## MySQL的引擎

MySQL数据库引擎取决于MySQL在安装的时候是如何被编译的。要添加一个新的引擎，就必须重新编译MYSQL。在缺省情况下，MYSQL支持三个引擎：ISAM、MYISAM和HEAP。另外两种类型INNODB和BERKLEY（BDB），也常常可以使用。如果技术高超，还可以使用MySQL+API自己做一个引擎。下面介绍几种数据库引擎：

## 常用数据库引擎

##### 1. ISAM：

ISAM是一个定义明确且历经时间考验的数据表格管理方法，它在设计之时就考虑到 数据库被查询的次数要远大于更新的次数。因此，ISAM执行读取操作的速度很快，而且不占用大量的内存和存储资源。ISAM的两个主要不足之处在于，它不 支持事务处理，也不能够容错：如果你的硬盘崩溃了，那么数据文件就无法恢复了。如果你正在把ISAM用在关键任务应用程序里，那就必须经常备份你所有的实 时数据，通过其复制特性，MYSQL能够支持这样的备份应用程序。

特点:   
1. 读取操作效率高   
2. 不支持事务外键   
3. 不能容错

##### 2. HEAP：

HEAP允许只驻留在内存里的临时表格。驻留在内存里让HEAP要比ISAM和MYISAM都快，但是它所管理的数据是不稳定的，而且如果在关机之前没有进行保存，那么所有的数据都会丢失。在数据行被删除的时候，HEAP也不会浪费大量的空间。HEAP表格在你需要使用SELECT表达式来选择和操控数据的时候非常有用。要记住，在用完表格之后就删除表格。

特点：  
1. HEAP表是访问数据速度最快的MySQL表   
2. 但如果MySQL或者服务器重新启动，表中数据将会丢失

用法：

如论坛的在线人数统计，这种表的数据应该是无关紧要的,就几个简单的字段,数据也不多,记录数怎么也不会超过1000吧,但是操作是最频繁的(基本用户的每次动作都要更新这个表).

##### 3. MyISAM：

MyIASM是MySQL默认的引擎，是MySQL的ISAM扩展格式但是它同样没有提供对数据库事务的支持，也不支持行级锁和外键，因此当INSERT(插入)或UPDATE(更新)数据时即写操作需要锁定整个表，效率便会低一些。不过MyIASM中存储了表的行数，于是SELECT COUNT(\*) FROM TABLE时只需要直接读取已经保存好的值而不需要进行全表扫描。如果表的读操作远远多于写操作且不需要数据库事务的支持，那么MyIASM也是很好的选择。

特点   
1. 是ISAM的扩展格式   
2. 是Mysql的默认引擎   
3. 提供了表格锁机制来优化多个并发读写操作   
4. 强调了快速读取操作   
5. MyIASM中存储了表的行数，于是SELECT COUNT(\*) FROM TABLE时只需要直接读取已经保存好的值而不需要进行全表扫描。

##### 4. InnoDB：

Innodb引擎提供了对数据库ACID事务的支持，并且实现了SQL标准的四种隔离级别，关于数据库事务与其隔离级别的内容请见数据库事务与其隔离级别这篇文章。该引擎还提供了行级锁和外键约束，它的设计目标是处理大容量数据库[系统](http://www.2cto.com/os/" \t "http://www.2cto.com/database/201503/_blank)，它本身其实就是基于MySQL后台的完整数据库系统，MySQL运行时Innodb会在内存中建立缓冲池，用于缓冲数据和索引。但是该引擎不支持FULLTEXT类型的索引，而且它没有保存表的行数，当SELECT COUNT(\*) FROM TABLE时需要扫描全表。当需要使用数据库事务时，该引擎当然是首选。由于锁的粒度更小，写操作不会锁定全表，所以在并发较高时，使用Innodb引擎会提升效率。但是使用行级锁也不是绝对的，如果在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表。特点:   
1. Innodb引擎提供了对数据库ACID事务的支持   
2. 实现了SQL标准的四种隔离级别   
3. 提供了行级锁和外键约束

4.由于锁的粒度更小，写操作不会锁定全表，所以在并发较高时，使用Innodb引擎会提升效率。  
5. 它没有保存表的行数，当SELECT COUNT(\*) FROM TABLE时需要扫描全表。

