37 | 什么时候会使用内部临时表？

第16和34篇中，介绍了sort buffer、内存临时表和join\_buffer，这三个数据结构都是用来存放语句执行过程的中间数据，以辅助SQL语句的执行的，在排序的时候用到了sort buffer,在使用join是用到了join buffer.

什么时候MySQL会使用内部临时表呢？

**Union 执行流程**

create table t1(id int primary key, a int, b int, index(a));

delimiter ;;

create procedure idata()

begin

declare i int;

set i=1;

while(i<=1000)do

insert into t1 values(i, i, i);

set i=i+1;

end while;

end;;

delimiter ;

call idata();

然后执行下面语句

(select 1000 as f) union (select id from t1 order by id desc limit 2);

这条union语义是，取两个子集的并集，重复的行只取一行。

下图是explain 结果

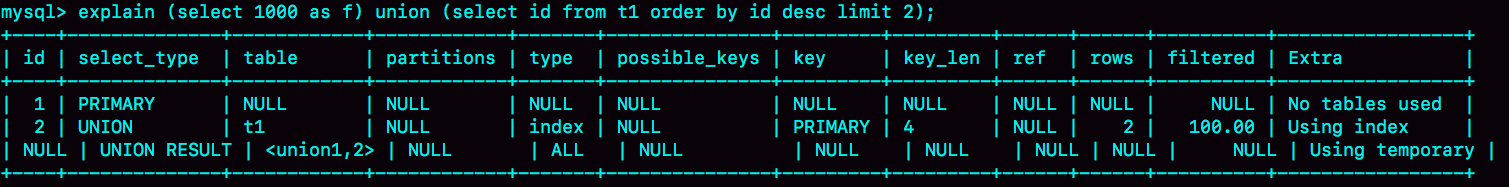


图1 union语句explain 结果

·第二行的key=PARMARY，说明第二个子句用到了索引id

