# 原理

由于GROUP BY 实际上也同样会进行排序操作，而且与ORDER BY相比，GROUP BY 主要只是多了排序之后的分组操作。当然，如果在分组的时候还使用了其他的一些聚合函数，那么还需要一些聚合函数的计算。所以，在GROUP BY 的实现过程中，与ORDER BY一样也可以利用到索引。

在MySQL中，GROUP BY的实现同样有多种（三种）方式，其中有两种方式会利用现有的索引信息来完成GROUP BY，另外一种为完全无法使用索引的场景下使用。

## 松散(Loose)索引扫描

何谓松散索引扫描实现 GROUP BY 呢？实际上就是当MySQL完全利用索引扫描来实现 GROUP BY的时候，并不需要扫描所有满足条件的索引键即可完成操作得出结果。

下面我们通过一个示例来描述松散索引扫描实现GROUP BY，在示例之前我们需要首先调整一下group\_message表的索引，将gmt\_create字段添加到group\_id和user\_id字段的索引中。

然后再看如下 Query 的执行计划：

sky@localhost: example 09:26:15> EXPLAIN

-> SELECT user\_id,max(gmt\_create)

-> FROM group\_message

-> WHERE group\_id < 10

-> GROUP BY group\_id,user\_id\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

select\_type: SIMPLE

table: group\_message

type: range

possible\_keys: idx\_gid\_uid\_gc

key: idx\_gid\_uid\_gc

key\_len: 8

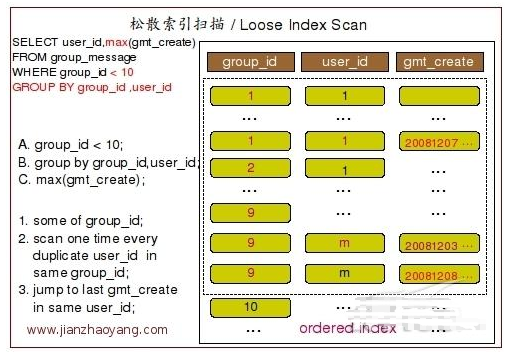
ref: NULL

rows: 4

Extra: Using where; Using index for group-by

我们看到在执行计划的 Extra 信息中有信息显示“Using index for group-by”，实际上这就是告诉我们，MySQL Query Optimizer 通过使用松散索引扫描来实现了我们所需要的GROUP BY操作。

下面这张图片描绘了扫描过程的大概实现：



要利用到松散索引扫描实现 GROUP BY，需要至少满足以下几个条件：

1、GROUP BY条件字段必须在同一个索引中最前面的连续位置;

2、在使用GROUP BY的同时，只能使用MAX和MIN这两个聚合函数;

3、如果引用到了该索引中GROUP BY条件之外的字段条件的时候，必须以常量形式存在;

**为什么松散索引扫描的效率会很高?**

因为在没有WHERE子句，也就是必须经过全索引扫描的时候，松散索引扫描需要读取的键值数量与分组的组数量一样多，也就是说比实际存在的键值数目要少很多。而在WHERE子句包含范围判断式或者等值表达式的时候，松散索引扫描查找满足范围条件的每个组的第1个关键字，并且再次读取尽可能最少数量的关键字。

## 紧凑(Tight)索引扫描

紧凑索引扫描实现GROUP BY和松散索引扫描的区别主要在于他需要在扫描索引的时候，读取所有满足条件的索引键，然后再根据读取的数据来完成GROUP BY操作得到相应结果。

sky@localhost : example 08:55:14> EXPLAIN

-> SELECT max(gmt\_create)

-> FROM group\_message

-> WHERE group\_id = 2

-> GROUP BY user\_id\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

select\_type: SIMPLE

table: group\_message

type: ref

possible\_keys: idx\_group\_message\_gid\_uid,idx\_gid\_uid\_gc

key: idx\_gid\_uid\_gc

key\_len: 4

ref: const

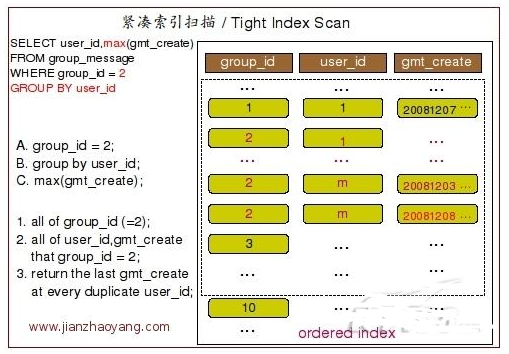
rows: 4

Extra: Using where; Using index

1 row in set (0.01 sec)

