

Mysql查询性能优化-善用Explain语句

原创 2016年11月28日 14:50:50 标签：mysql / 性能优化 / explain 3221

Mysql查询性能优化-善用Explain语句

在项目中验证sql语句执行效率的时候最直观的方式就是查看其执行时间，但是在线上环境中如果不慎运行一个效率十分低下的sql导致数据库down掉了，那就悲剧了。并且只看执行时间，并无法有效的定位影响效率的原因。因此通过EXPLAIN命令查看SQL语句的执行计划，根据执行计划可以对SQL进行相应的优化。理解SQL执行计划各个字段的含义这时候显得十分重要。

下图
EXPLAIN SELECT COUNT(*) FROM blog

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|-------|------|---------------|--------|---------|--------|------|----------------|
| 1 | SIMPLE | blog | ALL | (Null) | (Null) | (Null) | (Null) | 6 | Using filesort |

这是一个简单的sql的执行计划，可以看到其包含十个字段来描述这个执行计划。其中比较重要的字段有select_type、Type、ref、Extra
下面为更好的理解执行计划，这里对每个字段进行相应的解释。

1.id

一个复杂的sql会生成多执行计划如下图：
EXPLAIN SELECT COUNT(*) FROM (SELECT id from blog where id = 1) a

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|--------|--------|---------------|---------|---------|--------|--------|---------------------------|
| 1 | PRIMARY | (Null) | (Null) | (Null) | (Null) | (Null) | (Null) | (Null) | Select tables optimized a |
| 2 | DERIVED | blog | const | PRIMARY | PRIMARY | 4 | | 1 | Using index |

图1

可以看到含有子查询的sql产生了两条记录，分别表示该条sql的执行顺序。

2.select_type

查询类型，有如下几种值

- 2.1 simple 表示简单查询，没有子查询和union 如图1所示
- 2.2 primary 最外边的select，在有子查询的情况下最外边的select查询就是这种类型如图2所示
- 2.3 union union语句的后一个语句执行的时候为该类型如图2.1所示

EXPLAIN SELECT COUNT(*) FROM blog UNION SELECT id from blog where id = 1

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|--------|--------------|-----------|-------|---------------|---------|---------|--------|--------|-------------|
| 1 | PRIMARY | blog | index | (Null) | PRIMARY | 4 | (Null) | 2 | Using index |
| 2 | UNION | blog | const | PRIMARY | PRIMARY | 4 | const | 1 | Using index |
| (Null) | UNION RESULT | <union1,2 | ALL | (Null) | (Null) | (Null) | (Null) | (Null) | |

图2.1

- 2.4 union result union语句的结果 如图2.1所示。

。 。 。 。 。 。

3.table

使用的表名

4.type

连接类型，十分重要的字段 按照代表的效果由最优到最差情况进行介绍。

- 4.1、system 表仅有一行 const的特例。
- 4.2、const 最多匹配一行并且使用primarykey 或 unique索引，才会是const。

EXPLAIN SELECT * FROM blog where id =1

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|-------|-------|---------------|---------|---------|-------|------|-------|
| 1 | SIMPLE | blog | const | PRIMARY | PRIMARY | 4 | const | 1 | |

下面这种情况搜索到一条数据但是没有用到主键或索引 所以type不是const 关于all的含义将在下文介绍

EXPLAIN SELECT * FROM blog LIMIT 1



暮光

关注

原创 6 粉丝 1 喜欢 2 评论 0

等级： 博客 访问量： 6044 积分： 123 排名： 119万+



橡树湾二手房



博主最新文章

更多文章

Java垃圾回收算法以及分代回收机制

面试题3 二维数组中的查找 java实现

剑指offer面试题 java实现版 面试题2 实现Singleton模式

JVM学习一：学习java内存区域

Struts2上传文件出错应注意

文章分类

jvm学习 2篇

剑指offer面试题 java实现 2篇

mysql 1篇

文章存档

2017年2月 1篇

2016年11月 1篇

2015年5月 2篇

2015年4月 1篇

2015年3月 1篇

博主热门文章

Mysql查询性能优化-善用Explain语句 3215

面试题3 二维数组中的查找 java实现 1156

Java垃圾回收算法以及分代回收机制 1036

Struts2上传文件出错应注意 229

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|-------|------|---------------|--------|---------|--------|------|-------|
| 1 | SIMPLE | blog | ALL | (Null) | (Null) | (Null) | (Null) | 2 | |

