# 概述

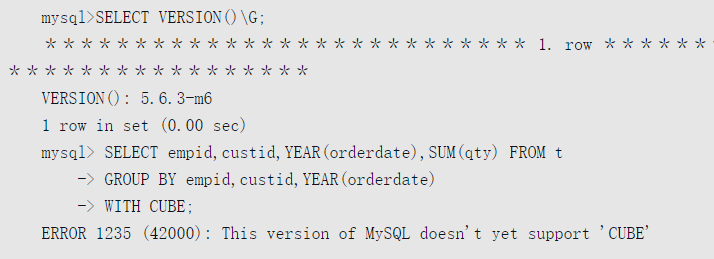
由于GROUP BY实际上也同样会进行排序操作，而且与ORDER BY相比，GROUP BY主要只是多了排序之后的分组操作（MySQL5.7）。当然，如果在分组的时候还使用了其他的一些聚合函数，那么还需要一些聚合函数的计算。所以，在GROUP BY的实现过程中，与ORDER BY一样也可以利用到索引。

## CUBE

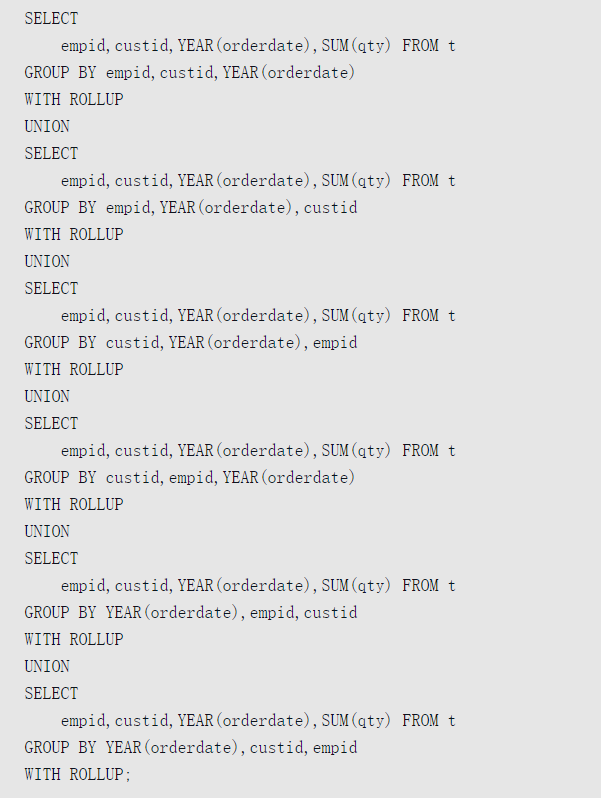
MySQL数据库支持CUBE和ROLLUP关键字，**作为GROUP BY子句的选项，应用在对多个维度进行聚合的OLAP查询中**。可以将ROLLUP看做CUBE的一种特殊情况。目前MySQL数据库仅支持CUBE关键字，而没有真正在数据库层面实现对CUBE的支持。但是可以通过ROLLUP来模拟CUBE的情况，只不过在性能上可能没有数据库原生支持更高效。

ROLLUP是CUBE的一种特殊情况。和ROLLUP操作一样，CUBE也是一种对数据的聚合操作，但是ROLLUP只在层次上对数据进行聚合，而CUBE对所有的维度进行聚合。具有N个维度的列，CUBE需要2N次分组操作，而ROLLUP只需要N次分组操作。