·第三行的Extra字段，表示在子查询的结果集做union的时候，使用了临时表（Using temporary）

语句的执行流程是这样的

1. 创建一个内存临时表，这个临时表只有一个整型字段f,并且f是主键字段
2. 执行第一个子查询，得到1000这个值，并存入临时表中
3. 执行第二个子查询

·拿到第一行id=1000,试图插入临时表中，由于1000这个值已经存在于临时表中，违 反了唯一约束，所以插入失败，然后继续执行

·取到第二行id=999,插入临时表成功

4.从临时表按行取出数据，返回结果，并删除临时表，结果中包含数据分别是1000和999

这个过程流程图如下

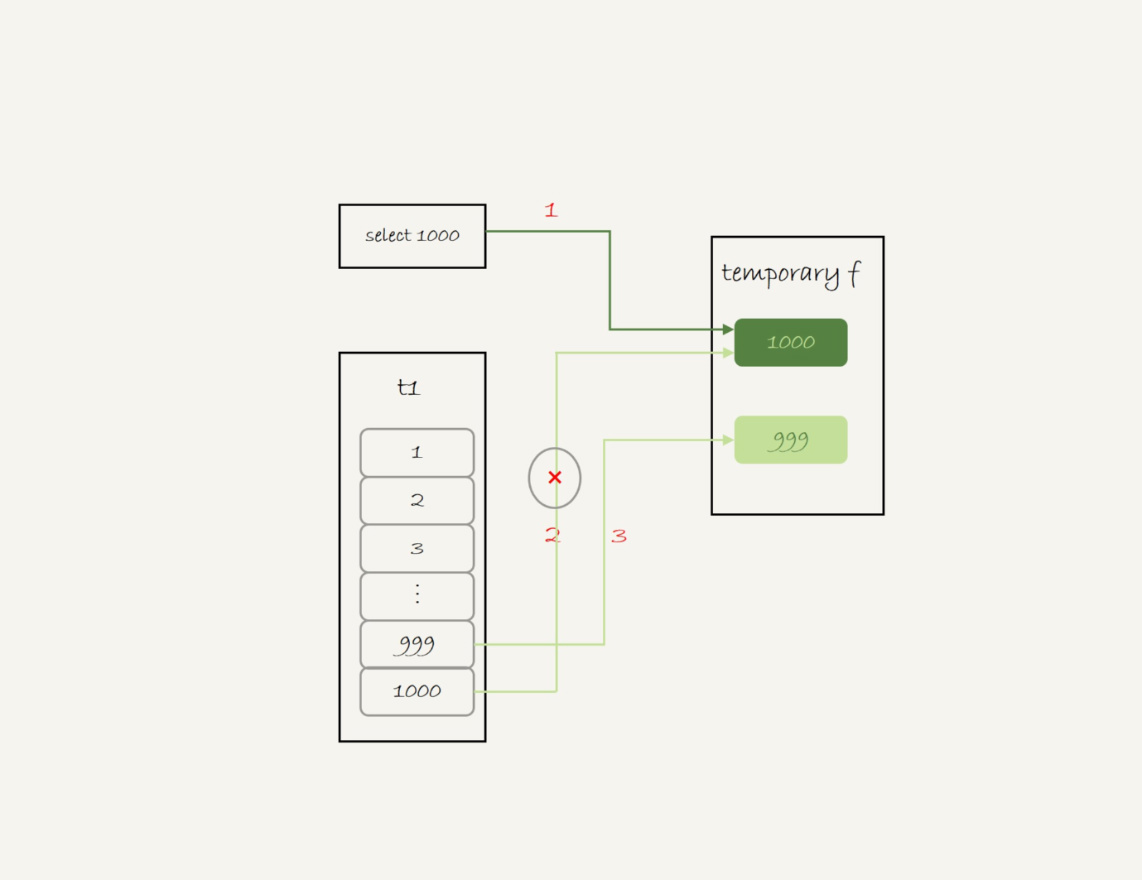


图2 union执行流程

这里内存临时表起到了暂存数据的作用，而且计算过程还用上了临时表主键id的唯一约束，实现了union的语义

如果把上面的union换成uion all的话，就没有了“去重”的语义，这样执行的时候，就依次执行子查询，得到的查询结果直接作为结果集的一部分，发给客户端，因此也就不需要临时表了

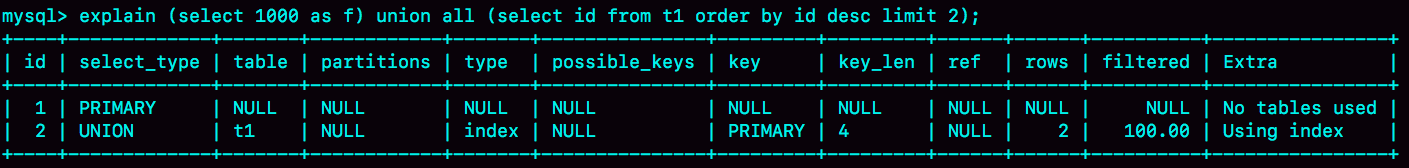


图3 union all 的explain 结果

第二行Extra字段显示的Using index,表示只使用了覆盖索引，没有使用临时表

**group by 执行流程**

另外一个常见的使用临时表的例子是group by

select id%10 as m, count(\*) as c from t1 group by m;

