2012年8月15日percona CEO做的PPT

**MySQL Indexing Best Practices**

PS:写完了查找不会的东西才看到之前早有人写过了。。

<http://blog.csdn.net/baiyangfu_love/article/details/7931417>

########################################

理解索引对DBA和开发人员都至关重要

索引也并非难事

理解索引；为你的程序设置最好的索引；围绕MySQL的一些限制条件工作；

索引是为了什么？

增快数据访问

约束属性(唯一和外键上)

**mysql使用索引干什么？**

1、数据查找。

2、排序。

3、避免读“数据”

4、特殊优化。

**几种常见索引类型：**

BTREE

最主要的索引类型

RTREE

MyISAM ONLY for GIS

HASH

memory、NDB

BITMAP

不被MySQL支持

FULLTEXT

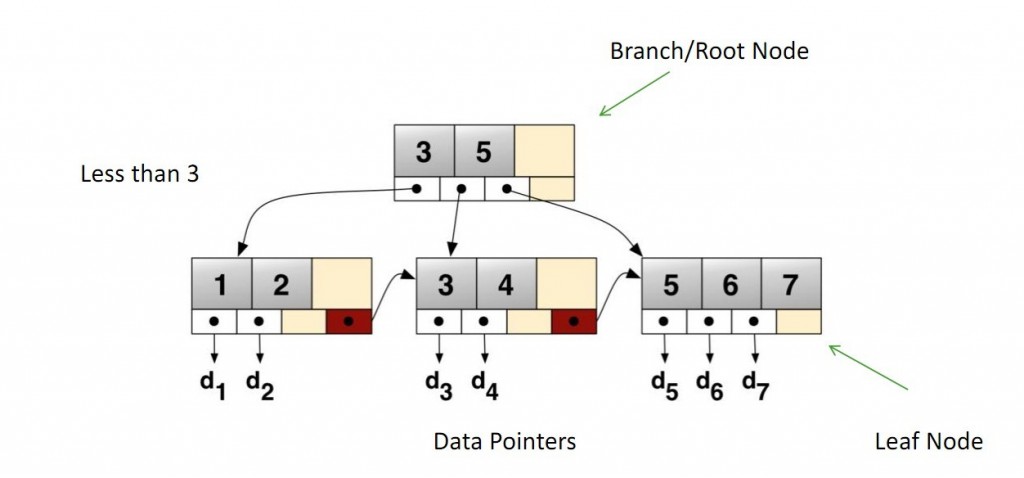
MyISAM中含有，innoDB计划在5.6包含

一些BTREE类的索引：

有很多不同的实现方式

B+ TREE主要用于磁盘存储，数据存放在叶节点

**B+TREE示意图**



**MyISAM和InnoDB的索引：**

MyISAM：**数据指针指向数据文件的物理偏移量**。所有索引实质上相同。//In MyISAM data pointors point to physical offset in the data file.

InnoDB： 主键（不论是隐含的还是明确指示的）PRIMARY KEY存贮数据在索引的leaf pages,不是指针中//PRIMARY KEY -store data in the leaf pages of the index,not pointer

次要(secondary)索引：和普通的B-Tree索引差不多了，store primary key as data pointer，**只不过在Secondary Index的所有Leaf Nodes上面同时包含了所指向数据记录的主键信息，而不是直接指向数据记录的位置信息**。所以，如果主键太大，会影响整个索引占的空间，影响查询性能。

**BTREE索引能干什么？**

查找KEY=5的列 (point lookup点查找？)

查找KEY>5的列(open range)

查找5<KEY<10的列(close range)

不能查找最后一位key是0的行//NOT find all rows with last digit of the KEY is ZERO 这不能被定义为范围操作

**字符串索引：**

LIKE关键字是一种特殊的范围查找。前缀索引例如ABC%其实也是一种特殊的范围(range),但是像%ABC后缀就不能被索引优化。

**多列索引：**

排列要求定义之后，比较第一列然后第二列,例如：(1,2,3)<(1,3,1)

这种仍是只有一个BTREE索引，而不是对每一列都索引

**索引开销Overhead of the indexing**:

索引需要额外的开销index are costly(还是英语的描述简洁明了);Do not add more than you need;in most cases extending index is better than adding new one

