# MySQL性能优化

**By Brother.Lee 20180116 Shanghai**

# SQL脚本规范

/\*-------------------------------------------------------------------------------------------------------------  
| SVN路径：http://dev.tq365.cn/svn/sql/项目名称  
| table\_开发人员姓名\_日期\_序号.sql：创建或修改表的sql脚本  
| data\_开发人员姓名\_日期\_序号.sql：插入或修改数据的sql脚本  
| func\_开发人员姓名\_日期\_序号.sql：创建或修改函数的sql脚本  
| sp\_开发人员姓名\_日期\_序号.sql：创建或修改存储过程的sql脚本  
| 每当DEV库表、数据或存储过程/函数有变更时，应及时将相应sql脚本记录于SVN共享目录；每次打包会备份此文件夹sql脚本后，并清空该文件夹  
--------------------------------------------------------------------------------------------------------------\*/  
  
--新建表  
--字段类型后用NULL和NOT NULL表示是否可为空|DEFAULT后加入字段默认值|若主键设置为自增可加入AUTO\_INCREMENT|PRIMARY KEY定义主键|COMMENT后加入表名备注  
**CREATE TABLE `example\_table`** (  
*`id`* bigint(**20**) unsigned **NOT NULL** COMMENT '示例表ID',  
*`example\_column`* **varchar**(**32**) **NULL DEFAULT NULL** COMMENT '示例字段',  
**PRIMARY KEY** (*`id`*)  
) COMMENT = '示例表';  
  
--删除表  
**DROP TABLE `example\_table`**;  
  
--添加字段  
--字段类型后用NULL和NOT NULL表示是否可为空|DEFAULT后加入字段默认值|COMMENT后加入字段备注|AFTER `example\_column`表示在字段example\_column后加入新字段  
**ALTER TABLE `example\_table` ADD COLUMN** *`example\_column2`* **varchar**(**32**) **NULL DEFAULT NULL** COMMENT '示例字段2' AFTER `example\_column`;  
  
--修改字段名  
**ALTER TABLE `example\_table`** CHANGE **COLUMN** `example\_column` `example\_column1` **varchar**(**32**) **NULL DEFAULT NULL** COMMENT '示例字段1' AFTER `id`  
  
--修改字段其他信息  
**ALTER TABLE `example\_table`** MODIFY **COLUMN** `example\_column` **varchar**(**64**) **NOT NULL DEFAULT** '' COMMENT '示例字段' AFTER `id`;  
  
--删除字段  
**ALTER TABLE `example\_table` DROP COLUMN** example\_column;  
  
--添加唯一索引  
**ALTER TABLE `example\_table` ADD UNIQUE** INDEX `uk\_example\_column1`(`example\_column1`);  
--添加普通索引  
**ALTER TABLE `example\_table` ADD** *INDEX* `idx\_example\_column2`(`example\_column2`);  
--添加联合索引  
**ALTER TABLE `example\_table` ADD** *INDEX* `idx\_example\_column`(`example\_column1`, `example\_column2`);  
  
--删除索引  
**ALTER TABLE `example\_table` DROP** INDEX `idx\_example\_column2`;

第一天: 大型WEB架构设计

第二天: LVS网络负载均衡

第三天: Squid代理服务器

第四天: Mysql主从复制和分区技术

第五天: Mysql性能优化

第六天: Mrtg网络监控和Shell邮件报警

第七天: Cacti网络监控,邮件报警和LAMP压力测试

第八天: Sphinx全文索引

第九天: Mongodb数据库上

第十天: Mongodb数据库下

# (一).MySQL数据库优化:

1. Mysql基础操作
2. 常用的SQL技巧
3. SQL语句优化
4. MySQL数据库优化
5. Myisam表锁
6. Mysql服务器优化

## 一.Mysql基础操作

* 1. mysql表复制
  2. mysql索引
  3. mysql视图
  4. mysql内置函数
  5. mysql预处理语句
  6. mysql事务处理
  7. mysql存储
  8. mysql触发器
  9. 重排auto\_increment值

### 1.1.mysql表复制

创建表t1

**CREATE TABLE t1** (  
*id* **int** unsigned **not null** auto\_increment **PRIMARY key**,  
*name* **VARCHAR** (**30**)  
) COMMENT = '示例表';

复制表结构+复制表数据

**create table t2 like** t1;  
**insert into t2 select** *\** **from t1**;

### 1.2.mysql索引

#### 1.2.1.alter table用来创建普通索引,unique索引或primary key索引

**alter table TABLE\_NAME ADD** *INDEX* index\_name(column\_list);  
**alter table TABLE\_NAME ADD UNIQUE**(column\_list);  
**alter table TABLE\_NAME ADD PRIMARY KEY**(column\_list);

#### 1.2.2.create index

**CREATE** INDEX index\_name **ON TABLE\_NAME** (column\_list);  
**CREATE UNIQUE** INDEX index\_name **ON table\_name**(column\_list);

#### 1.2.3.drop index

**DROP** INDEX index\_name **ON TABLE\_NAME** ;

#### 1.2.4.alter table table\_name drop

**alter table table\_name DROP** INDEX index\_name;  
**alter table TABLE\_NAME DROP PRIMARY KEY**;

#### \*.如果主键上有自增属性,则不能用drop直接删除主键索引;首先去掉自增属性后,方可删掉主键索引!

**alter table t1** modify id **int** unsigned **not null**;  
**alter table t1 DROP PRIMARY KEY**;

### 1.3.mysql视图

#### 1.3.1.创建视图

**create view** v\_t1 **as select** *\** **from t1 where** *id* > **4 and** *id* < **8**;

#### 1.3.2.视图的帮助信息

? **view**

#### 1.3.3.查看视图

show tables;

#### 1.3.4.删除视图

**drop view** v\_t1;

#### \*.如果视图v\_t1所依赖的表t1数据发生损坏,或者被删除,则视图v\_t1也会发生错误!

### 1.4.mysql内置函数

#### 1.4.1.字符串函数



#### 1.4.2.数学函数



#### 1.4.3.日期和时间函数



### 1.5.mysql预处理语句

设置stmt1预处理,传递一个数据作为一个where判断条件:

**prepare** stmt1 **from** “select *\** from **t1** where *id* > ?”;

设置一个变量

**set** @i = **1**;

执行stmt1预处理:

**EXECUTE** stmt1 **using** @i;

设置@i为5:

**set** @i = **5**;

再次去执行stmt1:

**EXECUTE** stmt1 **using** @i;

如何删除预处理stmt1:

**drop prepare** stmt1;

### 1.6.mysql事务处理

关闭自动提交功能

**set** autocommit = **0**;

从表t1中删除了一条记录

**delete from t1 where** *id* = **11**;

此时做一个p1还原点

savepoint p1;

再次从表t1中删除一条记录

**delete from t1 where** *id* = **10**;

再次做一个p2还原点

savepoint p2;

此时恢复到p1还原点,当然后面的p2这些还原点自动会失效

**rollback to** p1;

退回到最原始的还原点

**ROLLBACK** ;

#### \*.ENGINE =MyISAN引擎是不支持事务机制的!

修改表引擎,让其使用可以支持事务的Innodb;

**alter table t1** engin=innodb;

### 1.7.mysql存储

创建一个存储p1();

\d //  
**create procedure** p1()  
**begin  
set** @i=**0**;  
while @i<**10** do  
**select** @i;  
**set** @i=@i+**1**;  
**end** while;  
**end**;  
//

执行存储过程p1();

\d;  
call p1();

查看procedure p1()的status信息

show **PROCEDURE** status\G

查看procedure p1()的具体信息

show **create PROCEDURE** p1\G

### 1.8.mysql触发器

修改delimiter为//(语句终结符改为//)

\d //

#### 创建一个名字为tg1的触发器,当向t1表中插入数据时,就向t2表中插入一条数据

**create** trigger tg1 before **insert on** t1 **for** each row  
**BEGIN  
insert into t2**(id) **values**(new.id);  
**end**//

语句终结符该为分号;

\d;

准备两个空表t1和t2

**select** *\** **from t1**;  
**select** *\** **from t2**;

向t1表中插入多条数据:

**insert into t1 values**(**1**),(**2**),(**3**),(**4**);

#### 如何制作深处表t1后t2表中的记录也会跟着删除呢?

\d //

**create** trigger tg2 before **delete on** t1 **for** each row  
**begin delete from t2 where** id=old.id;  
**end**//  
  
\d;

#### 如何在制作更改表t1后t2表中的记录跟着修改呢?

\d //  
**create** trigger tg2 before **update on** t1 **for** each row  
**begin update t2 set** id =new.id **where** id=old.id;  
**end**//  
  
\d;

#### 查看触发器:

show triggers;

### 1.9.重排auto\_increment值

mysql数据库自动增长的ID如何恢复

清空表的时候不能用

**delete from table\_name**;

而要用:

truncate **table table\_name**;

这样auto\_increment就恢复成1了;

或者清空内容后直接用alter命令修改表;

**alter table table\_name** auto\_increment = **1**;

## 二.常用的sql技巧

2.1.正则表达式的使用

2.2.巧用rand()提取随机行

2.3.利用GROUP BY 的WITH ROLLUP子句统计

2.4.用BIT GROUP FUNCTIONS做统计

2.5.使用外键需要注意的问题

2.6.mysql中help的使用

### 2.1.正则表达式的使用

mysql利用regexp命令提供给用户扩展的正则表达式功能,具体模式序列如下:



使用正则表达式$和[...]进行匹配

**select name**,email **from** t **where** email REGEXP "@163[.,]com$";

使用like方式查询:

**select name**,email **from** t **where** email **like** "%@163.com" **or** email **like** "%@163,com";

### 2.2.巧用rand()提取随机行

mysql数据库中有一个随机函数rand()是获取一个0-1之间的数,利用这个函数一起和order by能够把数据随机排序.