这个语句逻辑是把表t1里的数据，按照id%10进行分组统计，并按照m的结果排序后输出

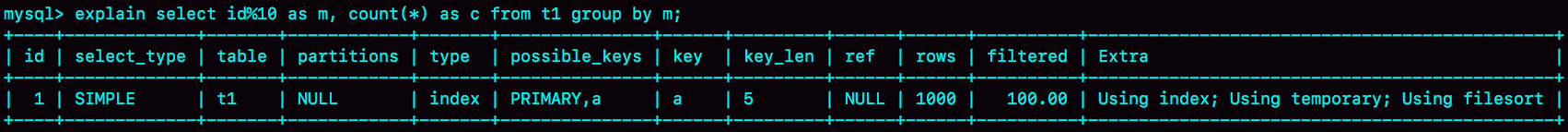


图 4 group by的explain结果

Extra字段，得到三个信息

·Using index 表示这个语句使用了覆盖索引，选择索引a,不用回表

·Using temporary 表示使用了临时表

·Using filesort 表示需要排序

语句的执行流程是这样的

1. 创建内存临时表，表里有两个字段m和c,主键是m
2. 扫描表t1的索引a，依次取出叶子节点上的id值，计算id%10的结果，即为x,

·如果临时表中没有主键为x的行，就插入一个记录（x,1）

·如果表中有主键为x的行，就将x这一行的c值加1

1. 遍历完成后，在根据字段m做排序，得到结果集返回给客户端