**Write**:更新索引占了数据库的大多数写开销

**Read**:浪费磁盘和内存空间，增加查询优化时的额外开销(additional overhead)。

**索引开销的影响：**

innoDB的长主键会使所有的Secondary keys 更长更慢

innoDB的”随机”主键(不连续的主键例如以字符串为主键)插入新值会造成大量的页拆分(page splits)

更长的索引一般更慢

index with insertion in random order --SHA1(‘password’)

low selectivity index cheap for insert --index on gender

关联索引(correlated indexes)的开销更小；如：insert\_time 与 auto\_increment \_id 关联。

**InnoDB表的索引：**

数据是以PRIMARY KEY 来划分的：

选择合适的PRIMARY KEY

For comments – (POST\_ID,COMMENT\_ID) can be good PRIMARY KEY storing all comments for single post close together

•Alternatively “pack” to single BIGINT

PRIMARY KEY 其实默认会加到所有的索引中。

              KEY(A) 其实是KEY(A,ID)

              覆盖索引(covering index)对排序非常有用。

**MySQL怎么使用索引：**

数据查找

排序

避免读”数据”

特殊优化

**1、查找数据**

SELECT \* FROM EMPLOYEES WHERE LAST\_NAME=”SMITH”

一般在LAST\_NAME上使用索引

还可以在(DEPT,LAST\_NAME)上使用多列索引SELECT \* FROM EMPLOYEES WHERE LAST\_NAME=”SMITH” AND DEPT=”accounting”

**使用多列索引有点麻烦：**

**INDEX（A,B,C）        列的顺序很重要，他会影响查询 。以下三种情况：**

                    a、**使用索引查询**

                            A>5

                            A=5 AND B>6

                            A=5 AND B=6 AND C=7

                            A=5 AND B IN (2,3) AND C>5

                     b、**不使用索引查询**

                            B>5  起始列不匹配索引

                            B=6 AND C=7  起始列不匹配索引

                      c、**使用部分索引**

                            A>5 AND B=2    在第一列使用范围查找，只是用这一列的部分。

                             A=5 AND B>6 AND C=2      在第二列使用范围查询，使用前两列索引部分。

**MySQL优化规则：mysql在使用多列索引时，遇到第一个范围查询时，将停止使用索引查询，但是，如果右面的是 IN（……）的话，将会继续使用索引列查询。**

**2、排序**

SELECT \* FROM PLAYERS ORDER BY SCORE DESC LIMIT 10

                  将会在SCORE列使用索引，如果SCORE 列上没有索引，将会使用 “filesort”(external sort)，这将是非常耗时的。

SELECT \* FROM PLAYERS WHERE COUNTRY="US"  ORDER BY SCORE  DESC LIMIT 10

经常联合使用索引用于查找，如果使用在（COUNTRY,SCORE）列上的索引，将会非常高效。

**使用多列索引：将会更加严格：  KEY (A, B)**

**a、使用索引的情况**：

                                 ORDER BY A 使用第一列排序

                                 A=5 ORDER BY B                       EQ过滤第一列，使用第二列排序

                                 ORDER BY A DESC, B DESC      以相同的方式，使用两列索引排序

                                 A>5 ORDER BY A                            范围查询在第一列，并且排序也在这一列

**b、不使用索引排序的情况：**

                                  ORDER BY B sorting by second column in the index

                                   A >5 ORDER BY B range on the first column,sorting by second

                                   A IN (1,2) ORDER BY B in-range on first column

                                   ORDER BY A DESC, B ASC sorting in the different order

                       You can’t sort in different order by 2 columns。 you can only have equality comparison (=)for columns which are not part of ORDER BY --not even IN() works in this case

                       在不同的索引列使用不同的方式进行排序将不使用索引排序。只能在非ORDER BY 列上使用 “等于” 过滤，否则，不使用索引排序。

**3、避免读数据**

               “covering index ”：覆盖索引，指你查询要求的字段，只扫描索引树就能满足要求了。applies to index use for specific query,not type of index

                只读索引，不读数据

                 SELECT STATUS FROM ORDERS WHERE CUSTOMER\_ID=123

--KEY (CUSTOMER\_ID, STATUS)

                 索引一般比数据要小。access is a lot more sequential --access through data pointers is often quite “random”