目前MySQL数据库定义了CUBE关键字，但不支持CUBE操作。下面是在MySQL数据库中使用CUBE后出现的错误提示：

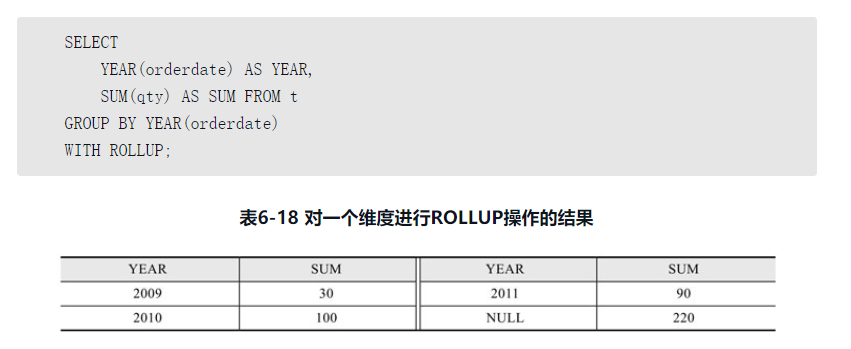


可以通过ROLLUP来模拟CUBE，例如，上述SQL语句可重写为：



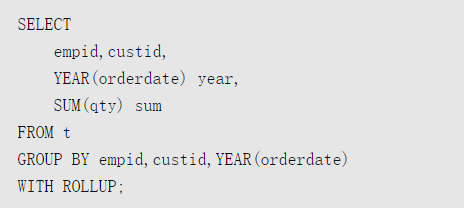
## WITH ROLLUP

ROLLUP是根据维度在数据结果集中进行的聚合操作。假设用户需要对N个维度进行聚合查询操作，普通的GROUP BY语句需要N个查询和N次GROUP BY操作。而ROLLUP的优点是一次可以取得N次GROUP BY的结果，这样可以提高查询的效率，同时大大减少了网络的传输流量。

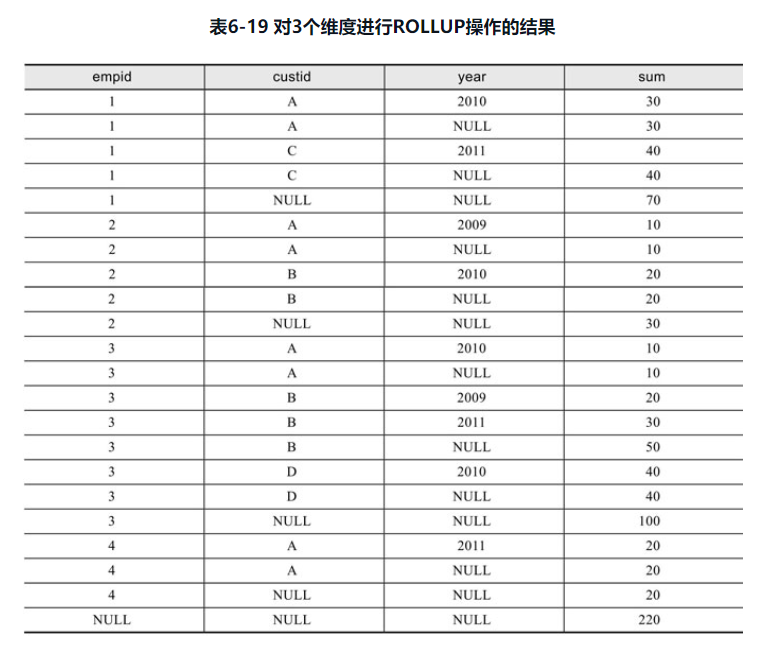


上述的ROLLUP操作只对单个列，也就是一个维度进行聚合。这时产生的结果和GROUP BY操作产生的结果没有什么太大不同，唯一不同的是最后一行产生了（NULL， 220）的结果，表示对所有YEAR的SUM再进行一次聚合，表示产生所有订单数量的总和。

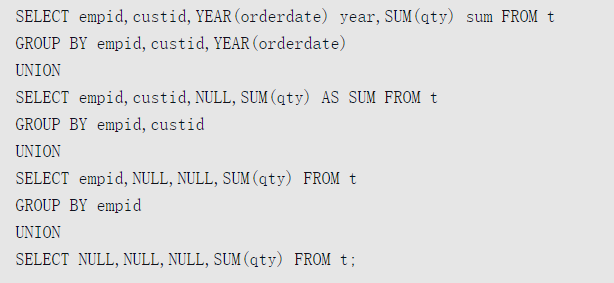
对单个维度进行ROLLUP操作只是可以在最后得到聚合的数据，对比GROUP BY语句并没有非常大的优势。对多个维度进行ROLLUP才能体现出ROLLUP的优势，例如：