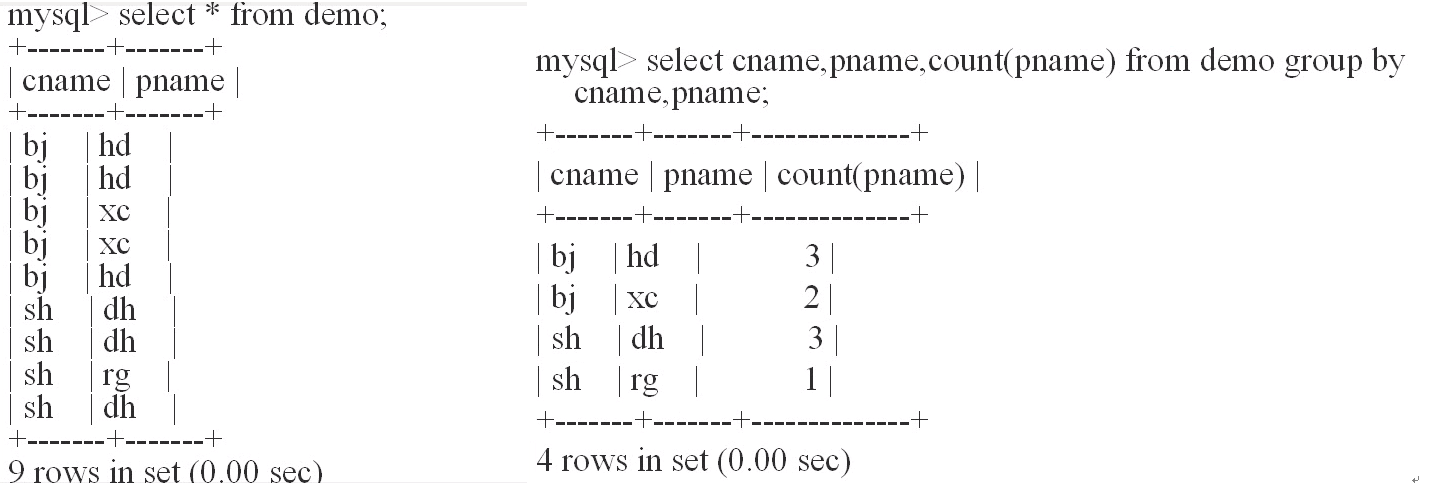
**select** *\** **from** stu **order by** rand();

下面是通过limit随机抽取了3条数据样本:

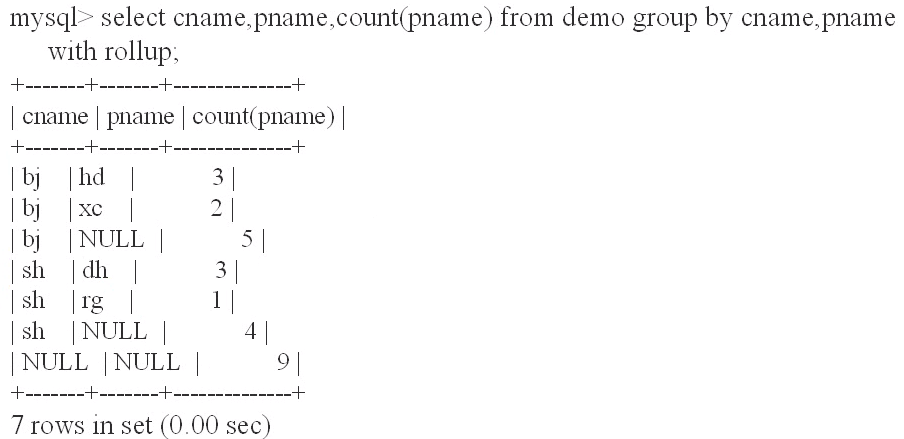
**select** *\** **from** stu **order by** rand() limit **3**;

### 2.3.利用GROUP BY 的WITH ROLLUP子句统计

使用group by的with rollup子句可以检索出更多的分组聚合信息.



**注意:with rollup不可以和order by同时使用.**



### 2.4.用BIT GROUP FUNCTIONS做统计

在使用group by语句时可以同时使用bit\_and,bit\_or函数来完成统计工作.这两个函数的作用主要是做数值之间的逻辑位运算.

**select** id, **bit\_or**(kind) **from** order\_rab **group** id;

对order\_rab表中id分组时对kind做位与和或计算.

**select** id, **bit\_and**(kind) **from** order\_rab **group** id;

### 2.5.使用外键需要注意的问题

创建外键的方式如下:

**create table temp**(*id* **int**,*name* **char**(**20**),**FOREIGN key**(*id*) **REFERENCES** outTable(*id*) **on delete cascade on update cascade**);

**注意:Innodb类型的表支持外键,myisam类型的表,虽然创建外键可以成功,但是不起作用,主要原因是不支持外键.**

### 2.6.mysql中help的使用



## 三.SQL语句优化

3.1.优化sql语句的一般步骤

3.2.索引问题

3.3.两个简单实用的优化方法

3.4.常用的SQL的优化

### 3.1.优化sql语句的一般步骤

#### 3.1.1.通过show status命令了解各种SQL的执行效率.

格式:

show [**session** | **global**] status;

其中:session (默认)表示当前连接;

global表示自数据库启动至今.

show status;  
show **global** status;  
show status **like** 'Com\_%';  
show **global** status **like** 'Com\_%';

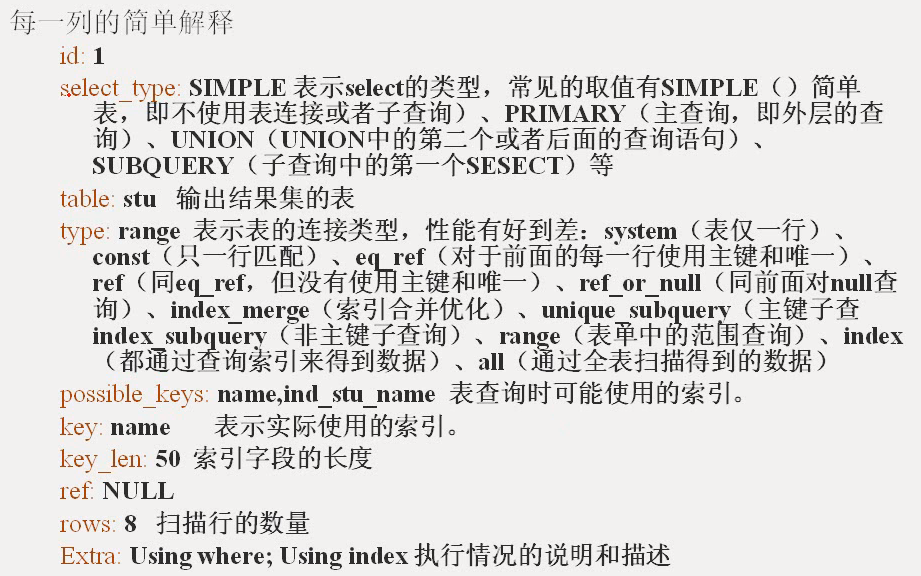
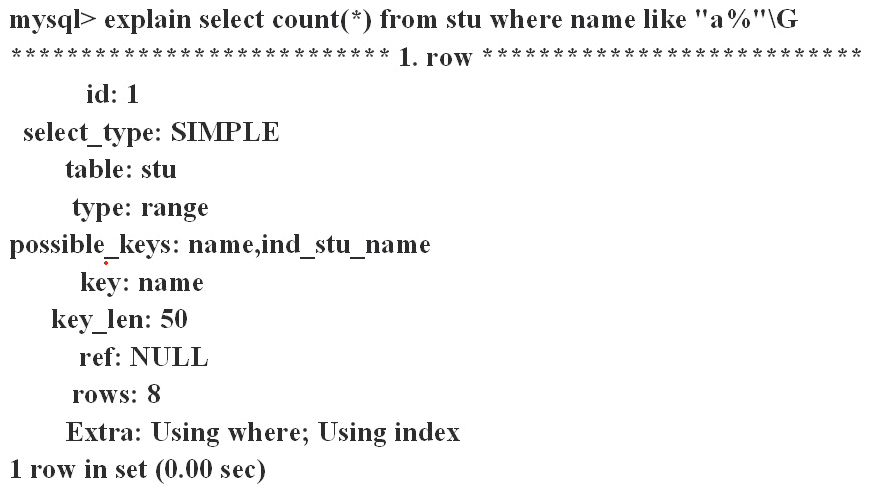


#### 3.1.2.定位执行效率较低的SQL语句:

**1**).explain **select** *\** **from table where** id = **1000**;  
**2**).desc **select** *\** **from table where** id = **1000**;

**desc select** *\** **from table where** id = **1000**\G;

#### 3.1.3.通过EXPLAIN分析较低效率SQL的执行计划



### 3.2.索引问题

索引是数据库优化中最常见也是最重要的手段之一,通过索引通常可以帮助用户解决大多数的SQL性能问题.

#### 3.2.1.索引的存储分类

MYISAM存储引擎的表的数据和索引是自动分开存储的,各自是独一的一个文件,InnoDB存储引擎的表的数据和索引是存储在同一个表空间里面,但可以有多个文件组成.

Mysql目前不支持函数索引,但是能对列的前面某一部分进行索引,例如name字段,可以只取name的前4个字符进行索引,这个特性可以大大缩小索引文件的大小,用户在设计表结构的时候也可以对文本列根据此特性进行灵活设计.

**create** index ind\_company2\_name **on** company2(**name**(**4**));

其中company表名ind\_company2\_name索引名;

#### 3.2.2.Mysql如何使用索引

索引用于快速找出在某个列中有一特定值的行.对相关列使用索引是提高SELECT操作性能的最佳途径.

1.使用索引

(1).对于创建的多列索引,只有查询的条件中用到最左边的列,索引一般就会被使用.如下创建一个复合索引:

**create** index ind\_sales2\_com\_mon onsales2(company\_id,moneys);

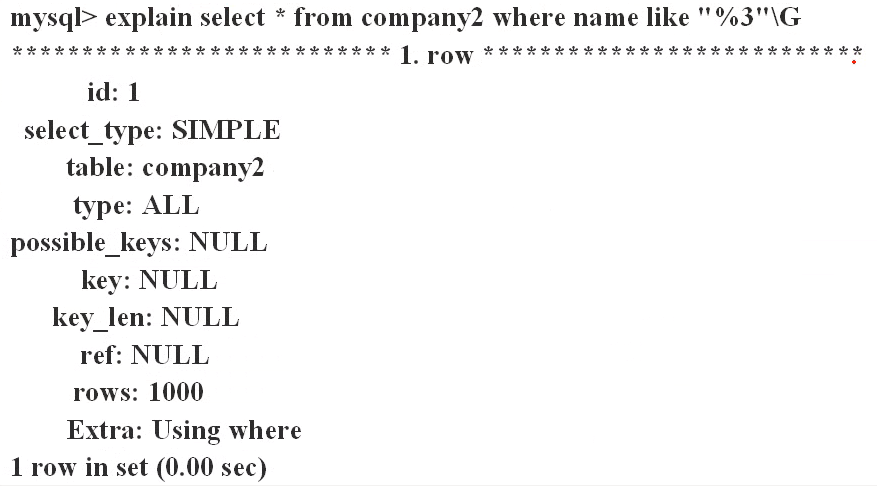
然后按company\_id进行查询,发现使用到了复合索引.

explain **select** *\** **from** sales2 **where** company\_id=**2006**\G;

使用下面的查询就没有使用到复合索引.

explain **select** *\** **from** sales2 **where** moneys=**1**\G;

(2).使用like的查询,后面如果是常量并且只有%不在第一个字符,索引才可能会被使用,如下:



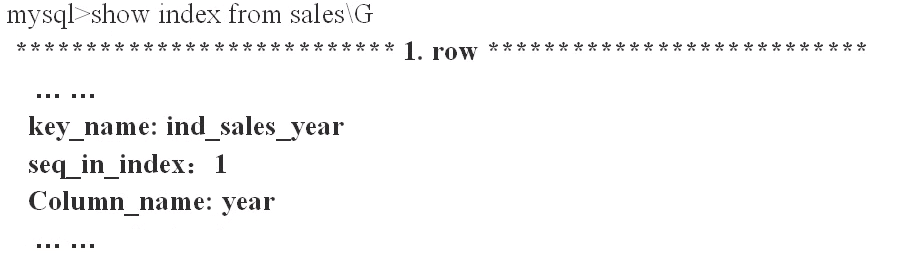
2.存在索引但不使用索引

(1).如果mysql估计使用索引比全表扫描更慢,则不使用索引.例如如果列key\_part1均匀分布在1到100之间,查询时使用索引就不是很好

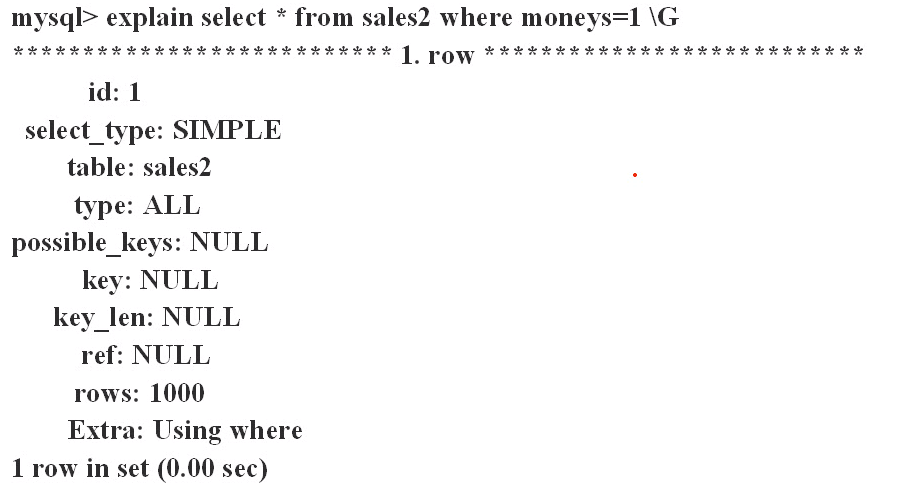
**select** *\** **from table\_name where** key\_part1 > **1 and** key\_part1 < **90**;

(2).如果使用MEMORY/HEAP表并且where条件中不使用”=”进行索引列,那么不会用到索引.HEAP表只有在”=”的条件下会使用索引.

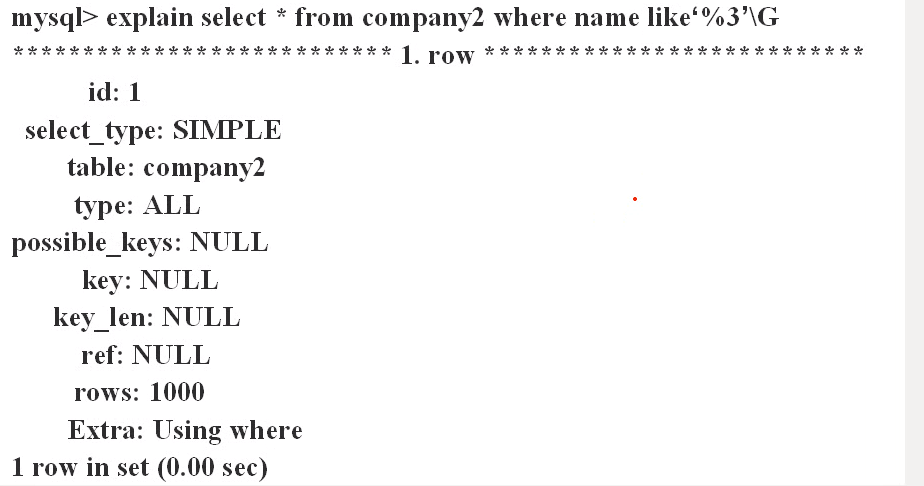
(3).用or分割开的条件,如果or前的条件中的列有索引,而后面的列中没有索引,那么设计的索引都不会被用到.



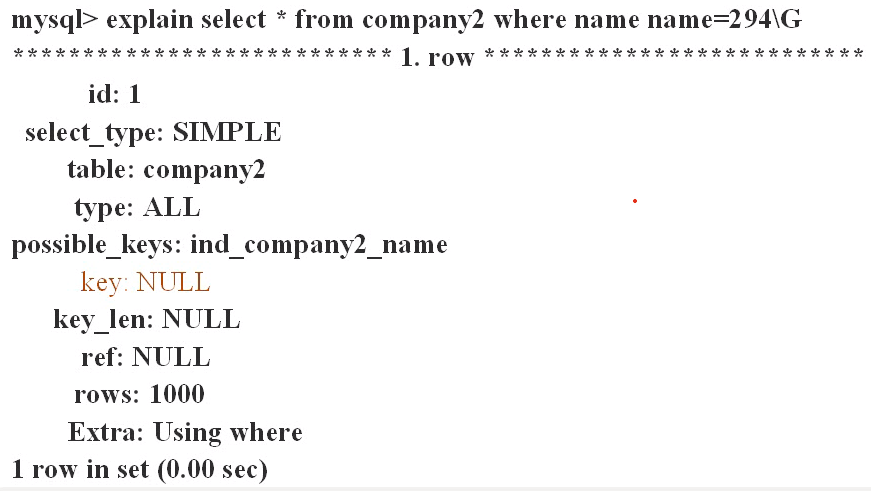
(4).如果不是所有列的第一部分,如下例子:可见虽然在money上面建有复合索引,但是由于money不是索引的第一列,那么在查询中这个索引也不会被MYSQL采用.



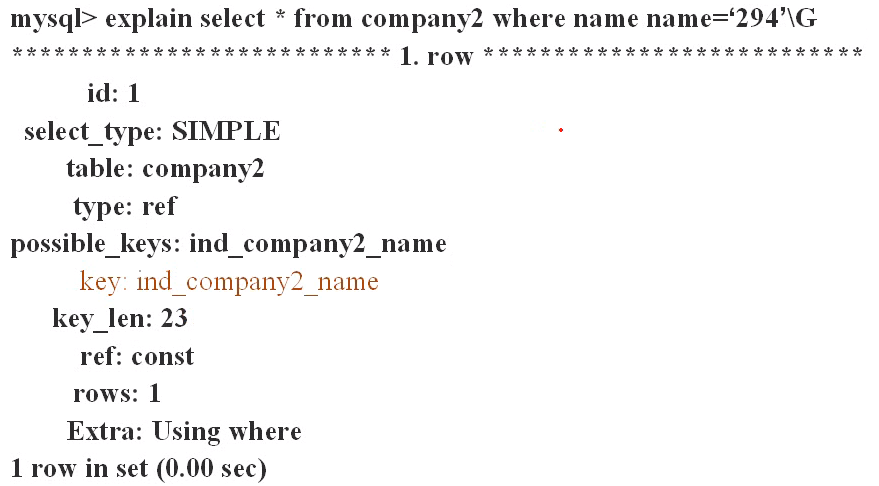
(5).如果like是以%开始,可见虽然在name上面建有索引,但是由于条件中like的值的”%”在第一位了,那么MYSQL也不会采用这个索引.



(6).如果列类型是字符串,但在查询时把一个数值类型常量赋值给了一个字符型的列名name,那么虽然在name列上有索引,但是也没有用到.



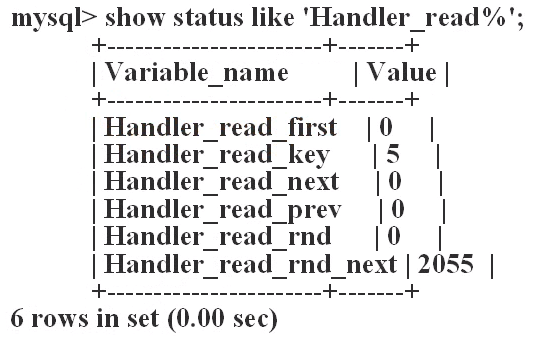
而下面的sql语句就可以正确使用索引.



3.2.3.查看索引使用情况

如果索引正在工作,Handler\_read\_key的值将很高,这个值代表一个行被索引值读的次数.

Handler\_read\_rnd\_next的值最高则意味着查询运行低效,并且应该建立索引补救.



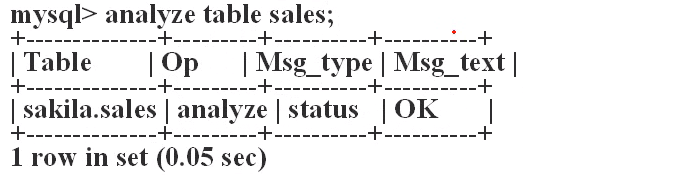
### 3.3两个简单实用的优化方法

对于大多数开发人员来说,可能只希望掌握一些简单实用的优化方法,对于更多更复杂的优化,更倾向于交给作业DBA来做.

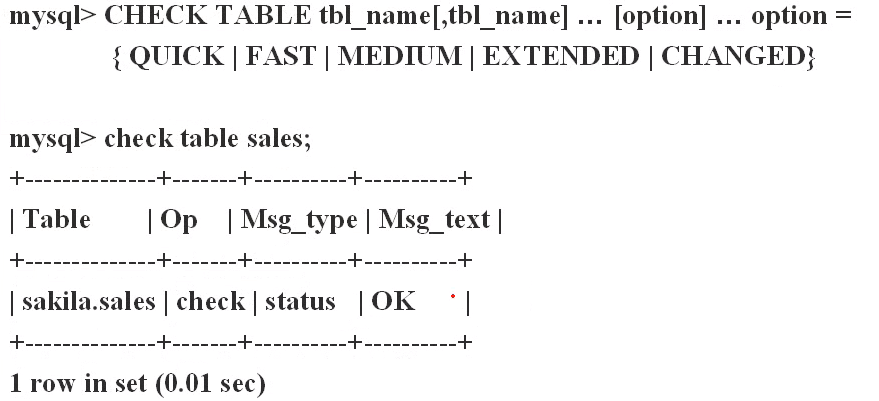
#### 3.3.1.定期分析表和检查表

ANALYZE [**LOCAL** | NO\_WRITE\_TO\_BINLOG] **TABLE** tal\_name[,tbl\_name]...

本语句用于分析和存储表的关键字分布,分析的结果 将可以使得系统得到准确的统计信息,使得SQL能够生成正确的可行计划.