4.3、eq_ref

根据mysql官方手册的解释：“对于每个来自于前面的表的行组合，从该表中读取一行。这可能除了const类型最好的联接类型。它用在索引的所有部分被联接使用并且索引是UNIQUE或PRIMARY KEY”。eq_ref可以用于使用=比较带索引的列。看下面的语句

EXPLAIN SELECT * FROM blog, author where blog.blog_author_id = author.id

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|--------|--------|---------------|---------|---------|-------------|------|-------|
| 1 | SIMPLE | blog | ALL | (Null) | (Null) | (Null) | (Null) | 2 | |
| 1 | SIMPLE | author | eq_ref | PRIMARY | PRIMARY | 4 | test.blog.i | 1 | |

EXPLAIN SELECT * FROM author, blog where blog.blog_author_id = author.id

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|--------|--------|---------------|---------|---------|-------------|------|-------|
| 1 | SIMPLE | blog | ALL | (Null) | (Null) | (Null) | (Null) | 2 | |
| 1 | SIMPLE | author | eq_ref | PRIMARY | PRIMARY | 4 | test.blog.i | 1 | |

4.4、ref

对于所有取自前表的行组合，所有的匹配项都是通过索引读出的。也可以理解为连接不能基于关键字选择单个行,可能查找到多个符合条件的行。叫做 ref 是因为索引要跟某个参考值相比较。这个参考值或者是一个常数,或者是来自一个表里的多表查询的结果值。

如下图。

EXPLAIN SELECT * FROM blog where blog.blog_author_id = 2 其中blog.blog_author_id有索引

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|-------|------|---------------|-----------|---------|-------|------|-------------|
| 1 | SIMPLE | blog | ref | author_id | author_id | 5 | const | 2 | Using where |

写到这里 相信大家还是对以上各种类型的解释有点迷迷糊糊。下面看一个等值连接的例子，会加深对索引和以上解释的理解。

SELECT * FROM author, blog where author.id=blog.blog_author_id and author.id = 2

这条语句查出作者2发表的所有博客。id为author表主键，mysql会自动为主键创建唯一索引。而blog.blog_author_id是blog一个普通字段，如果对其加个索引看一下运行的效果。

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|--------|-------|---------------|-----------|---------|-------|------|-------------|
| 1 | SIMPLE | author | const | PRIMARY | PRIMARY | 4 | const | 1 | |
| 1 | SIMPLE | blog | ref | author_id | author_id | 5 | const | 2 | Using where |

先观察一下这个执行计划，可以看出mysql对sql语句的执行已经做了很好的优化。这里可以看到其中一条优化规则，先做选择操作缩小连接操作的集合维度，再做连接操作，详细可查看mysql生成执行计划的优化策略。

解释一下：第一行代表mysql生成的第一个执行计划。即select * from author where id = 2。由于id是author表的主键，且表包含多条数据但仅命中一行，所以其类型为const。

第二行：对于blog表中author_id为2的记录有多个，且是通过索引读出的。满足ref的条件。

自然而然 如果把blog表中的author_id所以去掉，则其类型应该不会再是ref。让我们来验证这个想法。

drop index author_id on blog

再来执行以下查询语句

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|--------|-------|---------------|---------|---------|--------|------|-------------|
| 1 | SIMPLE | author | const | PRIMARY | PRIMARY | 4 | const | 1 | |
| 1 | SIMPLE | blog | ALL | (Null) | (Null) | (Null) | (Null) | 4 | Using where |