如果感觉自己的确技术高超，你还能够使用MySQL+API来创建自己的数据库引擎。这个API为你提供了操作字段、记录、表格、数据库、连接、安全帐号的功能，以及建立诸如MySQL这样DBMS所需要的所有其他无数功能。

### 两种常用数据库引擎对比

### MySQL有多种存储引擎，MyISAM和InnoDB是其中常用的两种。这里介绍关于这两种引擎的一些基本概念（非深入介绍）。 MyISAM是MySQL的默认存储引擎，基于传统的ISAM类型，支持全文搜索，但不是事务安全的，而且不支持外键。每张MyISAM表存放在三个文件中：frm 文件存放表格定义；数据文件是MYD (MYData)；索引文件是MYI (MYIndex)。 InnoDB是事务型引擎，支持回滚、崩溃恢复能力、多版本并发控制、ACID事务，支持行级锁定（InnoDB表的行锁不是绝对的，如果在执行一个SQL语句时MySQL不能确定要扫描的范围，InnoDB表同样会锁全表，如like操作时的SQL语句），以及提供与Oracle类型一致的不加锁读取方式。InnoDB存储它的表和索引在一个表空间中，表空间可以包含数个文件。 ****主要区别：**** •MyISAM是非事务安全型的，而InnoDB是事务安全型的。 •MyISAM锁的粒度是表级，而InnoDB支持行级锁定。 •MyISAM支持全文类型索引，而InnoDB不支持全文索引。 •MyISAM相对简单，所以在效率上要优于InnoDB，小型应用可以考虑使用MyISAM。 •MyISAM表是保存成文件的形式，在跨平台的数据转移中使用MyISAM存储会省去不少的麻烦。 •InnoDB表比MyISAM表更安全，可以在保证数据不会丢失的情况下，切换非事务表到事务表（alter table tablename type=innodb）。 ****应用场景：**** •MyISAM管理非事务表。它提供高速存储和检索，以及全文搜索能力。如果应用中需要执行大量的SELECT查询，那么MyISAM是更好的选择。--select •InnoDB用于事务处理应用程序，具有众多特性，包括ACID事务支持。如果应用中需要执行大量的INSERT或UPDATE操作，则应该使用InnoDB，这样可以提高多用户并发操作的性能。--insert，update ****常用命令： （1）查看表的存储类型（三种）：**** •show create table tablename •show table status from  dbname  where name=tablename •mysqlshow  -u user -p password --status dbname tablename ****（2）修改表的存储引擎：**** •alter table tablename type=InnoDB ****（3）启动mysql数据库的命令行中添加以下参数使新发布的表都默认使用事务：**** •--default-table-type=InnoDB ****（4）临时改变默认表类型：**** •set table\_type=InnoDB •show variables like 'table\_type'