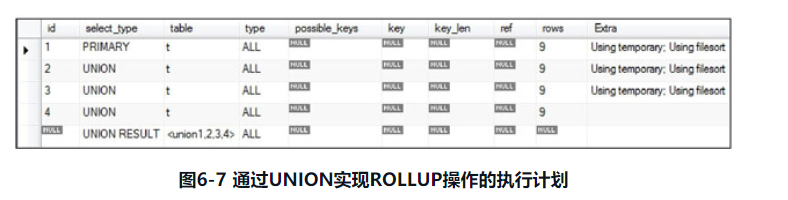
上述SQL语句根据empid、custid和YEAR（orderdate）3列进行GROUP BY操作，即需要对3列进行层次的维度操作，得到的结果如表6-19所示。



输出的结果还是比较直观和容易理解的，（NULL，NULL，NULL）表示最后的聚合， （empid，custid，year）表示对这3列进行分组的聚合结果，（empid，custid，NULL）表示对（empid、custid）两列进行分组的聚合结果，（empid，NULL，NULL）表示仅对empid列进行分组的聚合结果。因此上述ROLLUP语句等同于下面的SQL语句：



两者虽然可以得到相同的聚合结果，但是执行计划却完全不同。图6-7是通过UNION来实现ROLLUP操作的执行计划。



从图6-7中可以看出，SQL执行器需要通过4次的表扫描操作来得到结果，然后再通过UNION来进行集合的合并操作。反观ROLLUP，其执行计划如图6-8所示。



ROLLUP只需1次表扫描操作就能得到全部结果，因此SQL查询的效率在此得到了极大的提升。

在使用ROLLUP时需要注意以下几方面：

ORDER BY

LIMIT

NULL

ORDER BY语句一般用来完成排序操作，但是在含有ROLLUP的SQL语句中不能使用ORDER BY关键字，可见这两者为互斥的关键字。如果使用ORDER BY，则会出现如下的错误提示：



在含有ROLLUP的SQL语句中可以使用LIMIT，但是由于ROLLUP不能使用ORDER BY进行排序，因此LIMIT的结果阅读性差，在多数情况下无实际意义。

如果分组的列包含NULL值，那么ROLLUP的结果可能是不正确的，因为在ROLLUP中进行分组统计时值NULL具有特殊意义。因此在进行ROLLUP操作时，可以先将NULL值转换为一个不可能存在的值，或者没有特别含义的值，例如：



WITH ROLLUP可以实现在分组统计数据基础上再进行相同的统计（SUM,AVG,COUNT…）。

mysql> SELECT name, SUM(singin) as singin\_count FROM employee\_tbl GROUP BY name WITH ROLLUP;

+--------+--------------+

| name | singin\_count |

+--------+--------------+

| 小丽 | 2 |

| 小明 | 7 |

| 小王 | 7 |

| NULL | 16 |

+--------+--------------+

4 rows in set (0.00 sec)

其中记录 NULL 表示所有人的登录次数。

## MySQL8.0

MySQL5.7与MySQL8.0的区别 ：

**相同点：**  
 1、去掉重复值：根据group by后面的关键字只显示一行结果；

2、默认开启参数ONLY\_FULL\_GROUP\_BY，表示完全group by，即select后面跟的列group by后面也必须有，但是group by后面跟的列，select后面不一定需要出现；

**不同点：**

mysql5.7 group by默认还有排序功能，8.0默认只分组不排序，需要加order by才排序，这点可以从执行结果是否有Using filesort来判断。