分析表的语法如下(检查一个或多个表是否有错误)

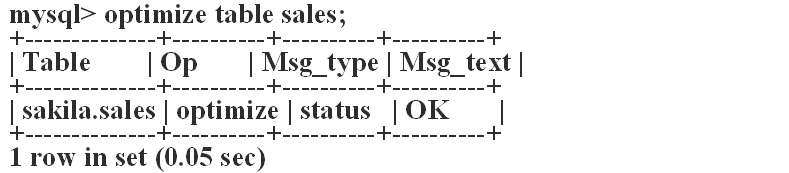


#### 3.3.2.定期优化表

优化表的语法格式:

OPTIMIZE [**LOCAL** | NO\_WRITE\_TO\_BINLOG] **TABLE** tbl\_name[,tbl\_name];

如果已经删除了表的一大部分,或者如果已经对含有可变长度行的表进行了很多的改动,则需要做定期优化.这个命令可以将表中的空间碎片进行合并,但是此命令只对MyISAM,BDB和InnoDB表起作用.



### 3.4.常用SQL的优化

本节将针对这些SQL语句介绍一些优化的方法.

#### 3.4.1.大批量插入数据

当用load命令导入数据的时候,适当设置可以提高导入的速度.

对于MyISAM存储引擎的表,可以通过以下方式快速的导入大量的数据.

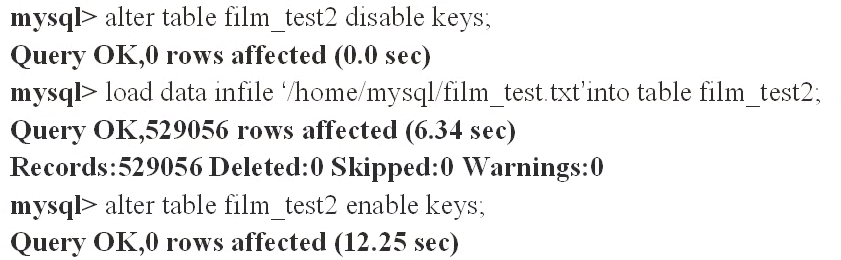


DISABLE KEYS和ENABLE KEYS 用来打开或关闭MyISAM表非唯一索引的更新,可以提高速度,**注意:对InnoDB表无效.**

没有使用打开或关闭MyISAM表非唯一索引:

load **data** infile '/home/mysql/film\_test.txt' **into table** film\_test2;

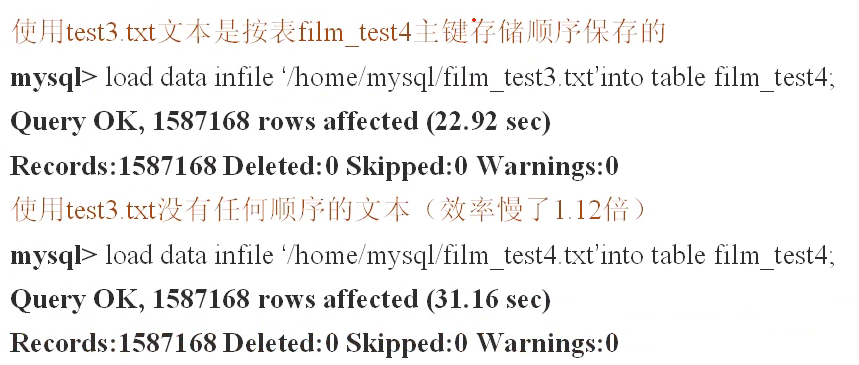
使用打开或关闭MyISAM表非唯一索引:



以上对MyISAM表的数据导入,但是对于InnoDB表并不能提高导入数据的效率;

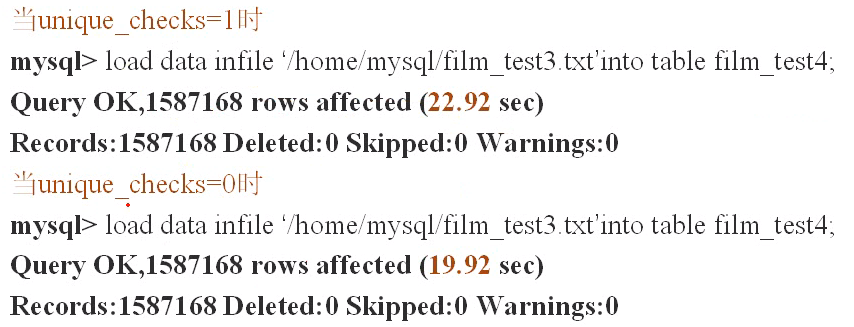
(1).针对于InnoDB类型的表数据导入的优化

因为InnoDB表的按照主键顺序保存的,所以将导入的数据主键的顺序排列,可以有效地提高导入数据的效率.



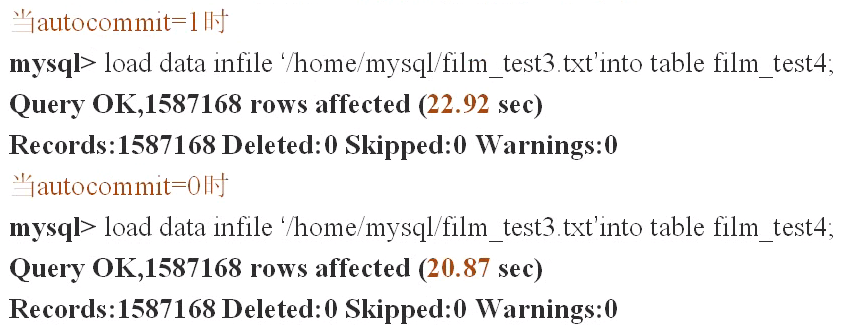
(2).关闭唯一性校验可以提高导入效率

在导入数据前先执行set unique\_checks=0,关闭唯一性校验,在导入结束后执行set unique\_checks=1,恢复唯一性校验,可以提高导入效率.



(3).关闭自动提交可以提高导入效率

在导入数据前先执行set autocommit=0,关闭自动提交事务,在导入结束后执行set autocommit=1,恢复自动提交,可以提高导入效率.



#### 3.4.2.优化insert语句

尽量使用多个值表的insert语句,这样可以大大缩短客户与数据库的链接,关闭等损耗.

可以使用insert delayed(马上执行)语句得到更高的效率.将索引文件和数据文件分别存放在不同的磁盘上.

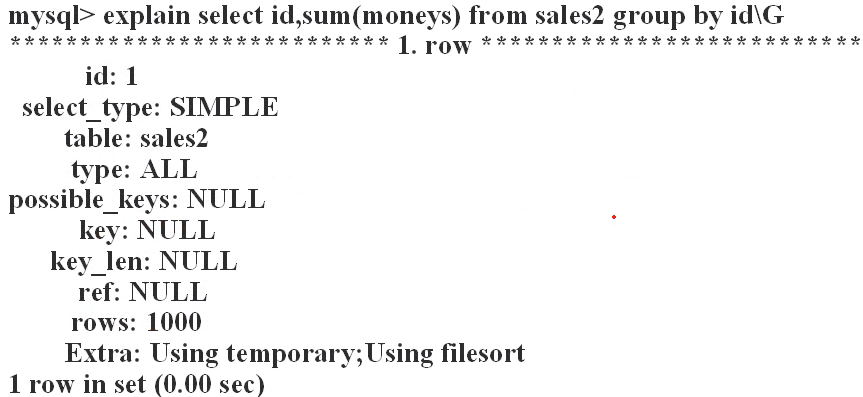
可以增加bulk\_insert\_buffer\_size变量值的方法来提高速度.但只对MyISAM表使用.

当从一个文件装载一个表时,使用LOAD DATA INFILE.这个通常比使用很多insert语句要快20倍.

#### 3.4.3.优化group by语句

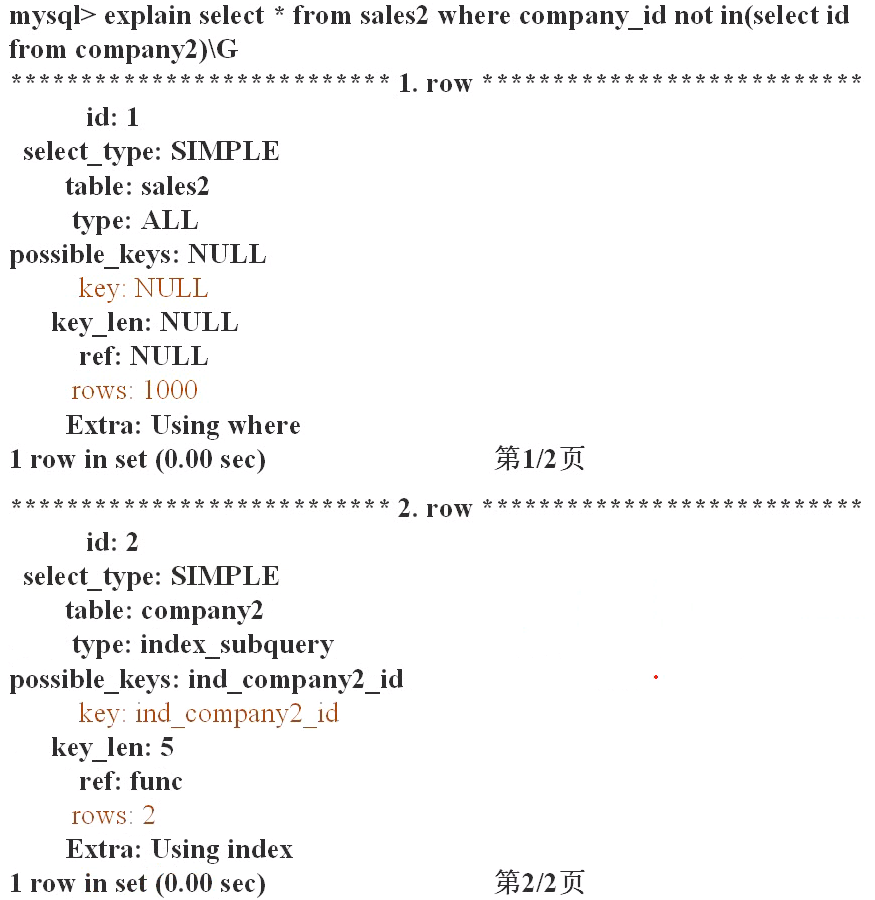
如果查询包含group by但用户想要避免排序结果的损耗,则可以使用order by null来禁止排序:

如果没有使用order by null 来禁止排序

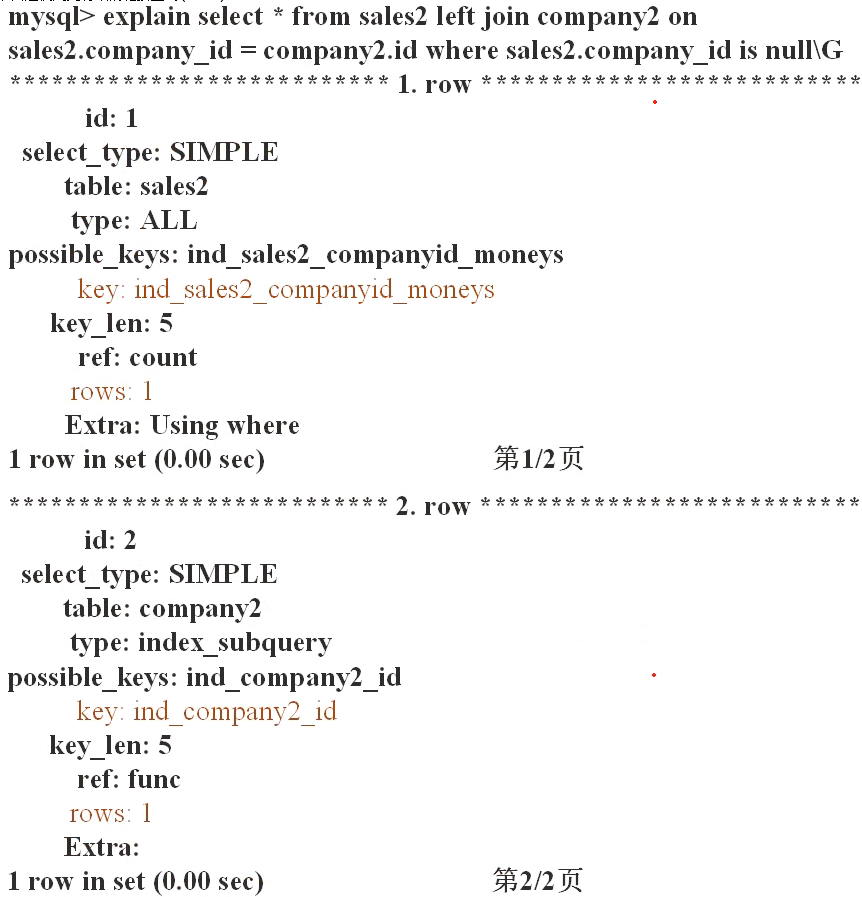


#### 3.4.5.优化嵌套查询

下面是采用嵌套查询的效果(可以使用更有效的链接查询(Join)替代).



下面是使用更有效的链接查询(Join).



从执行计划中可以明显看出查询扫描的记录范围和使用索引的情况都有了很大的改善.连接(Join)之所以更有效率一些,是因为MySQL不需要在内存中创建临时表来完成这个逻辑上的需要两个步骤的查询工作.

## 四.数据库优化

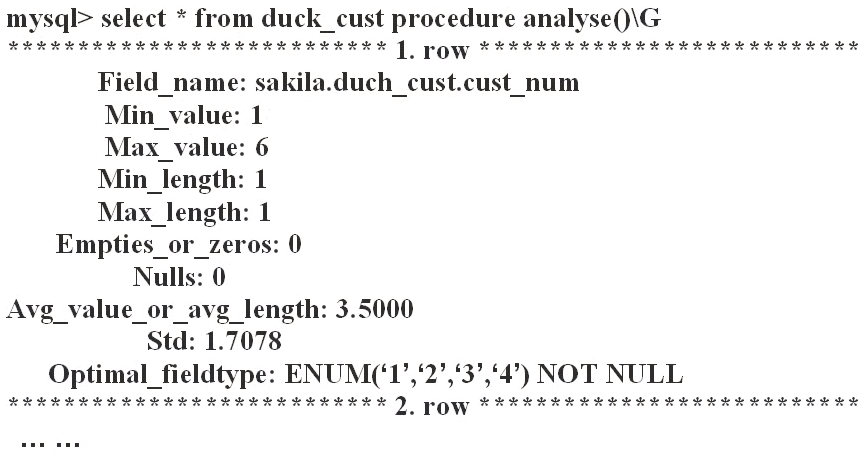
4.1.优化表的类型

4.2.通过拆分提高表的访问效率

4.3.使用中间表提高统计查询速度.

### 4.1.优化表的类型

在Mysql中,可以使用函数PROCEDURE ANALYSE()对当前应用的表进行分析,



### 4.2.大存储量解决

1.分库分表

2.分区

主要目的:

1. 减少表的记录数
2. 减少对操作系统的负担压力

### 4.3.中间表

中间表的产生:

1. view视图
2. 重新生成一个新表

## 五.MySQL服务器的优化

1.myisam读锁定

2.myisam写锁定

### 5.1.myisam读锁定

1.locak table t1 read;

2.开启另一个mysql连接终端,接着去尝试:select \* from t1;

3.再insert,update和delete t1这张表,你会发现所有的数据都停留在终端上没有真正的去操作

4.读锁定对我们做备份大量数据时非常有用.

mysqldump –uroot –p123 test>test.sql



# (二).\*\*\*\*\*\*\*\*\*数据库调优教程汇总\*\*\*\*\*\*\*\*\*\*\*

数据库调优教程（一） 前言&慢查询定义  
<http://blog.csdn.net/hzy38324/article/details/44903207>  
数据库调优教程（二） 慢查询数据准备  
<http://blog.csdn.net/hzy38324/article/details/44917119>  
数据库调优教程（三） 记录慢查询  
<http://blog.csdn.net/hzy38324/article/details/44917241>  
数据库调优教程（四） Explain性能分析命令  
<http://blog.csdn.net/hzy38324/article/details/44921299>  
数据库调优教程（五） 索引的作用和索引的种类  
<http://blog.csdn.net/hzy38324/article/details/44922651>  
数据库调优教程（六） 索引的相关操作  
<http://blog.csdn.net/hzy38324/article/details/44937625>  
数据库调优教程（七） 索引的代价  
<http://blog.csdn.net/hzy38324/article/details/44942891>  
数据库调优教程（八） 什么情况下不要使用索引  
<http://blog.csdn.net/hzy38324/article/details/44962055>  
数据库调优教程（九） 添加了索引但不被使用的几种常见可能  
<http://blog.csdn.net/hzy38324/article/details/44962407>  
数据库调优教程（十） 解决like’%str’时索引不被使用的4种方法  
<http://blog.csdn.net/hzy38324/article/details/44975859>  
数据库调优教程（十一） 设计一张漂亮的表  
<http://blog.csdn.net/hzy38324/article/details/44992649>  
数据库调优教程（十二） 优化sql语句  
<http://blog.csdn.net/hzy38324/article/details/44992759>  
数据库调优教程（十三） MySQL数据库其他优化方法  
<http://blog.csdn.net/hzy38324/article/details/44994177>  
  
关于数据库的文章发表到此告一段落，但是学习数据库的脚步不会停下，欢迎广大软件开发爱好者、专家、大神和JavDroider交流！！共勉！！共习！！！共进步！！！！

## **前言**

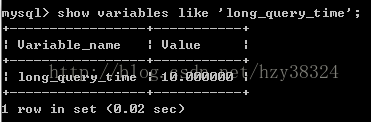
最近帮公司优化数据库，凭着之前所学，一步一步地将学习知识用于实践，总算是将速度蹭上去了，一个原本要执行1分多钟的查询现在只需要3秒。  
现把自己所学所思及所用加以总结，一方面为自己巩固知识，另一方面也给广大同学以一点点参考。  
  
本套教程共十三讲。  
在这系列的教程中，你将会学到  
1. 如何一步步地优化数据库  
2. 如何把索引的作用发挥的淋漓尽致  
3. 如何解决Mysql在like’%xxx%’模糊查询的情况下不使用索引  
4. 如何设计一张优质的表  
5. 如何写出高质量的sql语句  
6. 数据库优化的其他方法  
  
首先，让我们从发现慢查询开始。

## 一、 发现慢查询

如何从一个大项目中，迅速的定位执行速度慢的语句，这是本章节将要解决的问题。

### **1. 慢查询的定义**

怎样的查询才算是慢查询，有没有一个量化的标准呢？  
慢查询定义  
慢查询是指执行时间超过慢查询时间的sql语句。  
查看慢查询时间的方法  
show variables like 'long\_query\_time';   
可以显示当前慢查询时间。MySql默认慢查询时间为10秒



可以通过如下语句对慢查询的定义进行修改  
set global long\_query\_time=1;



（如果你的mysql设置了缓存，那么需要重新进入命令行窗口才会查出变化）  
需要注意的是，这个语句特意在变量前加上了global，表明这次的设置是对整个Mysql有效的，而默认情况下变量前的修饰符是session（会话），也就是只对当前窗口有效。  
这一讲只是开个头，下一讲，我们会为慢查询的发生准备数据，即创建一张大表。

上一讲我们谈论了慢查询的定义，这一讲我们来创建一张大表，为慢查询做数据准备。

### **2. 慢查询数据准备**

要想发现慢查询，首先要使慢查询发生。在一张普通数量级的表格中是不能发生慢查询的，除非你对于慢查询的定义时一个毫秒。因此我们必须手动创建一张大数量级的表，这里选择创建一张40万数量级的表（同学们也可以创建百万级的，如果你们的电脑很厉害。但是一般情况下，十万级的数据就可以看出慢查询了）。

#### 1) 创建数据库

Create database bigTable default character set GBK;

#### 2) 创建表

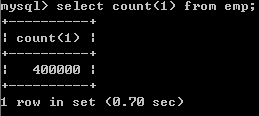
#部门表#  
CREATE TABLE dept(   
id int unsigned primary key auto\_increment,   
deptno MEDIUMINT UNSIGNED NOT NULL DEFAULT 0,   
dname VARCHAR(20) NOT NULL DEFAULT "",   
loc VARCHAR(13) NOT NULL DEFAULT ""   
) ENGINE=INNODB DEFAULT CHARSET=GBK ;   
  
  
#雇员表#  
CREATE TABLE emp  
(  
id int unsigned primary key auto\_increment,  
empno MEDIUMINT UNSIGNED NOT NULL DEFAULT 0, /\*编号\*/  
ename VARCHAR(20) NOT NULL DEFAULT "", /\*名字\*/  
job VARCHAR(9) NOT NULL DEFAULT "",/\*工作\*/  
mgr MEDIUMINT UNSIGNED NOT NULL DEFAULT 0,/\*上级编号\*/  
hiredate DATE NOT NULL,/\*入职时间\*/  
sal DECIMAL(7,2) NOT NULL,/\*薪水\*/  
comm DECIMAL(7,2) NOT NULL,/\*红利\*/  
deptno MEDIUMINT UNSIGNED NOT NULL DEFAULT 0 /\*部门编号\*/  
)ENGINE=INNODB DEFAULT CHARSET=GBK ;

