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | index | NULL          | idx_a_b | 10      | NULL | 10236 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

发现使用的是索引排序。

多个字段排序的情况，如果要通过添加索引优化，得注意排序字段的顺序与联合索引中列的顺序要一致。

因此，**如果多个字段排序，可以在多个排序字段上添加联合索引来优化排序语句。**

2.1.3 先等值查询再排序的优化

我们更多的情况是会先根据某个字段条件查出一部分数据，然后再排序，而这类 SQL 应该如果优化呢？看下面的实验：

表 t1中，根据 a=1000 过滤数据再根据 d 字段排序的执行计划如下：

```
explain select id,a,d from t1 where a=1000 order by d;
```

```
mysql> explain select id,a,d from t1 where a=1000 order by d;
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------+----------+---------------------------------------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref   | rows | filtered | Extra                                 |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------+----------+---------------------------------------+
|  1 | SIMPLE      | t1    | NULL       | ref  | idx_a_b       | idx_a_b | 5       | const | 1001 |   100.00 | Using index condition; Using filesort |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------+----------+---------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

可以在 Extra 字段中看到 “Using filesort”，说明使用的是 filesort 排序。

再看下根据 a=1000 过滤数据在根据 b 字段排序的执行计划（a、b 两个字段有联合索引）：

```
explain select id,a,b from t1 where a=1000 order by b;
```

```
mysql> explain select id,a,b from t1 where a=1000 order by b;
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------+----------+--------------------------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref   | rows | filtered | Extra                    |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------+----------+--------------------------+
|  1 | SIMPLE      | t1    | NULL       | ref  | idx_a_b       | idx_a_b | 5       | const | 1001 |   100.00 | Using where; Using index |
+----+-------------+-------+------------+------+---------------+---------+---------+-------+------+----------+--------------------------+
1 row in set, 1 warning (0.00 sec)
```

可以在 Extra 字段中看到“Using index”，说明使用的是索引排序。

因此，对于先等值查询再排序的语句，可以通过在条件字段和排序字段添加联合索引来优化此类排序语句。

2.2 去掉不必要的返回字段

有时，我们其实并不需要查询出所有字段，但是可能因为习惯问题，就写成查所有字段的数据了。我们看下下面两条 SQL 的执行计划：

```
select * from t1 order by a,b;          /* 根据a和b字段排序查出所有字段的值 */

select id,a,b from t1 order by a,b;     /* 根据a和b字段排序查出id,a,b字段的值 */
```

```
mysql> explain select * from t1 order by a,b;
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows  | filtered | Extra          |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
|  1 | SIMPLE      | t1    | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 10138 |   100.00 | Using filesort |
+----+-------------+-------+------------+------+---------------+------+---------+------+-------+----------+----------------+
1 row in set, 1 warning (0.00 sec)

mysql> explain select id,a,b from t1 order by a,b;
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref  | rows  | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+-------------+
|  1 | SIMPLE      | t1    | NULL       | index | NULL          | idx_a_b | 10      | NULL | 10138 |   100.00 | Using index |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

根据执行计划的结果，可以看到，查询所有字段的这条 SQL 是 filesort 排序，而只查 id、a、b 三个字段的 SQL 是 index 排序，为什么查询所有字段会不走索引？

这个例子中，查询所有字段不走索引的原因是：扫描整个索引并查找到没索引的行的成本比扫描全表的成本更高，所以优化器放弃使用索引。

2.3 修改参数

在本节一开始讲 order by 原理的时候，接触到两个跟排序有关的参数：max_length_for_sort_data、sort_buffer_size。

- max_length_for_sort_data：如果觉得排序效率比较低，可以适当加大 max_length_for_sort_data 的值，让优化器优先选择全字段排序。当然不能设置过大，可能会导致 CPU 利用率过低或者磁盘 I/O 过高；
- sort_buffer_size：适当加大 sort_buffer_size 的值，尽可能让排序在内存中完成。但不能设置过大，可能导致数据库服务器 SWAP。