**4、特殊优化**

                  在以下情况中，索引将对 MIN() MAX()**聚合函数**的优化很有帮助：

                   SELECT MAX(ID) FROM TBL;

                   SELECT MAX(SALARY) FROM EMPLOYEE GROUP BY DEPT\_ID

                              将使用 （DEPT\_ID, SALARY） 索引

                               使用索引 进行 group by

**索引 和 Joins**

mysql一般使用 “Nested Loops” 实现 joins

SELECT \* FROM POSTS, COMMENTS WHERE AUTHOR="PETER" AND COMMENTS.POST\_ID = POSTS.ID

先在POSTS表中查找作者是peter的posts信息，对于每一个posts去COMMENTS表中找出所有的comments

索引只会在将要查找的表中使用。所以，上面例子中，POSTS.ID（索引）不会在这个查找中用到

**使用多列索引：**

MySQL可以使用多于一个的索引(index merge)

SELECT \* FROM TBL WHERE A=5 AND B=6

可以分别使用 A 列 和 B列的单独索引。但是，使用（A,B）两列的索引更好。

SELECT \* FROM TBL WHERE A=5 OR B=6

两个单独列（A）（B）的索引 都会有很好的过滤效果。但是(A, B)两列的索引不会被使用.why？

**前缀索引：**

（什么是前缀索引？）

有时需要索引很长的字符列，它会使索引变大而且变慢。一个策略就是模拟哈希索引。但是有时这也不够好，那通常可以索引开始的几个字符，而不是全部值，以节约空间并得到好的性能。这使索引需要的空间变小，但是也会降低选择性。索引选择性是不重复的索引值和表中所有行的比值。高选择性的索引有好处，因为它使mysql在查找匹配的时候可以过波掉更多的行。唯一索引的选择率为1，为最佳值。

如果索引BLOG和TEXT列，或者很长的varchar列，就必须定义前缀索引，因为mysql不允许索引它们的全文化。可以在同一个查询中针对许多不同的前缀长度进行计算，选择选择性好的。

前缀索引能很好地减少索引的大小及提高速度，但是mysql不能在order by 和group by查询中使用前缀索引，也不能把它们用伯覆盖索引。

有时后缀索引也挺有用，例如查找某个域名的所有电子邮件地址。mysql不支持反向索引，但是可以把反向字符串保存起来，并且索引它的前缀。可以用触发器维护这种索引。

你可以在列的最左边前缀创建索引。

ALTER TABLE TITLE ADD KEY(TITLE(20))

需要在BLOB/TEXT列上创建索引

可以显著地减小？can be significantly smaller

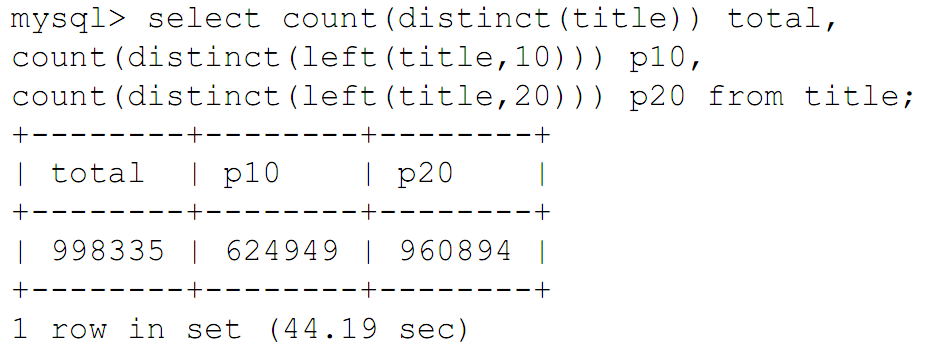
不能作为覆盖索引使用(前缀索引不能用在覆盖索引)

选取前缀的长度也是一个问题

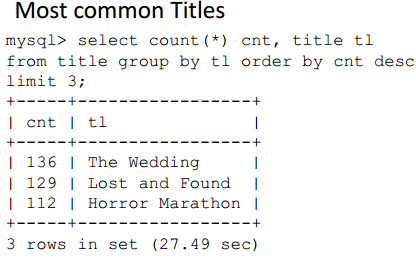
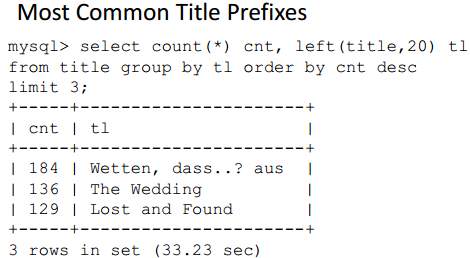
如何选择前缀索引的长度？