执行流程如下

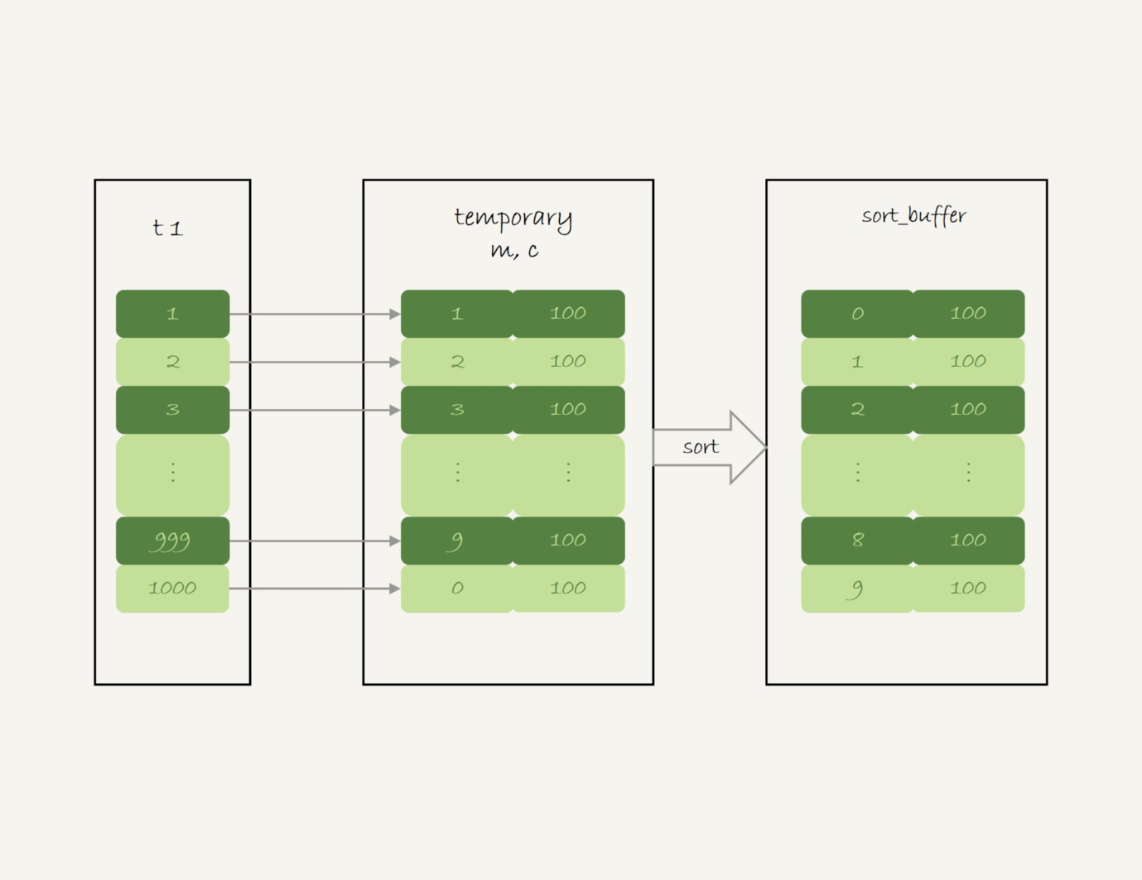


图 5 group by 的执行流程

图中最后一步，对内存临时表做了排序，在17篇介绍过，贴图，方便回顾

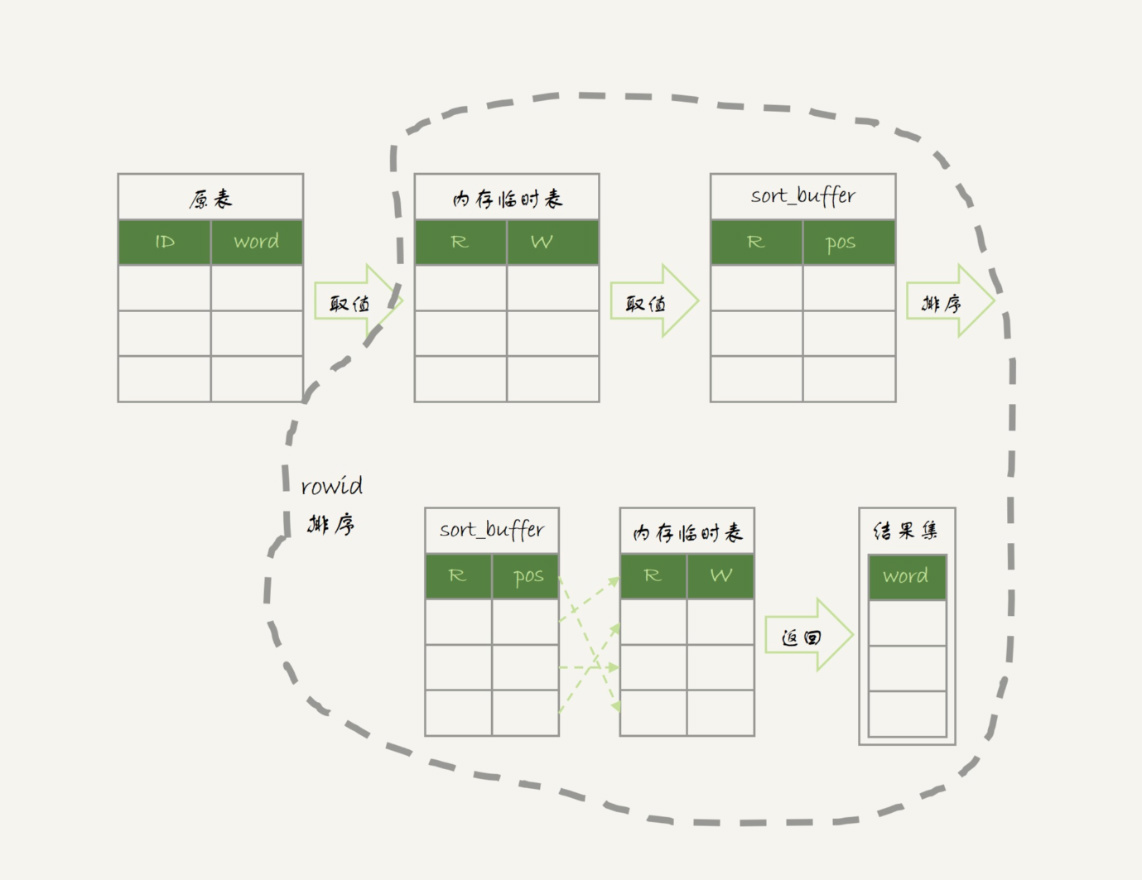


图 6 内存临时表排序流程

临时表的排序过程是图6中虚线框的过程

接下来，看这条语句的执行结果

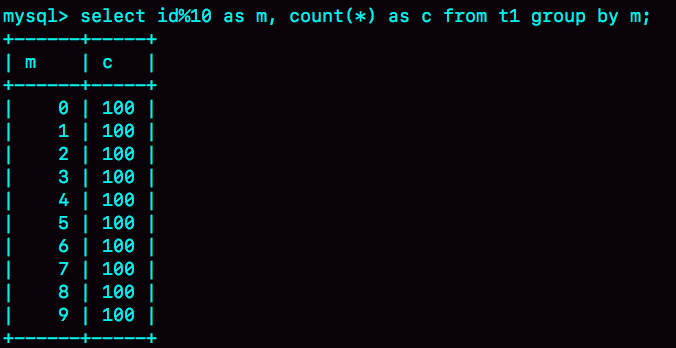


图 7 group bu 的执行结果

如果不需要对结果进行排序，可以在SQL语句末尾增加order by null 也可以改成

select id%10 as m, count(\*) as c from t1 group by m order by null;