#### 3) 创建函数

函数用于随机产生数据，保证每条数据都不同  
#函数1 创建#  
#创建函数. 用于随机产生字符串。该函数接收一个整数  
  
delimiter $$#定义一个新的命令结束符合  
create function rand\_string(n INT)   
returns varchar(255) #该函数会返回一个字符串  
begin   
#chars\_str定义一个变量 chars\_str,类型是 varchar(100),默认值'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLMNOPQRSTUVWXYZ';  
declare chars\_str varchar(100) default  
'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLMNOPQRSTUVWXYZ';  
declare return\_str varchar(255) default '';  
declare i int default 0;  
while i < n do   
set return\_str =concat(return\_str,substring(chars\_str,floor(1+rand()\*52),1));  
set i = i + 1;  
end while;  
return return\_str;  
end $$  
  
#函数2创建#  
#用于随机产生部门编号  
create function rand\_num( )  
returns int(5)  
begin   
declare i int default 0;  
set i = floor(10+rand()\*500);  
return i;  
end $$

#### 4) 创建存储过程

#存储过程一#  
#该存储过程用于往emp表中插入大量数据  
create procedure insert\_emp(in start int(10),in max\_num int(10))  
begin  
declare i int default 0;   
#set autocommit =0 把autocommit设置成0  
set autocommit = 0;   
repeat  
set i = i + 1;  
insert into emp (empno, ename ,job ,mgr ,hiredate ,sal ,comm ,deptno ) values ((start+i) ,rand\_string(6),'SALESMAN',0001,curdate(),2000,400,rand\_num());  
until i = max\_num  
end repeat;  
commit;  
end $$  
  
执行存储过程，往emp表添加40万条数据  
call insert\_emp(100001,400000);  
  
查询，发现Emp表插入了40万条记录



#存储过程二#  
#往dept表添加随机数据  
create procedure insert\_dept(in start int(10),in max\_num int(10))  
begin  
declare i int default 0;   
set autocommit = 0;   
repeat  
set i = i + 1;  
insert into dept (deptno ,dname,loc ) values ((start+i) ,rand\_string(10),rand\_string(8));  
until i = max\_num  
end repeat;  
commit;  
end $$  
  
执行存储过程二  
delimiter ;  
call insert\_dept(100,10);

至此，数据准备完成。我们创建了大表emp。

下一讲，我们将利用已创建的大表去发现慢查询，并试着把慢查询记录到日志中。

上一讲我们为慢查询的发生做了数据准备，这一讲就让我们来发现慢查询，同时把它记录到文件中。

### **3. 记录慢查询**

此时我们已经有让慢查询发生的成本了。执行以下语句，你就知道什么叫慢！查！询！

select empno from emp where ename='';

file://C:\Users\ray\AppData\Local\Temp\ct_tmp/1.png

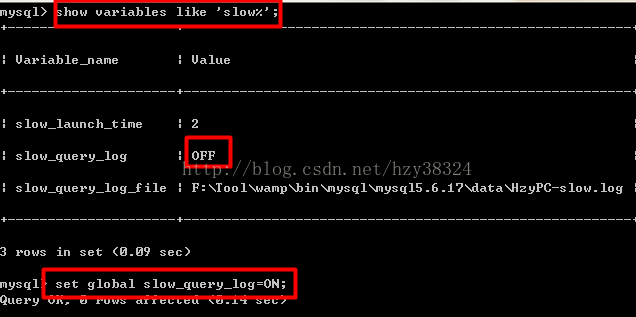
一个很明显找不到结果的查询语句居然也执行了近3秒钟。

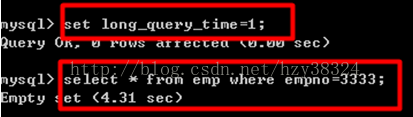
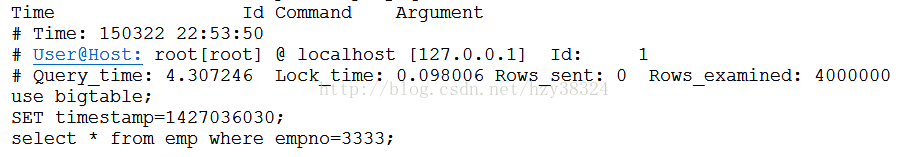
这时候，作为DBA就应该把这个sql语句记录下来，是记在记事本还是写在笔记本呢？不用想太多了，不用你自己记，Mysql提供了慢查询日志功能，自动帮你记录慢查询的语句。

1) 把慢查询的sql记录到日志中  
首先你要打开慢查询日志文件记录器  
使用  
show variables like 'slow%';

你会发现默认情况下慢查询日志记录器关闭的

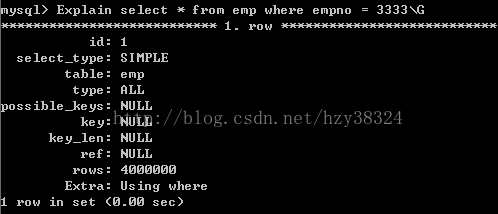
使用

set global slow\_query\_log=ON;   
打开之  


此时你会发现mysql安装目录下的data文件夹中出现了以你本机名命名的日志文件  
  
此时再执行慢查询操作  
  
打开日志文件发现出现记录  
  
以后只需要定期检查日志文件就可以找到慢查询语句了  
注意：  
在找到慢查询语句后，要通过反复使用select语句确认慢查询，注意只能使用select语句，就算是原来语句为delete或者update等，也要用select代替，因为只有select不会弄脏数据库  
2) 另一种发现慢查询语句的方法  
要是你使用Hibernate进行J2ee开发，可以使用一下方式。  
在页面中进行操作，当发现某个操作的响应较慢时，查看Eclipse控制台的Hibernate输出sql语句，此语句即为慢查询语句。  
  
本章结束，下一章我们将讲解如何使用mysql的性能分析命令Explain。

上一章我们将了如何发现慢查询并把语句记录到日志中，那么在发现慢查询后，要怎样才能知道语句的问题发生在哪里。本章将介绍怎样使用Mysql提供的数据库性能分析命令，对sql语句进行分析。

## 二、数据库性能分析Explain命令

Explain的作用是生成一个QEP（查询执行计划），可以帮助我们在不真正执行某个sql语句时，就看到mysql怎样执行，这样方便我们去分析sql指令的好坏。  
执行如下语句  
Explain select \* from emp where empno = 3333\G   


对于返回的信息，我们主要关心一下几个

### 1) **Type**

ALL全表扫描,通常是不好的，其他的如index、range、const、ref、system则是较好的

### 2) **Possible\_keys**

可能被用到的索引

### 3) **Key**

查询过程中实际使用的索引，当为null时表示没有使用索引，通常是不好的

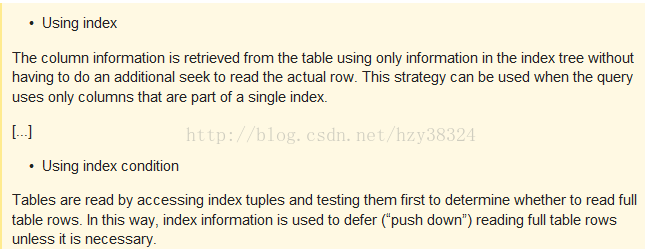
### 4) **key\_len**

索引字段最大可能使用的长度，也叫索引基数。索引基数越大，表明可能查找的行数越多，查询效率越慢。

### 5) **Rows**

MySQL 估计的需要扫描的行数。只是一个估计。越多表明查找的行数越多，自然越慢。

### 6) **Extra**

显示上述信息之外的其它信息，非常重要。其主要有一下返回结果。  
Usingindex  
表明此查询使用了覆盖索引(CoveringIndex)，即通过索引就能返回结果，无需访问表。（覆盖索引是一种非常优秀的索引，其使用见<http://blog.csdn.net/hzy38324/article/details/44857721>）  
若没显示"Usingindex"表示读取了表数据。  
Usingindex condition  
可能会使用索引  
Usingwhere  
表示 MySQL 服务器先读取整行数据，再检查此行是否符合 where 句的条件，符合就留下，不符合便丢弃。效率较慢。  
Usingfilesort  
表示Mysql会按查询所需的顺序对结果进行排序，这时就会出现 Usingfilesort 。排序自然会增加查询时间，导致效率变慢。解决方法是利用索引进行排序。若查询所需的排序与使用的索引的排序一致，因为索引是已排序的，因此按索引的顺序读取结果返回，此时就不会出现Using filesort。  
关于“Using index” 和 “Using index condition”的区别，笔者参考了一下stackoverflow上的一篇文章  
<http://stackoverflow.com/questions/1687548/mysql-explain-using-index-vs-using-index-condition>  
里面是这么解释的  
  
简单的说  
Using index就是一定使用索引，这种索引成为覆盖索引，Using index condition则是在必要的时候才使用索引  
怎样才能让Usingindex condition变成Using index，答案自然是创建一个覆盖索引，同样，笔者将会在之后章节介绍[覆盖索引](http://blog.csdn.net/hzy38324/article/details/44857721)如何创建。  
本章结束，下一章我们将全面讲解性能优化的利器——索引。

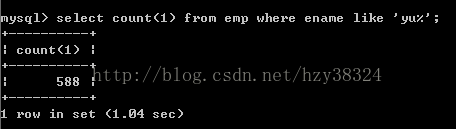
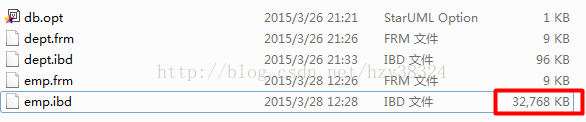
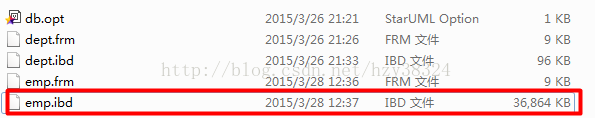
## 三、 索引

上一章我们学习了使用explain来生成一个查询执行计划（QEP），从而发现语句存在的问题。在explain返回的结果中，有三个是和索引有关的（possible key、key、extra），可见索引在改善查询效率上的显著地位。  
这一章，我们将全方位讲解如何使用索引来优化我们的数据库。

### 1. 索引的作用

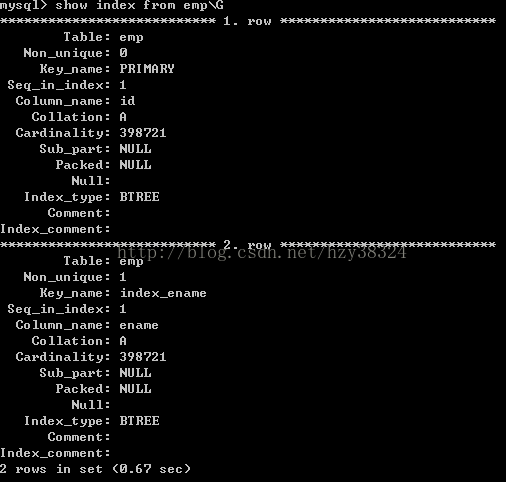
索引的原理是利用特殊的查找算法（如二叉树算法），限制访问的行数，提高访问效率。  
另外，索引还可以高效地在表之间建立连接操作，高效的对结果进行排序等等

### 2. 基本索引种类及创建

索引可以分为主键索引、普通索引、唯一索引和全文索引。  
1) 主键索引  
当一张表，把某个列设为主键的时候，则该列就是主键索引  
create table aaa   
(id int unsigned primary key auto\_increment ,   
name varchar(32) not null defaul ‘’);   
  
这是id 列就是主键索引.  
  