前缀索引应该具有很好的过滤能力。

check number of distinct prefixes vs number of total distinct values 去重前缀VS所有去重数据，效果如下：



检查异常：确保没有太多的行前缀相同：

**MySQL如何选择使用哪种索引？**

MySQL优化器会算出使用某个索引需要查询的行数。但是，他并不是简单选择查询行数最少得索引。

Estimates number of rows it needs to access for given index by doing “dive” in the table

• Uses “Cardinality” statistics if impossible

– This is what ANALYZE TABLE updates

其他的一些启发和思路：More on Picking the Index

• Not Just minimizing number of scanned rows

• Lots of other heuristics and hacks

– PRIMARY Key is special for Innodb PRIMARY KEY 对InnodB非常重要

– Covering Index benefits 覆盖索引会很有益处

– Full table scan is faster, all being equal 如果过滤效果不好，全表扫描会更快

– Can we also use index for Sorting 我们可以使用索引进行排序

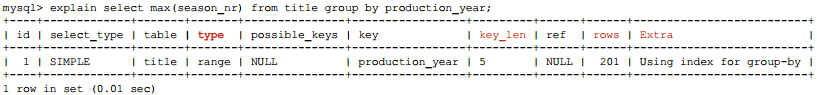
• Things to know

– Verify plan MySQL is actually using 核实mysql真正使用的执行计划

– Note it can change dynamically based on constants and data 基于常量和数据的执行计划会动态改变的

**记住：使用 EXPLAIN 查看，这个你懂的。**EXPLAIN 是一个很好的看MySQL语句执行过程的工具 真正的执行过程会有所不同

http://dev.mysql.com/doc/refman/5.5/en/usingexplain.html



**使用索引的一些例子：**

MySQL Explain 101：

Look at the “type” sorted from “good” to “bad”

system,const,eq\_ref,ref,range,index,ALL

Note “rows” – higher numbers mean slower query

Check “key\_len” – shows how many parts of the key are really used

Watch for Extra.

Using Index - Good

Using Filesort, Using Temporary - Bad

**索引策略：**

Indexing Strategy：

Build indexes for set of your performance critical queries

Look at them together not just one by one

Best if all WHERE clause and JOIN clauses are using indexes for lookups

At least most selective parts are

Generally extend index if you can, instead of creating new indexes

Validate performance impact as you’re doing changes

**索引策略例子：**

                       索引列的顺序也很重要，尽量将过滤能力强的列放到前面：

                                                    SELECT \* FROM TBL WHERE A=5 AND B=6

                                                     SELECT \* FROM TBL WHERE A>5 AND B=6

                                                     索引 （B,A）速度更快。

不要对性能影响不是太大的查询建索引。太多的索引会导致系统变慢。

**技巧 1：Trick #1 Enumerating Ranges 列举范围(也就是使用IN)**

如：索引KEY (A, B)

SELECT \* FROM TBL WHERE A BETWEEN 2 AND 4 AND B=5

只会使用索引的第一列

SELECT \* FROM TBL WHERE A IN (2,3,4) AND B=5

将会使用索引的所有列

**技巧 2：Trick #2 Adding Fake Filter 增加假过滤条件**

如：索引KEY(GENDER,CITY)

SELECT \* FROM PEOPLE WHERE CITY="NEW YORK"不

会使用索引。

SELECT \* FROM PEOPLE WHERE GENDER IN("M" , "F") AND CITY="NEW YORK"

将会使用索引

**技巧 3：Trick #3 Unionizing Filesort** 加入FILESORT

               如：索引KEY(A, B)

SELECT \* FROM TBL WHERE A IN (1， 2) ORDER BY B LIMIT 5

不用使用索引进行排序

(SELECT \* FROM TBL WHERE A=1 ORDER BY B LIMIT 5) UNION ALL (SELECT \* FROM TBL WHERE A=2 ORDER BY B LIMIT 5) ORDER BY B LIMIT 5;

将会使用索引进行排序，“filesort”  只需要对10行进行排序。