这样直接跳过最后排序阶段，直接从临时表中取数据返回，结果如图

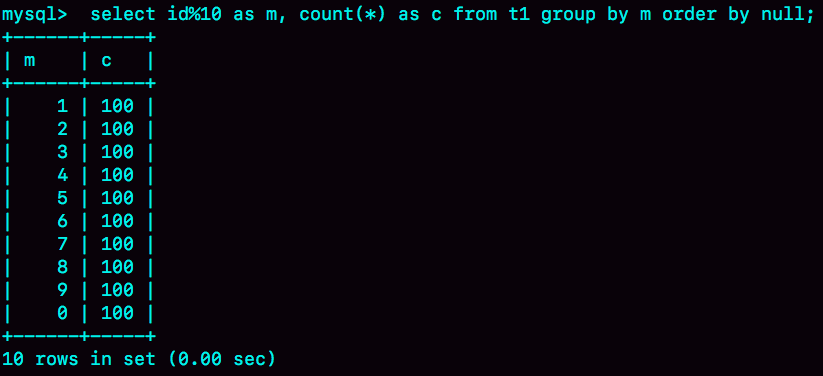


图 8 group by + order by null的结果（内存临时表）

由于表t1中的id值是从1开始，因此返回的结果集第一行是id=1,扫描到id=10的时候才插入m=0这一行，因此结果集里最后一行才是m=0.

这个例子里由于临时表只有10行，内存可以放得下，因此全过程只是用了内存临时表，但是内存临时表的大小是有限制的，参数tmp\_table\_size就是控制这个内存大小的，默认是16M

如果执行下面的语句序列

set tmp\_table\_size=1024;

select id%100 as m, count(\*) as c from t1 group by m order by null limit 10;

把内存临时表的大小限制为最大1024字节，并且把语句改成id%100,这样返回的结果里有100行数据。但是，这时的内存临时表大小不够存下100行数据，执行过程中会发现内存临时表大小达到了上限（1024字节）

此时就会把内存临时表转成磁盘临时表，磁盘临时表默认使用的引擎是innoDB。此时，返回的结果如图9

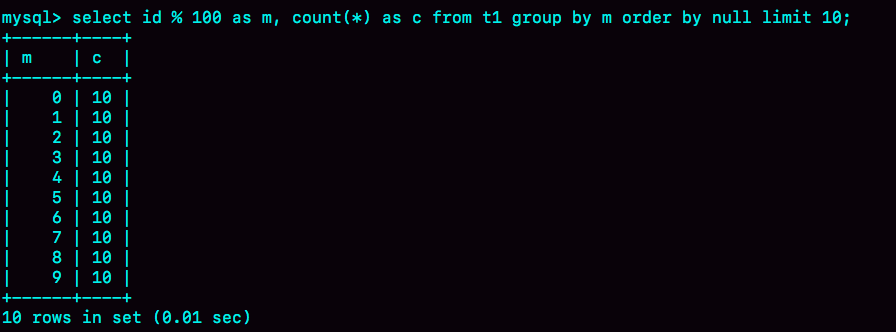


图 9 group by + order by null 的结果（临时磁盘表）

如果这个表t1的数据量很大，很可能这个查询需要的磁盘临时表就会占用大量的磁盘空间

**Group by优化方法 -- 索引**

不论使用内存临时表还是磁盘临时表，group by 逻辑都需要构造一个带唯一索引的表，执行代价都是比较高的。如果表的数据量比较大，上面这个group by语句执行起来会很慢，有什么优化方案呢？