如果你创建表时，没有指定主键索引，也可以在创建表后再添加  
alter table 表名 add primary key (列名);   
2) 普通索引  
一般来说，普通索引的创建，是先创建表，然后在创建普通索引  
create index 索引名 on 表 (列名1,列名2);   
当括号中的列不止一个时，我们称之为多列索引，多列索引在一定情况下可以进化为覆盖索引（后面会介绍）。  
这里我们通过在emp表中创建普通索引来见识一下索引的威力。  
在未创建索引前  
我们执行以下语句  
select count(1) from emp where ename like 'yu%';   
  
  
接着我们创建索引  
create index index\_ename on emp(ename);   
file://C:\Users\ray\AppData\Local\Temp\ct_tmp/2.png  
再次执行  
select count(1) from emp where ename like 'yu%';   
  
**不可思议！  
索引的威力可见一斑！**  
  
当然索引是有开销的  
这是我在未创建索引时mysql安装目录下data/bigtable/emp.ibd文件的大小，对于innodb引擎的表来说，ibd文件存放了表的数据和索引。  
  
在我创建了索引之后  
  
**显然，索引会占用主机的空间，这也称作索引的开销。**  
3) 唯一索引  
当表的某列被指定为unique约束时，这列就是一个唯一索引  
create table ddd(id int primary key auto\_increment , name varchar(32) unique);   
这时, name 列就是一个唯一索引  
在创建表后，再去创建唯一索引  
create unique index 索引名 on 表名 (列表..);   
unique字段可以为NULL,并可以有多NULL,即Null！=Null。但是如果是具体内容，则不能重复  
4) 全文索引  
全文索引，只对MyISAM引擎有用。主要是针对对文件，文本的检索, 比如文章或者段落,.  
它会把某个数据表的某个数据列出现过的所有单词生成一份清单  
全文索引的创建  
在创建表时创建  
  
CREATE TABLE articles (   
id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,   
title VARCHAR(200),   
body TEXT,   
FULLTEXT (title,body)   
)engine=myisam charset utf8;   
  
全文索引正确的用法是:  
select \* from articles where match(title,body) against(‘要搜索的单词’);

另外，mysql自己提供的fulltext针对英文生效，想要搜索中文需要使用sphinx全文检索引擎或者使用[加强版的模糊查询](http://blog.csdn.net/hzy38324/article/details/44857721)。  
注意：  
全文索引不完全等同于模糊查询  
比如title字段有这么个数据’abcd20088ccaa’,使用模糊查询select \* from articleswhere title like’%2008%’可以查找到，而使用全文检索select \* from articleswhere match(title) against(‘2008’);是检索不到的，因为2008不是一个单词！  
  
本讲结束，下一讲将讲解索引的相关操作。

### 3. 索引的相关操作

上一讲我们介绍了索引的作用和种类，这一讲我们谈谈索引的相关操作！  
1) 添加索引  
前面已经有所介绍，这里总结一下  
  
添加主键索引  
create table aaa   
(id int unsigned primary key auto\_increment ,   
name varchar(32) not null defaul ‘’);   
  
alter table 表名 add primary key (列名);   
  
添加普通索引   
create index 索引名 on 表 (列名1,列名2);   
  
添加唯一索引  
create table ddd(id int primary key auto\_increment , name varchar(32) unique);   
  
create unique index 索引名 on 表名 (列表..);   
  
添加全文索引  
CREATE TABLE articles (  
id INT UNSIGNED AUTO\_INCREMENT NOT NULL PRIMARY KEY,  
title VARCHAR(200),  
body TEXT,  
FULLTEXT (title,body)  
)engine=myisam charset utf8;  
  
ALTER TABLE articles ADD FULLTEXT (title,body);  
  
  
2) 查询索引  
show index from emp\G   
  
  
3) 删除索引  
drop index 索引名称 on 表名;   
4) 修改索引  
先删除，再创建  
  
本讲结束，下一讲将给大家聊聊索引的代价。

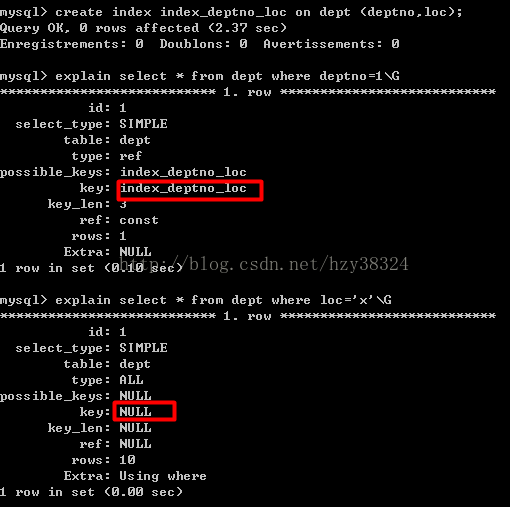
### 4. 索引的代价

前面几讲介绍了索引在提高性能方面的威力以及如何使用索引，但是索引也是有开销的  
  
1) 占用磁盘空间  
如前面所述，索引是有开销的，表现在添加索引后.ibd文件（innodb引擎）或者.myi文件（myisam引擎）会变大。  
  
2) 导致dml操作速度变慢  
添加索引后之所以会快，是因为表依据索引对数据按某种算法（二叉树等）进行排序，所以删除、增加、插入后二叉树要重新排序，导致执行效率降低。  
  
此时要看自己的数据库是dml语句执行的多还是dql语句执行的多  
使用以下语句可以查询  
查询一共执行了多少次select  
show status like ‘com\_select’   
查询一共执行了多少次insert  
show status like ‘com\_insert’   
  
以此类推  
一般来说，dql语句操作比dml语句要多得多！接近9:1  
既然索引有利有弊，那么怎样人为地控制索引的使用呢？  
强制不使用索引  
select xxx from table ignore index(index\_xxx)   
强制使用索引  
select \* from table force index(index\_xxx)   
本讲结束，下一讲来讲一下如何正确地使用索引。

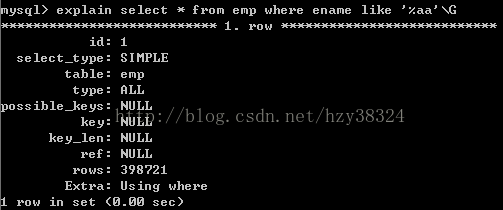
### 5. 什么情况下不要使用索引

既然索引是有代价的，那么就不要在不应该使用索引的情况下去使用它。  
  
1) 数据唯一性差的字段不要使用索引  
比如性别，只有两种可能数据。意味着索引的二叉树级别少，多是平级。这样的二叉树查找无异于全表扫描。  
2) 频繁更新的字段不要使用索引  
比如logincount登录次数，频繁变化导致索引也频繁变化，增大数据库工作量，降低效率。  
3) 字段不在where语句出现时不要添加索引  
只有在where语句出现，mysql才会去使用索引  
4) 数据量少的表不要使用索引  
使用了改善也不大  
  
另外。如果mysql估计使用全表扫描要比使用索引快，则不会使用索引。  
  
本讲结束，下一讲聊聊为什么有时候建了索引也不会被Mysql使用。

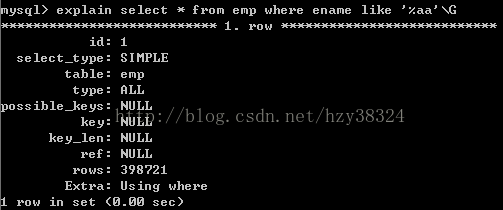
### 6. 添加了索引但不被使用的几种常见可能

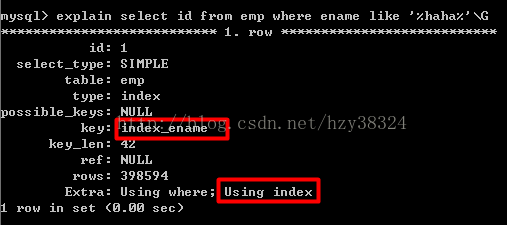
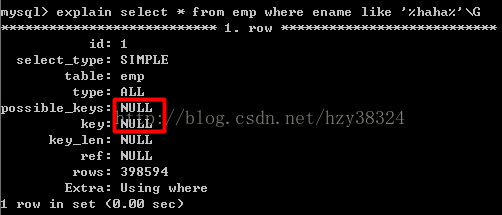
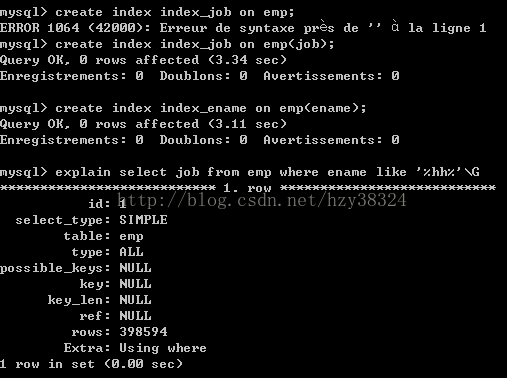
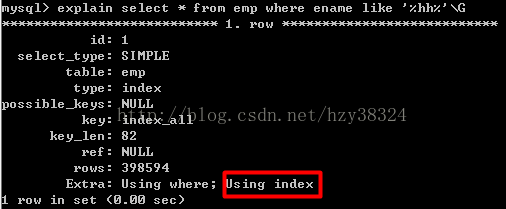
上一讲聊了什么时候不要使用索引，但有时候使用了也不见得会被使用。  
下面是几种添加了索引但不被使用的情况  
  
1) **多列索引查询条件没有使用最左边的字段**  
对于创建的多列索引，如果查询条件没有使用最左边的列，索引就不会被使用。  
多列索引：一个索引对应多个列  
比如  
我创建了这么个多列索引  
  
create index index\_deptno\_loc on dept (deptno,loc);   
  
如果where语句中有deptno则会使用索引，否则不使用  
如下  
  
  
2) **如果条件中有or**  
只要条件中有一个字段没有添加索引，则不会使用索引  
3) **类型不对应**  
比方说，如果列类型是字符串，那一定要在条件中将数据使用引号引用起来。否则不使用索引  
4) **mysql优化器的决定**  
如果mysql估计使用全表扫描要比使用索引快，则不使用索引

最后一点，也是笔者感受最深的一点

5) **like‘%aaa’不会使用到索引**  
只要模糊查询的模糊值在字符串前面，则不会使用索引‘%aaa’和‘\_aaa’都不会！  
如下  
  
应该说这是Mysql给程序员们开的一个玩笑。要是我的表数据量很大，而且又需要使用like’%%’这样的模糊查询来检索时，该怎么办？？  
下一章，笔者将会给大家分享解决这个问题的N种方法！