关闭mysql服务

net stop mysql

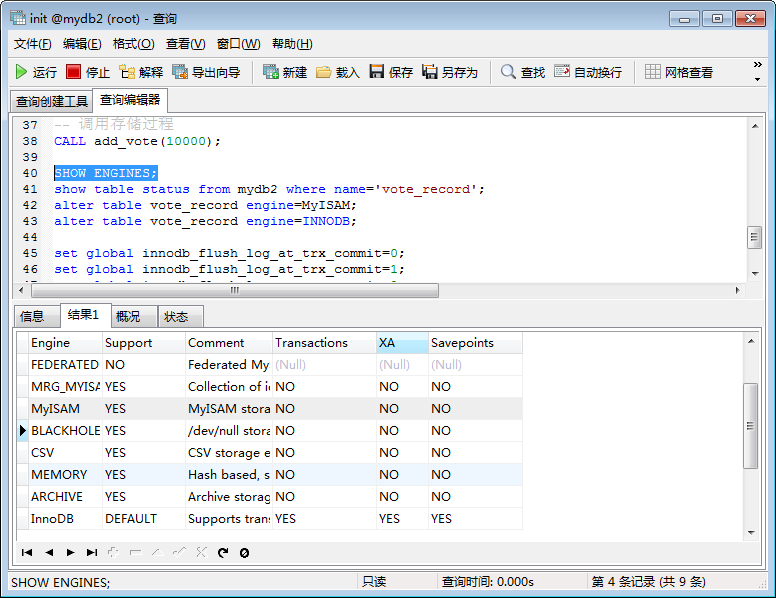
启动mysql服务

net start mysql

查看引擎

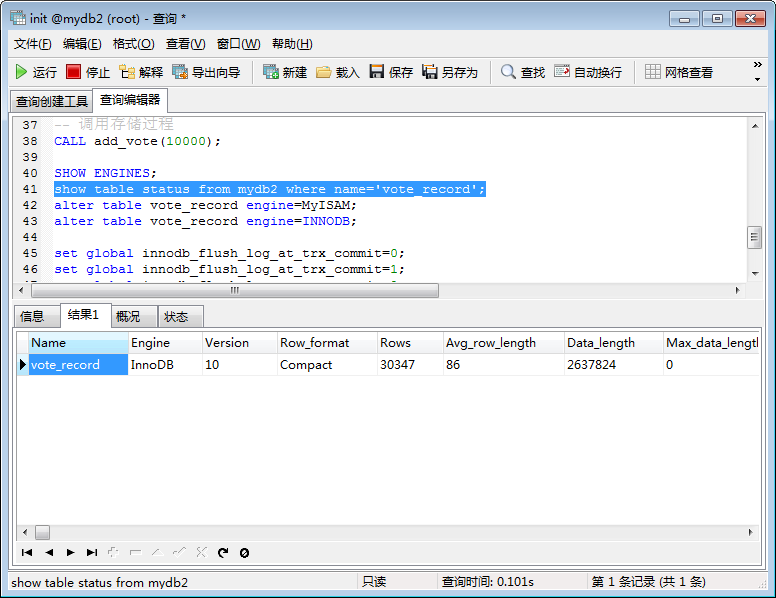
SHOW ENGINES;

看到现在默认的引擎是InnoDB



查看表所使用的引擎

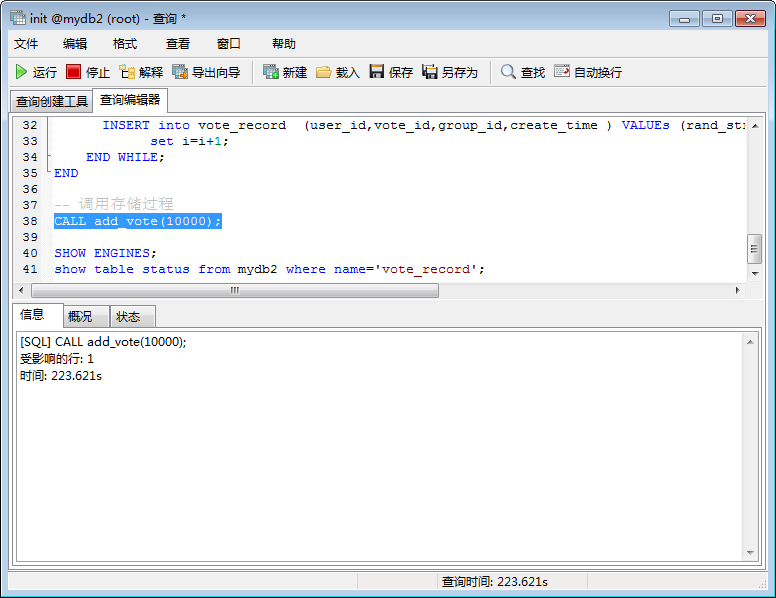
show table status from mydb2 where name='vote\_record';



一个表只能用一种引擎,同一个库中的不同表允许使用不同的引擎

可以让数据库灵活的适应各种场景

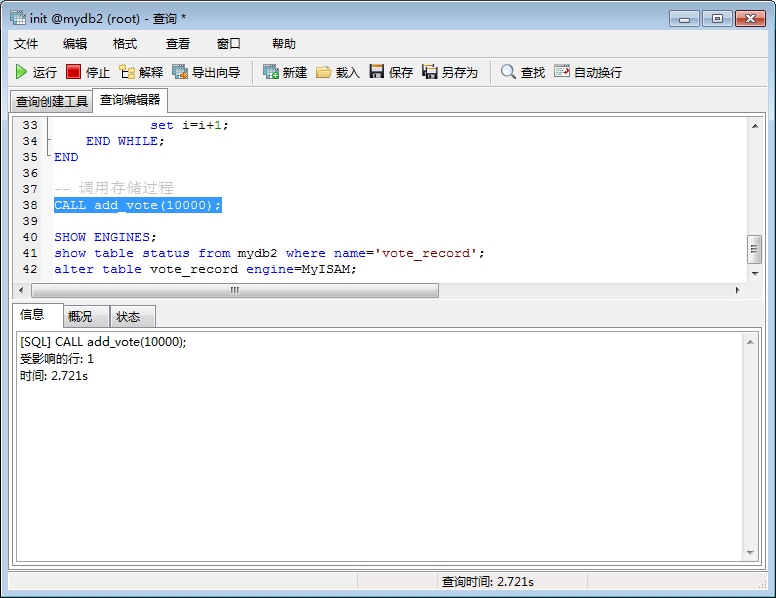
InnoDB插入1W条数据



修改引擎:

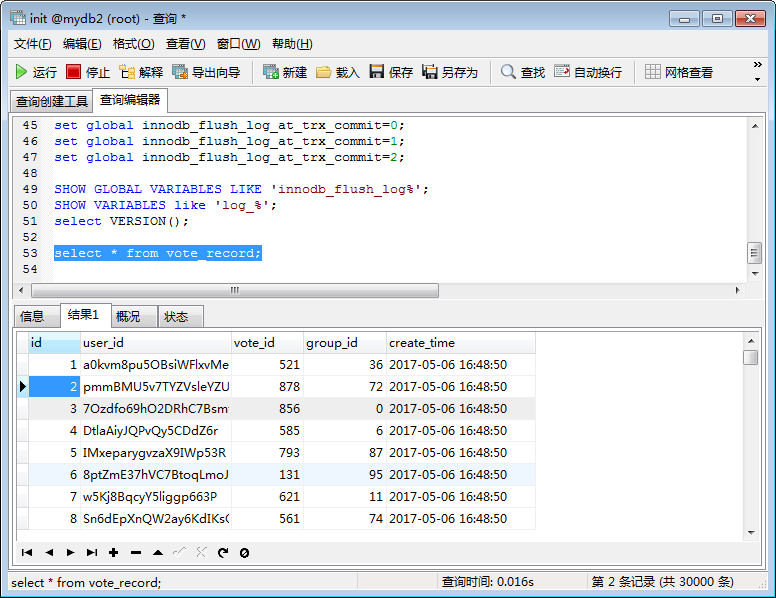
alter table vote\_record engine=MyISAM;

MyISAM插入1W条数据



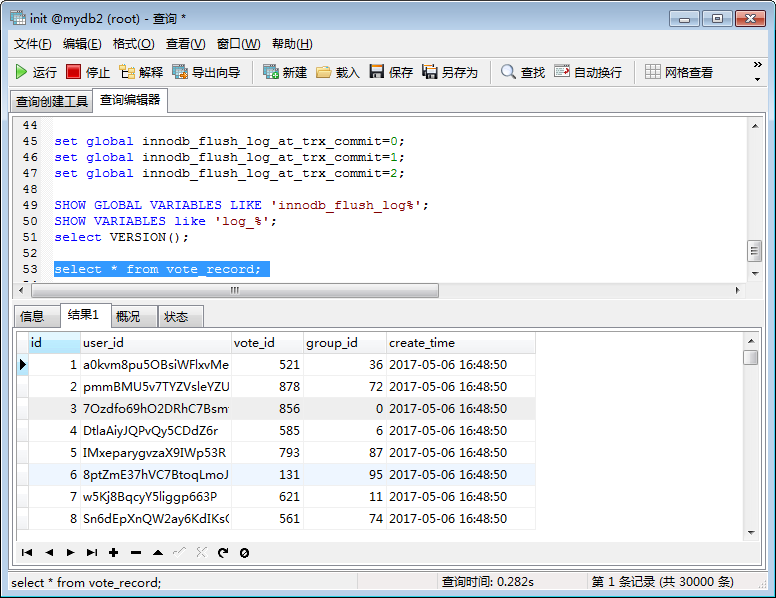
MyISAM

Select \* from vote\_record



InnoDB

Select \* from vote\_record



问题:区别上来说MyISAM的应用场景应该是大量读写操作，且不需要支持事务的情况下，查找能力强是应该的。

看之前介绍的时候InnoDB的写操作应该是比MyISAM更高效的，即使现在没有高并发，也不应该有这么大的差距，但是为什么现在在插入操作的时候条目的效率有两个数量级的差距？

既然现在不存在高并发的问题，那么问题就应该是出在其他的差别方面。

我们知道InnoDB是支持事务的，而MyISAM是不支持事务的，那有没有可能是因为事务的关系呢？

我们都明白事务是什么但是事务的底层是如何操作的呢？

首先来理解一下事务的流程:

MySQL 本身不提供事务支持，而是开放了存储引擎接口，由具体的存储引擎来实现，具体来说支持 MySQL 事务的存储引擎就是 InnoDB。

存储引擎实现事务的通用方式是基于 redo log 和 undo log 日志。

**Undo Log**原理：

为了满足事务的原子性，在操作任何数据之前，首先将数据备份到一个地方（这个存储数据备份的地方称为Undo Log）。然后进行数据的修改。如果出现了错误或者用户执行了ROLLBACK语句，系统可以利用Undo Log中的备份将数据恢复到事务开始之前的状态。

**Redo Log**原理：  
 和Undo Log相反，Redo Log记录的是新数据的备份。在事务提交前，只要将Redo Log持久化即可，不需要将数据持久化。当系统崩溃时，虽然数据没有持久化，但是Redo Log已经持久化。系统可以根据Redo Log的内容，将所有数据恢复到最新的状态。

总结：简单来说，redo log 记录事务修改后的数据, undo log 记录事务前的原始数据。

日志在内存里也是有缓存的，这里将其叫做log buffer

数据库本身不支持事务，老师上课讲事务知识点的时候，提到一个概念，数据库默认支持事务的，不过只是针对一条sql语句。所以现在更加深入理解是，数据库采用Innodb这个支持事务引擎的时候，引擎在做sql操作的时候会为每一条sql语句增加事务。所以每条sql的事务操作，就是使InnoDB引擎的添加要比MyISAM慢的主要原因。

如何解决这个问题呢？

提高数据的写入速度

**InnoDB配置**

innodb\_flush\_log\_at\_trx\_commit 参数定制化commit返回时日志是否刷新到磁盘

sync\_binlog是MySQL innodb将二进制日志文件刷新到磁盘上。

innodb事务日志redo,binlog逻辑过程如下:  
1.事务写入redo log buffer中；  
2.将log buffer刷新到redo log中，不过会先写一个TX PREPARE标记；  
3.写binlog  
4.在redo log中写入TX COMMIT标记；  
5.将写binlog成功的标记写入redo log。

二进制日志文件:binary log

记录所有更改数据的语句。

1.innodb\_flush\_log\_at\_trx\_commit参数：

My.ini

# If set to 1, InnoDB will flush (fsync) the transaction logs to the

# disk at each commit, which offers full ACID behavior. If you are

# willing to compromise this safety, and you are running small

# transactions, you may set this to 0 or 2 to reduce disk I/O to the

# logs. Value 0 means that the log is only written to the log file and

# the log file flushed to disk approximately once per second. Value 2

# means the log is written to the log file at each commit, but the log

# file is only flushed to disk approximately once per second.

innodb\_flush\_log\_at\_trx\_commit=1

============================================================================

0

 每隔一秒，把事务日志缓存区的数据写到日志文件中，以及把日志文件的数据刷新到磁盘上； log buffer会每秒写入到日志文件并刷写（flush）到磁盘。但每次事务提交不会有任何影响，也就是 log buffer 的刷写操作和事务提交操作没有关系。在这种情况下，MySQL性能最好，但如果 mysqld 进程崩溃，通常会导致最后 1s 的日志丢失。

意味着数据的写入，跟事务的提交是分开的，数据该怎么写入就怎么写入，而事务自己提交自己的，这两个互不影响

1

当设为默认值1的时候，每次提交事务的时候，都会将log buffer刷写到日志。

这意味着事务提交和数据的写入是同步的，每次提交数据，就执行写入磁盘的操作。

2

如果设为2,每次提交事务都会写日志，但并不会执行刷的操作。每秒定时会刷到日志文件。要注意的是，并不能保证100%每秒一定都会刷到磁盘，这要取决于进程的调度。

默认值1是为了保证完整的[ACID](http://en.wikipedia.org/wiki/ACID" \t "http://blog.csdn.net/liqfyiyi/article/details/_blank)。当然，你可以将这个配置项设为1以外的值来换取更高的性能，但是在系统崩溃的时候，你将会丢失1秒的数据。设为0的话，mysqld进程崩溃的时候，就会丢失最后1秒的事务。设为2,只有在**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/liqfyiyi/article/details/_blank)**崩溃或者断电的时候才会丢失最后1秒的数据。InnoDB在做恢复的时候会忽略这个值。