要解决group by 语句的优化问题，可以先想一下这个问题，执行group by语句为什么需要内存临时表？

Group by 的语义逻辑，是统计不同的值出现的个数，但是，由于每一行的id%100的结果是无序的，所以就需要有一个临时表，来记录并统计结果。

那么，扫描过程中可以保证出现的数据是有序的，是不是就解决了？

假设现在有个类似图10这么一个数据结构，来看看group by 可以怎么做

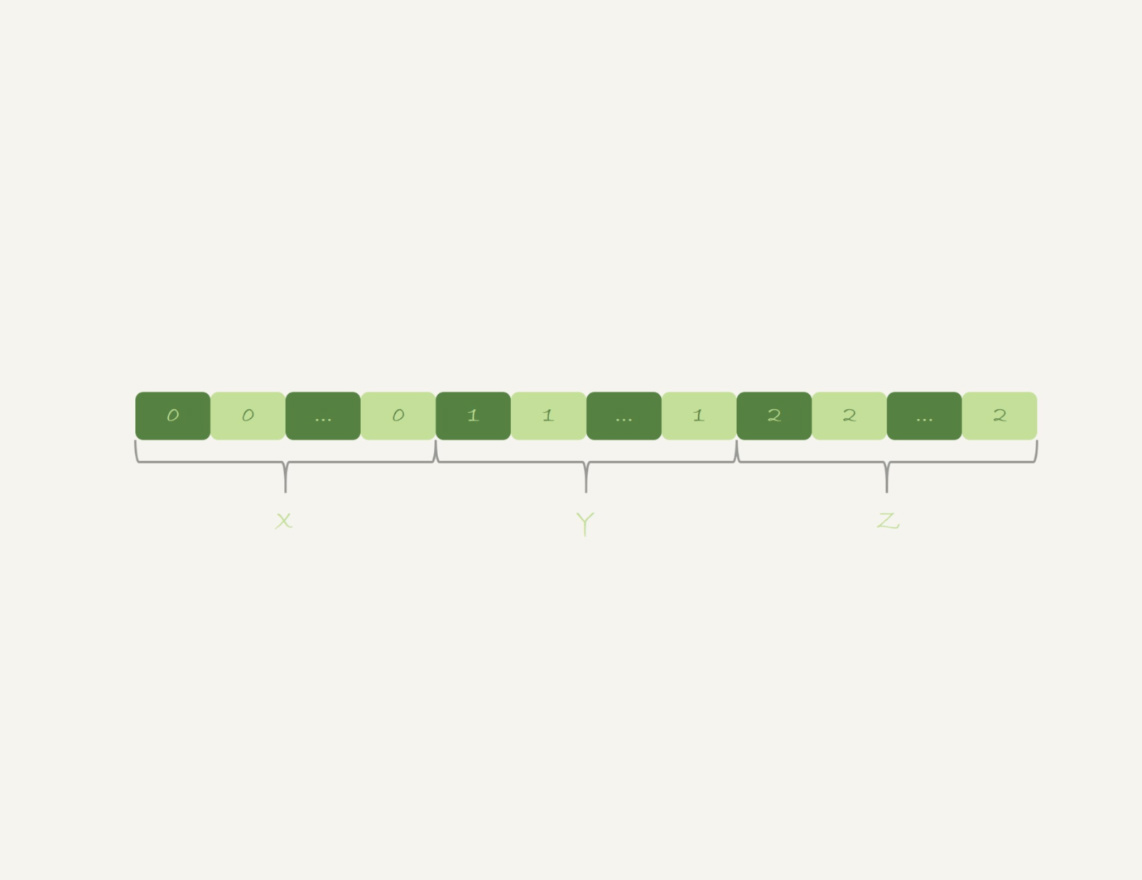


图10 group by 算法优化--有序输入

如果可以确保输入的数据是有序的，那么计算group by时，就只需从左到右，顺序扫描，一次累加，也就是下面这个过程；

·当碰到第一个1的时候，已经知道累计了X个0，结果集里的第一行就是（0，X）

·当碰到第一个2的时候，已经知道累计了Y个1，结果集里的第二行就是（1，Y）

按照这个逻辑执行的话，扫描到整个输入的数据结束，就可以拿到group by 的结果，不需要临时表，也不要再额外排序。

一定想到了innoDB索引，就是满足这个输入有序的条件

MySQL 5.7版本支持了generated column机制，用来实现列数据的关联更新，可以用下面的方法创建一个列z,然后在z列上创建一个索引（如果是MySQL 5.6及之前的版本，可以创建普通列和索引，来解决这个问题）

alter table t1 add column z int generated always as(id % 100), add index(z);