可以看到type类型变为ALL了，这种类型的效率非常慢，同时你可以看到rows这一行数据也发生了变化。由于没有索引，所以需要扫描全表。详细关于ALL类型和rows列的含义将在下文中介绍。

下面接着看下一个类型。

4.5 ref_or_null

如同ref，但是添加了MySQL可以专门搜索包含NULL值的行。在解决子查询中经常使用该联接类型的优化。或解释为MySQL 必须在初次查找的结果 里找出 null 条目,然后进行二次查找。这种类型没搞明白 做实验都没出现这种类型 希望各位朋友给个例子。

但是上面说的这五种类型是属于总体来说效果很不错的了。如果能满足以上类型的查询 基本上不需要太大的优化。

下面介绍效率较低几种类型 当出现以下几种类型的查询 就要好好考虑做优化了。



联想
(Lenovo)
移动硬盘
巧轻薄：

热销

立即访问

联系我们



请扫描二维码联系客服

webmaster@csdn.net

400-660-0108

QQ客服 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

4.6 index_merge

该联接类型表示使用了索引合并优化方法。在这种情况下，key列包含了使用的索引的清单，key_len包含了使用的索引的最长的关键元素。查看下面这条sql

EXPLAIN SELECT * FROM blog where blog_title = "first" and blog_author_id = 1

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|-------|-------------|-----------------|-----------------|---------|--------|------|-------------------|
| 1 | SIMPLE | blog | index_merge | author_id,title | author_id,title | 5,23 | (Null) | 1 | Using intersect(a |

大致解释一下索引和并优化的概念，这时mysql针对sql使用多个索引进行查询时的优化方案。通俗的说就是mysql会把同一个表的多个索引扫描的结果进行合并。详细的去看看相关博客。解释一下上述的例子，分别对blog_title和authorid创建索引，这时用and查询满足以上两种条件的结果，如果查到一条的话它就是ref 但是如果匹配多条的话他就会进行索引合并。

4.7unique_subquery

顾名思义 subquery可以看出这种类型跟子查询有关系，同时大家知道子查询在mysql中是十分不建议使用的一种查询方式，当遇到子查询时多思考如果通过连接查询来优化。尽可能少的使用IN语句。

在某些 IN 查询中使用此种类型,而不是常规的 ref:value IN (SELECT primary_key FROM single_table WHERE some_expr)

EXPLAIN SELECT * FROM blog where blog_author_id in (SELECT id from author where author_name = "test1")

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|--------------------|--------|-----------------|---------------|---------|---------|--------|------|-------------|
| 1 | PRIMARY | blog | ALL | (Null) | (Null) | (Null) | (Null) | 6 | Using where |
| 2 | DEPENDENT SUBQUERY | author | unique_subquery | PRIMARY | PRIMARY | 4 | func | 1 | Using where |

即使对authorname创建索引也是相同的执行计划

对于这种情况你可以将其改写成left join语句

SELECT blog.* FROM blog LEFT JOIN author ON blog_author_id = author.id WHERE author_name = "test1"

一样的执行结果 但是执行计划就是不同的如下图

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|--------|------|---------------------|-------------|---------|----------------|------|--------------------------|
| 1 | SIMPLE | author | ref | PRIMARY,author_name | author_name | 13 | const | 1 | Using where; Using index |
| 1 | SIMPLE | blog | ref | author_id | author_id | 5 | test.author.id | 1 | Using where |

可见这种查询就是用到了索引。效率可想而知。

4.8 index_subquery

在某些 IN 查询中使用此种类型,与 unique_subquery 类似,但是查询的是非唯一性索引:

value IN (SELECT key_column FROM single_table WHERE some_expr)

EXPLAIN SELECT * FROM author where id in (SELECT blog_author_id from blog where blog_title = "secend")

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|--------------------|--------|----------------|-----------------|-------------|---------|--------|------|--------------------------|
| 1 | PRIMARY | author | index | (Null) | author_name | 13 | (Null) | 5 | Using where; Using index |
| 2 | DEPENDENT SUBQUERY | blog | index_subquery | author_id,title | author_id | 5 | func | 1 | Using where |