总结下来：为1的时候，提交一次，写入一次事务日志到磁盘，写入数据到磁盘。0的时候，写入数据，事务也写入日志文件，但是事务的提交跟数据的写入没有关系，异步操作。2的时候，每次提交事务都会写入缓存里，每一秒定时刷到磁盘里这里是同步的操作，但是不是一条条的写入磁盘，而是先写入缓存，最后每隔1秒钟写入一次磁盘。

实验：

修改innodb\_flush\_log\_at\_trx\_commit值

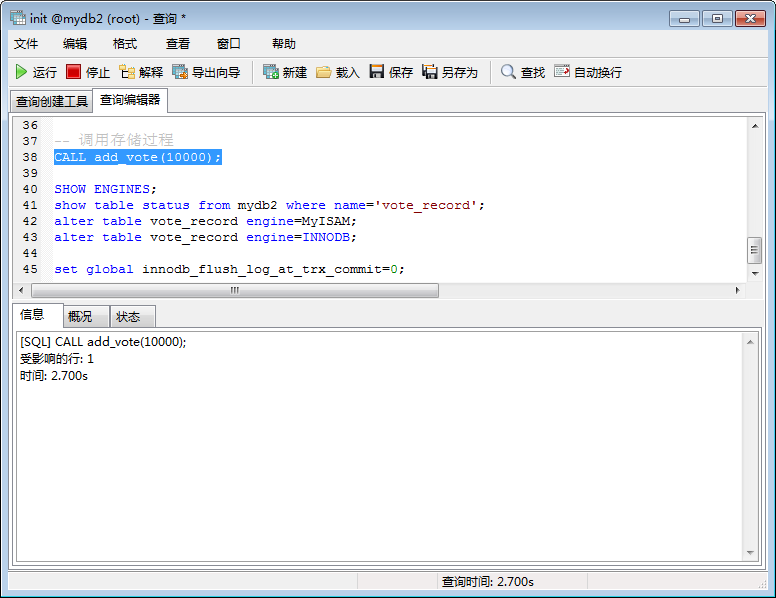
set global innodb\_flush\_log\_at\_trx\_commit=0;

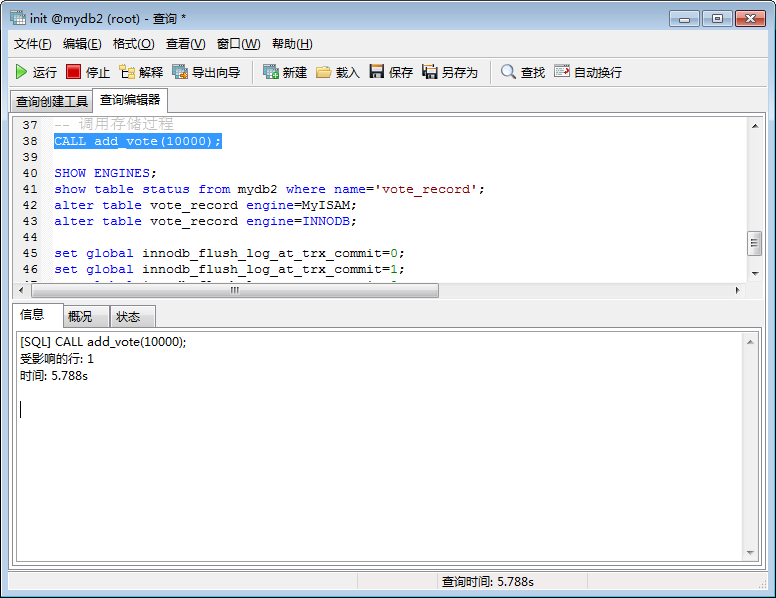
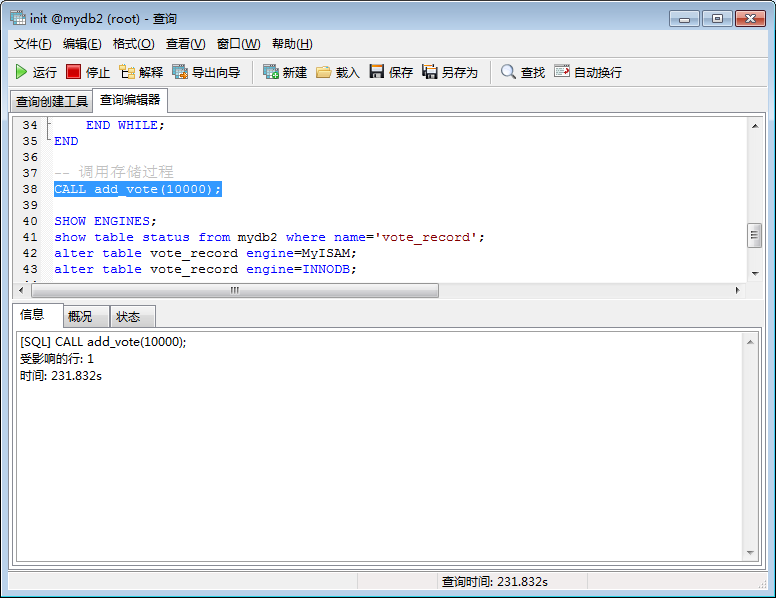
set global innodb\_flush\_log\_at\_trx\_commit=1;