# 原理

在优化group by查询的时候，一般的会想到两个名词：松散索引扫描（Loose Index Scan）和紧凑索引扫描（Tight Index Scan），因为通过这两种索引扫描就可以高效快速弟完成group by操作。  
 在group by操作在没有合适的索引可用的时候，通常先扫描整个表提取数据并创建一个临时表，然后按照group by指定的列进行排序。在这个临时表里面，对于每一个group的数据行来说是连续在一起的。完成排序之后，就可以发现所有的groups，并可以执行聚集函数（aggregate function）。可以看到，在没有使用索引的时候，需要创建临时表和排序。  
 MySQL建立的索引（B+Tree）通常是有序的，如果通过读取索引就完成group by操作，那么就可避免创建临时表和排序。因而**使用索引进行group by的最重要的前提条件是所有group by的参照列（分组依据的列）来自于同一个索引，且索引按照顺序存储所有的keys（即BTREE index，而HASH index没有顺序的概念）**。

MySQ有两种索引扫描方式完成group by操作，就是松散索引扫描和紧凑索引扫描。**在松散索引扫描方式下，分组操作和范围预测（如果有的话）一起执行完成的。在紧凑索引扫描方式下，先对索引执行范围扫描（range scan），再对结果元组进行分组**。

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。

参考资料：

<http://itindex.net/detail/54279-mysql-%E7%B4%A2%E5%BC%95-%E7%B4%A2%E5%BC%95>

## 松散(Loose)索引扫描

何谓松散索引扫描，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**。即索引中用于group的字段，没必要包含多列索引的全部字段。

例如：有一个索引idx(id1,id2,id3)，那么group by id1、group by id1,id2这样可以使用索引。注意，索引中用于group的字段必须符合索引的“最左前缀”原则，因此group by id1,id3是不会使用松散的索引扫描的。

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：

> 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

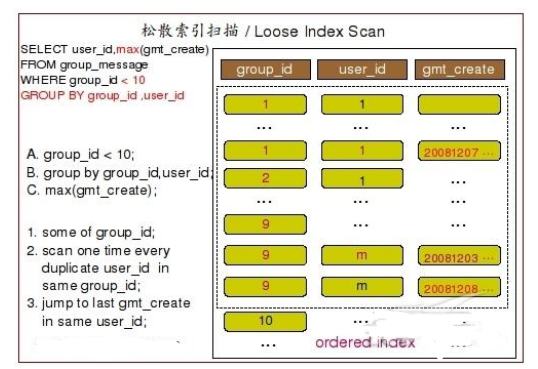
ref: NULL

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：



松散索引扫描相当于Oracle中的跳跃索引扫描（skip index scan），就是不需要连续的扫描索引中得每一个元组，扫描时仅考虑索引中得一部分。当查询中没有where条件的时候，松散索引扫描读取的索引元组的个数和groups的数量相同。如果where条件包含范围预测，松散索引扫描查找每个group中第一个满足范围条件，然后再读取最少可能数的keys。松散索引扫描只需要读取很少量的数据就可以完成group by操作，因而执行效率非常高。

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. 查询针对单表；
2. GROUP BY条件字段必须在同一个索引中**最前面**的**连续位置**，不能有其他字段;

比如表t1（c1,c2,c3,c4）上建立了索引（c1,c2,c3）。如果查询包含“group by c1,c2”，那么可以使用松散索引扫描。但是“group by c2,c3”(不是索引最左前缀)和“group by c1,c2,c4”(c4字段不在索引中)。

1. 如果在选择列表select list中存在聚集函数，只能使用 min()和max()两个聚集函数，并且指定的是同一列（如果min()和max()同时存在）。这一列必须在索引中，且紧跟着group by指定的列;

比如，select t1,t2,min(t3),max(t3) from t1  group by c1,c2。

自从5.5开始，松散索引扫描可以作用于在select list中其它形式的聚集函数，除了min()和max()之外，还支持：  
 1）AVG(DISTINCT), SUM(DISTINCT)和COUNT(DISTINCT)可以使用松散索引扫描。AVG(DISTINCT), SUM(DISTINCT)只能使用单一列作为参数。而COUNT(DISTINCT)可以使用多列参数。