### 7. 解决like’%str’不使用模糊查询的4种方法

上一讲最后说了，只要模糊查询的模糊值在字符串前面，则不会使用索引，‘%aaa’和‘\_aaa’都不会！  
如下  
  
应该说这是Mysql给程序员们开的一个玩笑。要是我的表数据量很大，而且又需要使用like’%%’这样的模糊查询来检索时，该怎么办？？  
接下来，笔者将会给大家分享解决这个问题的四种方法！  
1) **Select主键**  
只要Select的字段刚好是主键，那么就会使用到索引(只对innodb数据库有效)  
比如下面的  
select idfrom emp where ename like '%haha%'\G   
就使用了索引

  
select \* from emp where ename like '%haha%'\G   
则不使用索引  
  
除了主键，其他字段必须设置为覆盖索引才能使索引生效，不能单独设置索引  
比如下面这种是不会使用索引的  
  
可以采用分步查询的方法，先select主键再利用主键去找其他字段。不过好像比较麻烦！别怕！接下来会讲一种最优方法——覆盖索引法！  
2) **覆盖索引法**  
覆盖索引是一种特殊的多列索引，当多列索引指向一个查询语句中所有的字段时，该多列索引就被称为覆盖索引。  
使用覆盖索引可以解决问题！  
创建覆盖索引  
  
当然，如果你想要select很多字段甚至是select\*，那你可以创建一个多列索引指向所有字段（innodb可以不指向主键）  
  
注意：  
笔者发现，当覆盖索引指向的字段是varchar(380)及380以上的字段时，覆盖索引会失效！  
3) **全文索引法**  
此方法有较大局限。  
全文索引，只对MyISAM引擎有用。主要是针对对文件，文本的检索, 比如文章或者段落,.  
它会把某个数据表的某个数据列出现过的所有单词生成一份清单  
少于3个字符的单词不会被包含在全文索引里，可以通过修改my.cnf修改选项  
ft\_min\_word\_len=3  
但是！

全文索引不完全等同于模糊查询

比如title字段有这么个数据’abcd20088ccaa’,使用模糊查询select \* from articles wheretitle like’%2008%’可以查找到，而使用全文检索select \* from articles where match(title) against(‘2008’);是检索不到的，因为2008不是一个单词！  
4) **使用全文检索引擎工具包**  
采用lucene、Sphinx、solr等专门的全文检索开源工具可以检索某段字符串。  
  
本章结束，下一章将回归数据库的开发源头——设计好一张表。

## 四、 设计一张漂亮的表

上一章讲了如何通过索引来提速我们的查询语句，这一章让我们回到数据库开发的初始阶段，建表。谈谈怎样设计一张漂亮的表。

### **1. 满足三范式**

1NF: 即表的列的具有原子性,不可再分解，即列的信息，不能分解, 只要数据库是关系型数据库(mysql/oracle/db2/informix/sysbase/sqlserver)，就自动的满足1NF  
  
数据库的分类  
关系型数据库: mysql/oracle/db2/informix/sysbase/sql server  
非关系型数据库: 面向对象或者集合  
面向文档数据库: MongoDB  
  
2NF: 表中的记录是唯一的, 就满足2NF, 通常我们设计一个主键来实现。一般情况下，表中都必须设置主键，并且一般不含业务逻辑，可以设置为自增长。  
  
3NF: 即表中不要有冗余数据, 就是说，表的信息，如果能够被推导出来，就不应该单独的设计一个字段来存放.  
有时候为了减少查询次数，可以容忍冗余。

### 2. 字段数目不要过多

一张表的字段过于庞大，会导致查询时的扫描范围加大，即使你并没有查询相应字段。  
所以，如果一张表某个字段，信息量大，但是我们很少查询，则可以考虑把这些字段，单独的放入到一张表中，这种方式称为垂直分割

### 3. 选择合适的字段类型

在创建表的时候要选取最适用的字段属性。  
一般说来，数据库中的表越小，在它上面执行的查询也就会越快。因此，在创建表的时候，为了获得更好的性能，我们可以将表中字段的宽度设得尽可能小  
举个例子  
11位的手机号  
有两种类型可以存储  
Bigint、char(11)。Int类型最大2147483647不可存  
考虑到Bigint占20字节，而char(11)占11\*2=22（gbk）或11\*3=33(utf-8),**所以使用bigInt是最好的**。  
  
如果在创建表之后发现表的字段设置不够合理想修改时怎么修改呢？

### 4. 修改表结构

如果发现表的字段设置不够合理想修改时怎么修改呢，有两种方法。  
传统方法  
  
先使用show createtable 查看原先怎样创建字段的  
然后就可以根据需要修改了  
alter table patent\_data modify mingcheng varchar(500) DEFAULT NULL COMMENT '名称';  
  
  
快速方法（简单粗暴而高效）  
创建一个结构是你所需要的空表，关闭mysql，在文件夹中直接复制新生成的表的frm文件替换掉原来表的的frm文件  
注：  
Frm文件位于mysql安装目录下的 /data/数据库名 目录下  
  
本章结束，下一章将谈谈sql语句的优化方法。  
上一章讲了如何设计一张好的表，一张好的表自然需要好的sql语句去操作它。本章就来聊聊如何优化sql语句。

## 五.如何优化sql语句

### 1. Sql语句优化原则

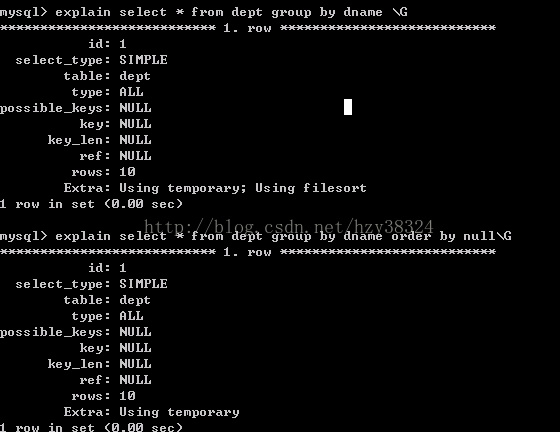
优化需要优化的Query  
定位优化对象性能瓶颈  
从Explain入手  
尽可能在索引中完成排序  
只取自己需要的Column  
尽可能避免复杂的join和子查询

### 2. 优化limit

select \* from test1 order by id limit 99999,10   
原语句虽然使用了id索引，但是相当于从第一行定位到99999行再去扫描后10行，相当于扫描全表  
如果改为  
select \* from test1 where id>=100000 order by id limit 10   
则直接定位到100000查找

### 3. 尽量避免SELECT \*命令

### 4. 不让排序

在使用group by 分组查询时，默认分组后，还会排序，可能会降低速度.  
比如:   
  
在group by后面增加 order by null 就可以防止排序.

### 5. 使用连接来替代子查询

有些情况下，可以使用连接来替代子查询。因为使用join，MySQL不需要在内存中创建临时表。  
[糟糕的效率]  
select \* from sales2 where company\_id not in(select id from company2)   
[简单处理方式]  
select \* from dept, emp where dept.deptno=emp.deptno;   
[左外连接，效率最高]  
select \* from dept left join emp on dept.deptno=emp.deptno;

### 6. 利用LIMIT 1取得唯一行

有时，当你要查询一张表是，你知道自己只需要看一行。在这种情况下，增加一个LIMIT 1会令你的查询更加有效。这样数据库引擎发现只有1后将停止扫描，而不是去扫描整个表或索引

### 7. 使用 EXISTS代替in

EXISTS要远比IN的效率高。里面关系到full table scan和range scan。几乎将所有的IN操作符子查询改写为使用EXISTS的子查询

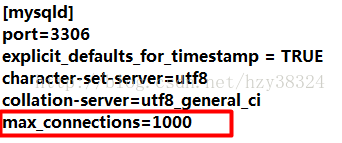
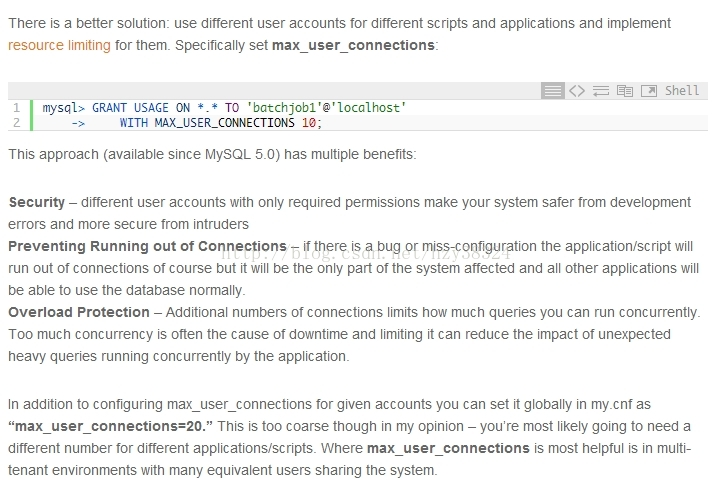
### 8. 不要手贱

没有必要时不要用DISTINCT和ORDER BY  
这些动作可以改在客户端执行，它们增加了额外的开销  
本章结束，下一章聊聊mysql数据库的其他优化方法。

## 六、 MySQL数据库其他优化方法

最后一章，谈谈前面没有涉及的一些优化方法。

### 1. My.ini配置文件

这里只介绍两个最重要的，其他的不过多介绍。  
1) 配置 最大连接数  
默认为100，建议设为1000  
  
更改 2015-04-14 08:00:00  
经csdn网友建议，参考自[Prevent MySQL downtime: Set max\_user\_connections](http://www.percona.com/blog/2014/07/29/prevent-mysql-downtime-set-max_user_connections/) by [Peter Zaitsev](http://www.percona.com/blog/author/admin/)  
在数据库不是只有一个用户在使用，而是允许多个用户并发访问时，如果只设置max\_connections,而没有设置max\_user\_connections，同样会出现连续过多的报错。  
  
  
2) 配置 缓存  
在my.ini的[mysqld]节点开启查询缓存，配置如下  
query\_cache\_size = 268435456  
query\_cache\_type = 1  
query\_cache\_limit = 1048576

### 2. 使用存储过程

存储过程相对sql语句来说的优点是：不需要编译，过程是一个已经编译好的对象

### 3. 水平分割

当一张越来越大时候，即使添加索引还慢的话，我们可以使用分表

### 4. 页面静态化

把每个请求最后生成的静态页面保存到服务器中，以后要是有相同的请求则直接显示静态页面。

整个数据库调优教程到此就该一段落了，算是把自己所学所用所得之经验都写进去了，如果同学们觉得有什么需要补充、有什么地方写的不好的，欢迎交流哈。