同样的要尽量避免使用这种方式的查询。

4.9 range

顾名思义，range意思就是范围。因此可以解释为：只检索给定范围的行,使用一个索引来选择行。key 列显示使用了哪个索引。当使用=、<>、>、>=、<、<=、IS NULL、<=>、BETWEEN 或者 IN 操作符,用常量比较关键字列时,可以使用 range。

这种类型解释的很清楚了 稍微举个栗子大家看看吧。

EXPLAIN SELECT * FROM blog where id > 2

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|-------|-------|---------------|---------|---------|--------|------|-------------|
| 1 | SIMPLE | blog | range | PRIMARY | PRIMARY | 4 | (Null) | 3 | Using where |

4.10 index

这种类型的意思也十分明显，查询过程中使用到了索引。解释为：全表扫描,只是扫描表的时候按照索引次序 进行而不是行。主要优点就是避免了排序,但是开销仍然非常大。举个栗子

EXPLAIN SELECT * FROM blog ORDER BY id

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|-------|-------|---------------|----------|---------|--------|------|-------|
| 1 | SIMPLE | blog | index | (Null) | PRIMAR 4 | (Null) | (Null) | 6 | |

4.11 all

最坏的情况,从头到尾全表扫描。性能最差的一种类型 遇到这种类型 你得想想 为什么不建索引！为什么不改造sql！改造sql也是为了让mysql运行的时候尽可能的使用到索引，这里又牵

扯出一个问题 如何建索引 数据库维护索引也是一件十分费时费力的事情。详细内容自行查询 本人还未总结~~~

这个就不举例子了 大家看看上边的例子 有很多连接查询计划中都存在all类型，顺便想想如何优化。

解释到这里大家对执行计划所代表的效率含义基本上有个认识了，现在对后面的字段进行介绍。

1.possible_keys

很明了 它的意思就是有可能使用到的索引。

6.key

MySQL 实际从 possible_key 选择使用的索引。 如果为 NULL,则没有使用索引。 很少的情况下,MYSQL 会选择优化不足的索引。 这种情况下,可以在 SELECT 语句中使用 USE INDEX (indexname)来强制使用一个索引或者用 IGNORE INDEX(indexname)来强制 MYSQL 忽略索引。

7.key_length

使用索引的长度。当然在不失精度的情况下 长度越小越好！

8.ref

显示索引的那一列被引用到了。

9.rows

MYSQL 认为必须检查的用来返回请求数据的行数，越大越不好。说明没有很好的使用到索引。

10 Extra

表示mysql解决查询的详细信息。

10.1 Using Index

表示使用到索引

10.2 using filesort

表示 MySQL 会对结果使用一个外部索引排序,而不是从表里按索引次序读到相关内容。可能在内存或者磁盘上进行排序。MySQL 中无法利用索引完成的排序操作称为“文件排序” 常见于 order by 和group by语句中。 注意如果你对排序列创建索引mysql仍然会提示你使用的是filesort，所以对于这个字段应该有自己的判断。

EXPLAIN SELECT * FROM blog order by blog_title

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|-------|------|---------------|--------|---------|--------|------|----------------|
| 1 | SIMPLE | blog | ALL | (Null) | (Null) | (Null) | (Null) | 6 | Using filesort |

10.3 Using temporary

表示进行查询时使用到临时表。当使用到临时表时，表示sql的效率需要进行相应的优化了。这种类型可能会在连接排序查询中出现。

为了便于理解先举一个例子。

EXPLAIN SELECT * FROM author,blog where author.id=blog.blog_author_id and blog.blog_title="first" order by author.id desc

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|--------|--------|---------------|---------|---------|-------------|------|--|
| 1 | SIMPLE | blog | ref | title | title | 23 | const | 2 | Using where; Using temporary; Using filesort |
| 1 | SIMPLE | author | eq_ref | PRIMARY | PRIMARY | 4 | test.blog.1 | 1 | |