2.4 几种无法利用索引排序的情况

如果要写出高效率的排序 SQL，几种无法利用索引排序的情况应该熟记于心，在写 SQL 是就应该规避掉。

2.4.1 使用范围查询再排序

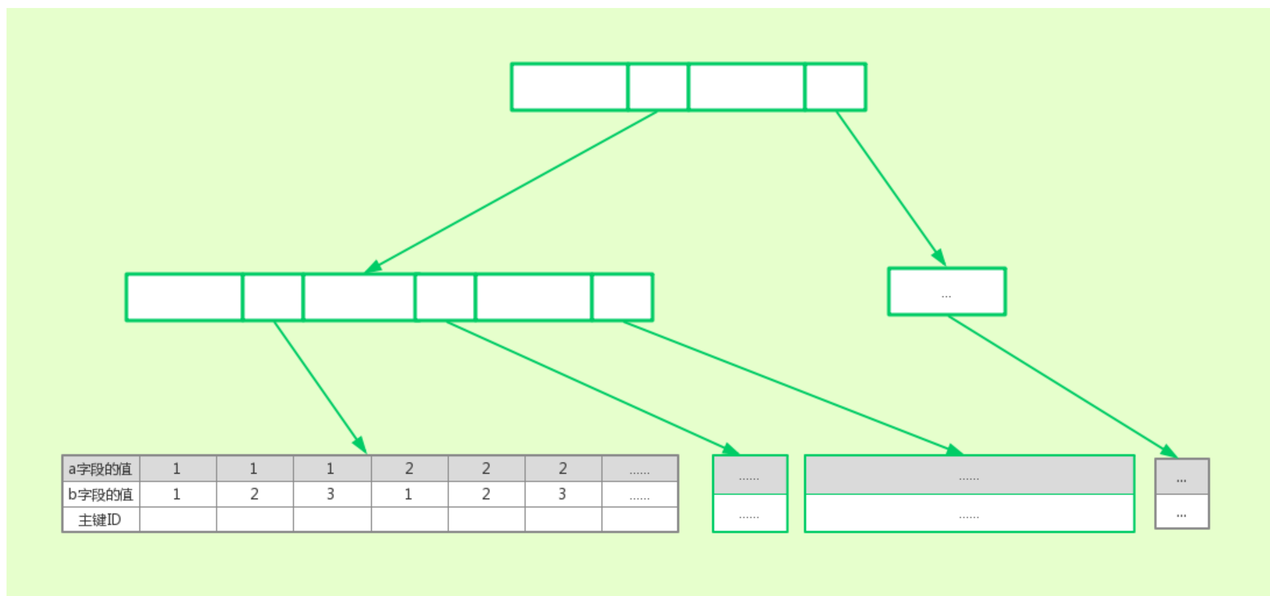
在本节 2.1.3 中介绍过，对于先等值过滤再排序的语句，可以通过在条件字段和排序字段添加联合索引来优化；但是如果联合索引中前面的字段使用了范围查询，对后面的字段排序是否能用到索引排序呢？下面我们通过实验验证一下：

```
explain select id,a,b from t1 where a>9000 order by b;
```

```
mysql> explain select id,a,b from t1 where a>9000 order by b;
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+------------------------------------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref  | rows | filtered | Extra                                    |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+------------------------------------------+
|  1 | SIMPLE      | t1    | NULL       | range | idx_a_b       | idx_a_b | 5       | NULL |    1 |   100.00 | Using where; Using index; Using filesort |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+------+----------+------------------------------------------+
1 row in set, 1 warning (0.00 sec)
```

这里对上面执行计划做下解释：首先条件 a>9000 使用了索引（关注 key 字段对应的值为 idx_a_b）；在 Extra 中，看到了“Using filesort”，表示使用了 filesort 排序，并没有使用索引排序。所以联合索引中前面的字段使用了范围查询，对后面的字段排序使用不了索引排序。

原因是：a、b 两个字段的联合索引，对于单个 a 的值，b 是有序的。而对于 a 字段的范围查询，也就是 a 字段会有多个值，取到 a，b 的值 b 就不一定有序了，因此要额外进行排序。联合索引结果如下图（为了便于理解，该图的值与上面所创建的表 t1 数据不一样）：



如上图所示，对于有 a、b 两个字段联合索引的表，如果对 a 字段范围查询，b 字段整体来看是无序的（如上图 b 的值为：1，2，3，1，2，3······）。

2.4.2 ASC 和 DESC 混合使用将无法使用索引

对联合索引多个字段同时排序时，如果一个是顺序，一个是倒序，则使用不了索引，如下例：

```
explain select id,a,b from t1 order by a asc,b desc;
```

```
mysql> explain select id,a,b from t1 order by a asc,b desc;
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+-----------------------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref  | rows  | filtered | Extra                       |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+-----------------------------+
|  1 | SIMPLE      | t1    | NULL       | index | NULL          | idx_a_b | 10      | NULL | 10236 |   100.00 | Using index; Using filesort |
+----+-------------+-------+------------+-------+---------------+---------+---------+------+-------+----------+-----------------------------+
1 row in set, 1 warning (0.00 sec)
```

3 group by 优化

默认情况，会对 group by 字段排序，因此优化方式与 order by 基本一致，如果目的只是分组而不用排序，可以指定 order by null 禁止排序。

4 总结

今天我们分享了 order by 和 group by 的一些优化技巧，下面我们一起来回顾下：

首先说到 MySQL 的两种排序方式：

- 通过有序索引直接返回有序数据
- 通过 Filesort 进行排序

建议优先考虑索引排序。

而Filesort又分为两种：

- 内存排序

- 磁盘文件排序

优先考虑内存排序。

Filesort 有三种排序模式：

- < sort_key, rowid >
- < sort_key, additional_fields >
- < sort_key, packed_additional_fields >

order by 语句的优化，这个是本节的重点：

- 通过添加合适索引
- 去掉不必要的返回字段
- 调整参数：主要是 max_length_for_sort_data 和 sort_buffer_size
- 避免几种无法利用索引排序的情况

最后说到 group by 语句的优化，如果只要分组，没有排序需求的话，可以加 order by null 禁止排序。

5 问题

对于本节生成的测试表 t1，下面这条 SQL 是否能用到索引排序：

```
select id,a,b from t1 order by b,a;
```

6 参考资料

《MySQL 5.7 Reference Manual》8.2.1.14 ORDER BY Optimization：https://dev.mysql.com/doc/refman/5.7/en/order-by-optimization.html

《深入浅出 MySQL》（第 2 版）：18.4.3 优化 ORDER BY 语句 和 18.4.4 优化 GROUP BY 语句

《高性能 MySQL》（第 3 版）：5.3.7 使用索引描述来做排序

《MySQL 技术内幕》（第 2 版）：5.6.2 联合索引

}

05 如何优化数据导入？

07 换种思路写分页查询