这时候的执行计划的 Extra信息中已经没有“Using index for group-by”了，但并不是说MySQL的GROUP BY操作并不是通过索引完成的，只不过是需要访问WHERE条件所限定的所有索引键信息之后才能得出结果。这就是通过紧凑索引扫描来实现GROUP BY的执行计划输出信息。

下面这张图片展示了大概的整个执行过程：



在MySQL中，MySQL Query Optimizer首先会选择尝试通过松散索引扫描来实现GROUP BY操作，当发现某些情况无法满足松散索引扫描实现GROUP BY的要求之后，才会尝试通过紧凑索引扫描来实现。

当GROUP BY条件字段并不连续或者不是索引前缀部分的时候，MySQL Query Optimizer无法使用松散索引扫描，设置无法直接通过索引完成GROUP BY操作，因为缺失的索引键信息无法得到。但是，如果Query语句中存在一个常量值来引用缺失的索引键，则可以使用紧凑索引扫描完成GROUP BY操作，因为常量填充了搜索关键字中的“差距”，可以形成完整的索引前缀。这些索引前缀可以用于索引查找。而如果需要排序GROUP BY结果，并且能够形成索引前缀的搜索关键字，MySQL还可以避免额外的排序操作，因为使用有顺序的索引的前缀进行搜索已经按顺序检索到了所有关键字。

## 临时表

MySQL在进行GROUP BY操作的时候要想利用所有，必须满足GROUP BY的字段必须同时存放于同一个索引中，且该索引是一个有序索引(如Hash索引就不能满足要求)。而且，并不只是如此，是否能够利用索引来实现GROUP BY还与使用的聚合函数也有关系。

前面两种GROUP BY的实现方式都是在有可以利用的索引的时候使用的，当MySQL Query Optimizer无法找到合适的索引可以利用的时候，就不得不先读取需要的数据，然后通过临时表来完成GROUP BY操作。

> EXPLAIN

-> SELECT max(gmt\_create)

-> FROM group\_message

-> WHERE group\_id > 1 and group\_id < 10

-> GROUP BY user\_id\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

id: 1

select\_type: SIMPLE

table: group\_message

type: range

possible\_keys: idx\_group\_message\_gid\_uid,idx\_gid\_uid\_gc

key: idx\_gid\_uid\_gc

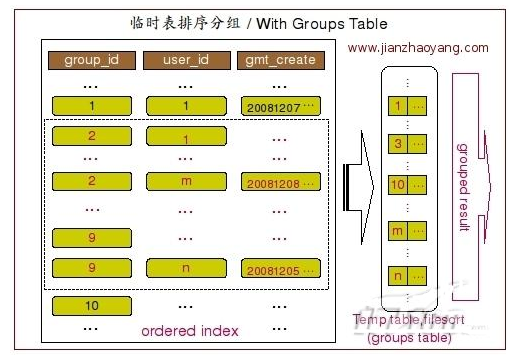
key\_len: 4

ref: NULL

rows: 32

Extra: Using where; Using index; Using temporary; Using filesort

这次的执行计划非常明显的告诉我们 MySQL通过索引找到了我们需要的数据，然后创建了临时表，又进行了排序操作，才得到我们需要的GROUP BY结果。整个执行过程大概如下图所展示：



当MySQL Query Optimizer发现仅仅通过索引扫描并不能直接得到GROUP BY的结果之后，他就不得不选择通过使用临时表然后再排序的方式来实现GROUP BY了。

在这样示例中即是这样的情况。group\_id并不是一个常量条件，而是一个范围，而且GROUP BY字段为user\_id。所以MySQL无法根据索引的顺序来帮助GROUP BY的实现，只能先通过索引范围扫描得到需要的数据，然后将数据存入临时表，然后再进行排序和分组操作来完成GROUP BY。

# 使用