这条语句是要查出写first这篇博客的博主信息，并按用户id排序。

先来看看mysql连接查询算法 Nested Loop Join 通过驱动表的结果集，一条一条的按照连接条件查询下个表中的记录。

这里出现了一个名词 驱动表

驱动表定义：

- 1.当连接条件确定时，查询条件筛选后记录少的为驱动表。
- 2.当连接条件不确定时，行数少的表为驱动表。

按照上述定义，由于blog_title经过筛选条件后查询得到的记录数为2，而未对author表进行条件过滤，因此该sql的驱动表为blog。

将过滤后的blog表的记录一条条的对author表查询，而后合并，这时需要按照author表的id字段进行排序，因此需要对合并结果(临时表)进行排序。

如果按照驱动表排序，则可以直接排序而无需临时表。

EXPLAIN SELECT * FROM author,blog where author.id=blog.blog_author_id and blog.blog_title="first" order by blog.id desc

| id | select_type | table | type | possible_keys | key | key_len | ref | rows | Extra |
|----|-------------|--------|--------|---------------|---------|---------|-------------|------|-------------|
| 1 | SIMPLE | blog | ref | title | title | 23 | const | 2 | Using where |
| 1 | SIMPLE | author | eq_ref | PRIMARY | PRIMARY | 4 | test.blog.1 | 1 | |

看 Python 如何诠释 “薪” 时代

Python全栈开发包含Python爬虫、前端、网站后台、Python机器学习与数据挖掘等,从0基础小白到Python 企业级web开发达人、自动化运维开发能手的进击,课程真实企业项目实战演练,全面系统学习python编程语言,从容应对企业中各式各样的.....

查看更多>>

29224

查看更多>>



3



写下你的评论...

Mysql常用30种SQL查询语句优化方法



youthsunshine 2016年12月05日 15:32 6568

1、应尽量避免在 where 子句中使用!=或 2、对查询进行优化,应尽量避免全表扫描,首先应考虑在 where 及 order by 涉及的列上建立索引。 3、应尽量避免在 where 子句中对...

mysql语句优化总结（一）



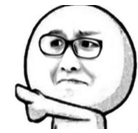
kevinlifeng 2015年01月28日 17:10 13521

Sql语句优化和索引 1.Innerjoin和左连接,右连接,子查询 A. inner join内连接也叫等值连接是, left/rightjoin是外连接。 SEL ECT A.id,A....

技术外文文献看不懂？教你一个公式秒懂英语

不背单词和语法，一个公式学好英语

广告



Mysql优化（一）—Sql语句



sunliduan 2015年12月21日 12:49 787

第一篇是最基础，同样也是最重要的Sql语句的优化。就和炒菜一样，假如我们的原材料，例如青菜或者肉是坏了的，即便我们拥有手艺一流的厨子，品质保证的厨具，也不可能做出美味的佳肴。所以，不仅要有好厨师好厨具...

Mysql sql语句优化的10条建议



u011650048 2016年06月13日 09:40 710

1、将经常要用到的字段（比如经常要用这些字段来排序，或者用来做搜索），则最好将这些字段设为索引 2、字段的种类尽可能用int或者tinyint类型。另外字段尽可能用not null 3、当...

mysql ori语句的优化



sunyuhua_keyboard 2017年10月26日 13:49 441

在某些情况下，or条件可以避免全表扫描的。1 .where 语句里面如果带有or条件, myisam表能用到索引， innodb不行。 1)myis am表： CREATE TABLE IF NO...

考勤管理系统

考勤管理系统哪个牌子比较好

百度广告



【MySQL】基于MySQL的SQL优化（一）——从用explain关键字分析SQL语句开始

explain显示了MySQL如何使用索引来处理select语句以及连接表。可以帮助选择更好的索引和写出更优化的查询语句。 如图：！ [这里写图片描述](http://img.blog.csdn....

tanglei6636 2016年10月17日 21:23 1526

MySQL优化-----从最简单的语句优化开始



u012116133 2016年09月20日 11:00 181

今天学习了一下MySQL的优化的内容，做下笔记先。MySQL可以从多个方面进行优化，SQL语句的优化，数据库结构优化，索引优化，存储引擎优化等。今天就说一下SQL语句的优化，我们经常会用到以下语法的...