2）在查询中没有group by和distinct条件。  
 3）之前声明的松散扫描限制条件同样起作用。

1. 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;

比如，select c1,c3 from t1 group by c1,c2不能使用松散索引扫描。而select c1,c3 from t1 where c3 =  3 group by c1,c2可以使用松散索引扫描。

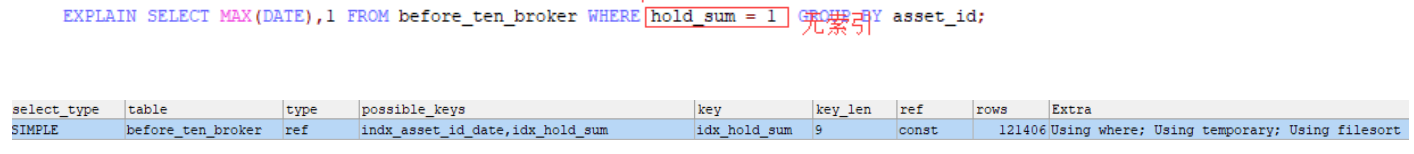
1. 如果查询中有where条件，则条件必须为索引，不能包含非索引的字段。

即索引中的列必须索引整个数据列的值(full column values must be indexed)，而不是一个前缀索引。

比如，c1 varchar(20), INDEX (c1(10)),这个索引没法用作松散索引扫描。

例如：EXPLAIN SELECT max(date),1 FROM before\_ten\_broker GROUP BY asset\_id;





**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

紧凑索引扫描可能是全索引扫描或者范围索引扫描，取决于查询条件。**当松散索引扫描条件没有满足的时候，group by仍然有可能避免创建临时表**。如果在where条件有范围扫描，那么紧凑索引扫描仅读取满足这些条件的keys（索引元组）。否则执行索引扫描。因为这种方式读取所有where条件定义的范围内的keys，或者扫描整个索引当没有where条件，因而称作紧凑索引扫描。对于紧凑索引扫描，只有在所有满足范围条件的keys被找到之后才会执行分组操作。  
 如果紧凑索引扫描起作用，那么必须满足：在查询中存在常量相等where条件字段（索引中的字段），且该字段在group by指定的字段的前面或者中间。来自于相等条件的常量能够填充搜索keys中的gaps，因而可能构成一个索引的完整前缀。索引前缀能够用于索引查找。如果要求对group by的结果进行排序，并且查找字段有可能组成一个索引前缀，MySQL同样可以避免额外的排序操作，因为对有序的索引进行的查找已经按照顺序提取所有的keys。

紧凑索引扫描实现GROUP BY和松散索引扫描的区别主要在于**它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**。

> 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

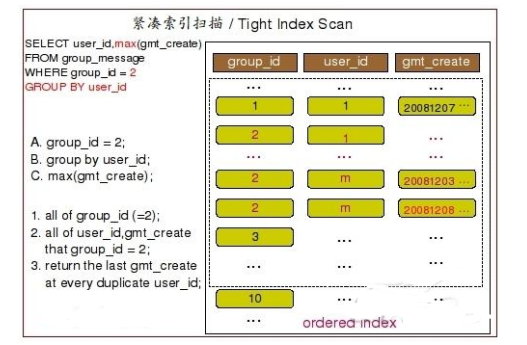
rows: 4

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

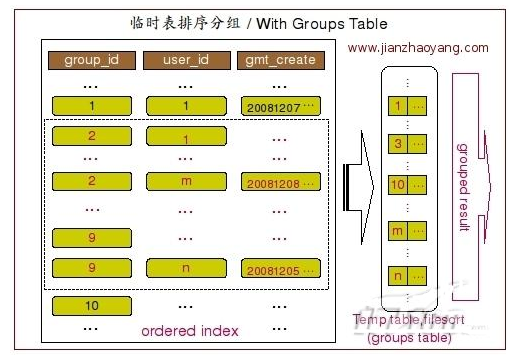
key\_len: 4

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。

# 优化

# 总结