这样，索引z上的数据就是类似图10这样有序的，上面group by语句可以改成

select z, count(\*) as c from t1 group by z;

优化后的group by 语句的explain如图

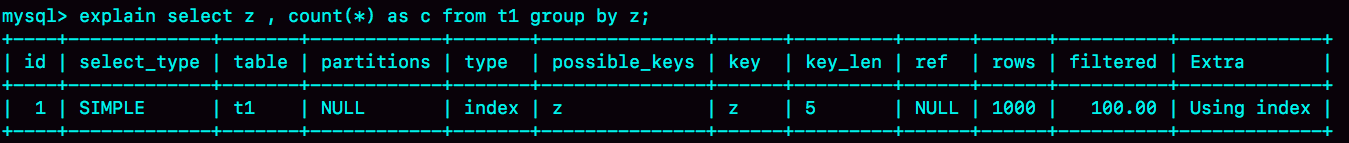


图 11 group by优化的explain结果

从Extra字段可以看到，这个语句的执行不再需要临时表，也不需要排序

**Group by 优化方法-- 直接排序**

如果可以通过加索引的来完成group by逻辑就再好不过了。但是，如果碰上不合适创建索引的场景，还是要老老实实的做排序，此时怎么优化group by？

一个group by语句中需要放到临时表的数据量特别大，却还是要按照“先放到内存临时表，插入一部分数据后，发现内存临时表不够用了在转成磁盘临时表”看上去有点傻。

有没有，MySQL直接走磁盘临时表的方法呢？--有

在group by语句中加入SQL\_BIG\_RESULT这个提示（hint）,就可以告诉优化器:这个语句涉及的数据量很大，请直接用磁盘临时表。

MySQL的优化器一看，磁盘临时表是B+树存储，存储效率不如数组来的高，所以，既然数据量很大，从磁盘空间考虑，还是直接用数组来存吧

select SQL\_BIG\_RESULT id%100 as m, count(\*) as c from t1 group by m;

执行流程是这样

1. 初始化sort\_buffer，确定放入一个整型字段，记为m
2. 扫描表t1的索引a,依次取出里面的id值，将id%100的值存入sort\_buffer中
3. 扫描完成后，对sort\_buffer的字段m做排序（如果sort\_buffer内存不够用，就会利用磁盘临时文件辅助排序）
4. 排序完成后，就得到了一个有序数组