经典MySQL语句大全



lvhao__sir 2017年03月08日 00:00 852

一、基础 1、说明：创建数据库 CREATE DATABASE database-name 2、说明：删除数据库 drop database dbname 3、说明：备份sql serve...

[MySQL学习] 常用SQL语句大全总结

 Hanrovey 2017年03月31日 10:02 11097

转载地址：http://www.cnblogs.com/0351jiazhuang/p/4530366.htm

ISQL是(Structured Query Language)结构化查询语言的简称，下面...

MySQL性能优化方案总结

 sinat_23080035 2016年10月13日 00:26 8242

 SQL进行优化， 效果: SQL和索引 > 数据库表结构 > 系统配置 > 硬件 ；但成本从低到高。...

3

.7万欧起全家三代定居希腊！

 全家三代享受免费教育与公立医疗，最快60天起即可定居希腊！



MySQL系列—如何优化你的 SQL SELECT 语句性能

SELECT语句的性能调优有时是一个非常耗时的任务，在我看来它遵循帕累托原则。20%的努力很可能会给你带来80%的性能提升，而为了获得另外20%的性能提升你可能需要花费80%的时间。除非你在金星工作， ...

 u012758088 2016年07月31日 18:03 1237

浅谈MySQL中优化sql语句查询常用的30种方法 yin767833376 2016年07月06日 14:19 624

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。 2.应尽量避免在 where 子句中使用!=或 3.应尽量避免在 where...

mysql 的sql优化

 ECHO_FOLLOW_HEART 2017年02月23日 15:06 295

show [session|global] status命令可以提供服务器状态信息，可以知道读写比率；回滚操作的情况，太频繁说明程序有问题 Slow _queries：慢查询的次数 定位执行效率较低的...

Mysql查询性能优化-善用Explain语句

 zbw18297786698 2017年01月11日 22:08 907

在项目中验证sql语句执行效率的时候最直观的方式就是查看其执行时间，但是在线上环境中如果不慎运行一个效率十分低下的sql导致数据库down掉了，那就悲剧了。并且只看执行时间，并无法有效的定位影响效率的...

MySQL多表查询核心优化

 u013761665 2016年03月22日 17:00 6550

在一般的项目开发中，对数据表的多表查询是必不可少的。而对于存在大量数据量的情况时（例如百万级数据量），我们就需要从数据库的各个方面来进行优化，本文就先从多表查询开始。 ...


免费云主机试用一年

云服务器免费试用

百度广告



MySQL多表查询核心优化

 xiaominggunchuqu 2016年03月23日 14:48 339

文章出处：http://blog.csdn.net/lemon_tree12138/article/details/50921193 概述 在一般的项目开发中，对数据表的...

mysql千万级大数据SQL查询优化

 u014421556 2016年07月29日 13:27 43349

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。 2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎...

MySQL性能优化的最佳21条经验

 waferleo 2012年01月05日 17:32 42688

今天，数据库的操作越来越成为整个应用的性能瓶颈了，这点对于Web应用尤其明显。关于数据库的性能，这并不只是DBA才需要担心的事，而这更是我们程序员需要去关注的事情。当我们去设计数据库表结构，对操作数据...

MySQL 使用explain分析sql语句的查询效率（一） JathamJ 2016年12月28日 16:24 6092

mysql explain用于分析sql 语句的执行及数据库索引的使用。本文将致力于帮助大家充分理解explain所返回的各项参数，从而使大家快速掌握explain用法技巧。如果你在看其他教程或视频后...

mysql中explain用法和结果的含义



u010061060

2016年09月08日 17:03



3366

explain select * from user explain extended select * from user id

SELECT识别符...



3