根据有序数组，得到数组里面的不同值，以及每个值的出现次数。这一步的逻辑，从前面的图10中了解过了

下面两张图分别是执行流程图和执行explain命令得到的结果

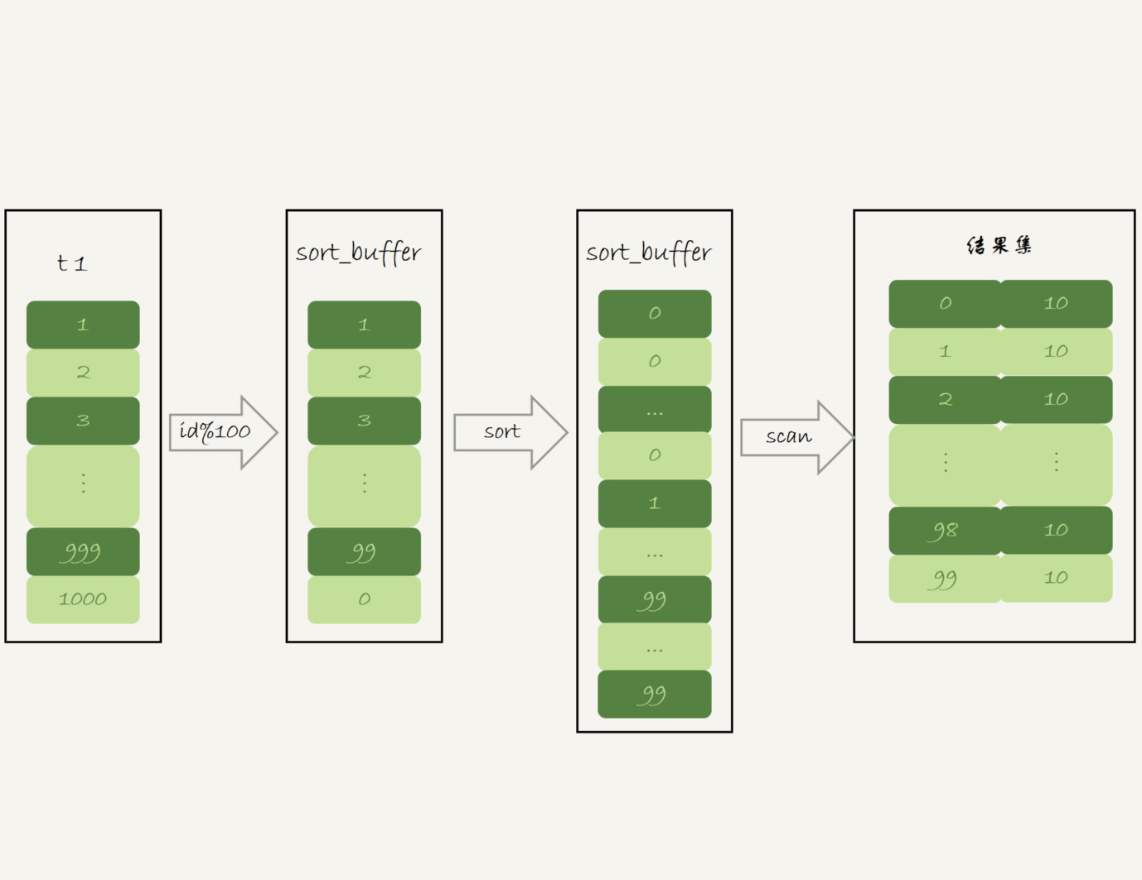


图 12 使用SQL\_BIG\_RESULT的执行流程图

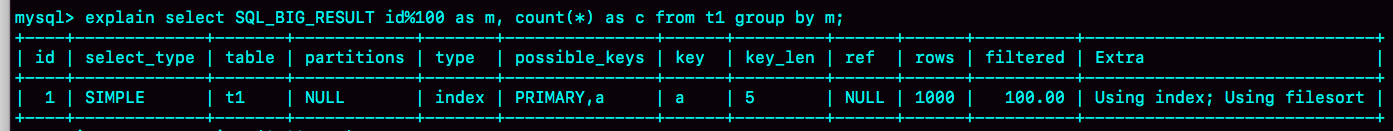


图 13 使用SQL\_BIG\_RESUTL的explain 结果

从Extra字段可以看到，这个语句没有再使用临时表，而是直接用了排序算法

基于上面的union、union all和group by语句的执行过程的分析，来回答文章开头的问题：

MySQL什么时候会使用内存临时表？

1. 如果语句执行过程中可以一边读数据，一边直接得到结果，是不需要额外内存的，否则就需要额外的内存，来保存中间结果
2. join\_buffer 是无序数组，sort\_buffer是有序数组，临时表是二维表结构
3. 如果执行逻辑需要用的二维表特性，就会优先考虑使用临时表，比如例子中 union需要用到唯一索引约束，group by 还需要用到另外一个字段来存累计数

总结

1. 如果对group by 语句的结果没有要求排序，要在语句后面加order by null
2. 尽量让group by过程用上表的索引，确认方法是explain结果里没有Using temporary和Using filesort
3. 如果group by需要统计的数据量不大，尽量只是使用内存临时表，也可以通过适当调大tmp\_table\_size参数，来避免用的磁盘临时表
4. 如果数据量实在太大，使用SQL\_BIG\_RESULT这个提示，来告诉优化器直接使用排序算法得到group by的结果

**思考题**

图8 和图9都是order by null ,为什么图8的返回结果里面，0是在结果集的最后一行，而图9的结果里面，0是在结果集的第一行？