set global innodb\_flush\_log\_at\_trx\_commit=2;

查看innodb\_flush\_log\_at\_trx\_commit值

SHOW GLOBAL VARIABLES LIKE 'innodb\_flush\_log%';

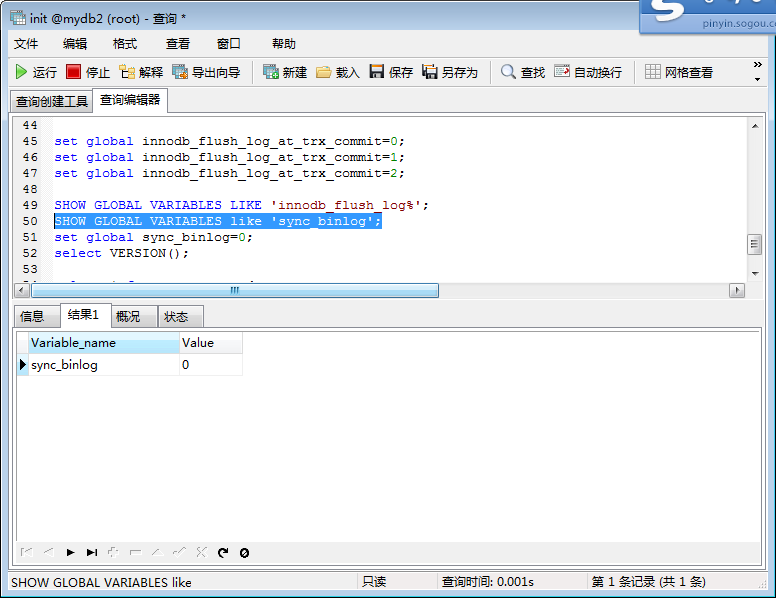




2.sync\_binlog

查看

SHOW GLOBAL VARIABLES like 'sync\_binlog';



设置

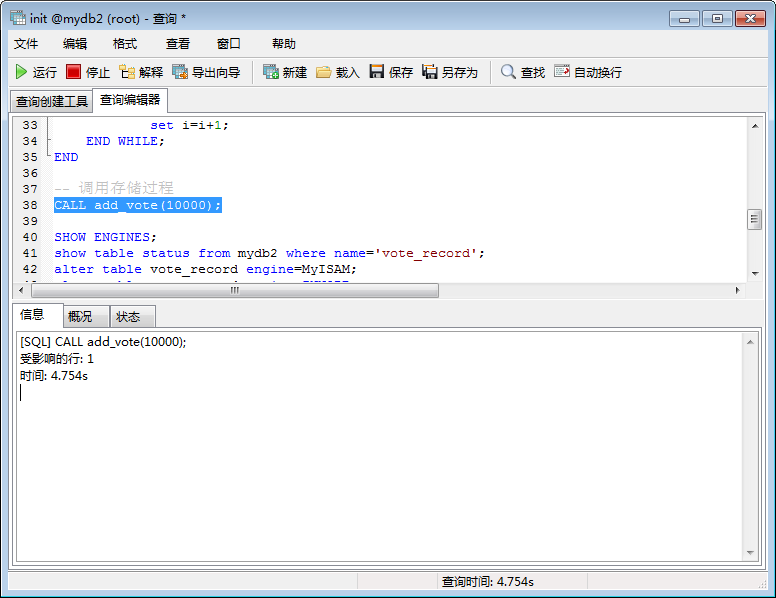
set global sync\_binlog=0;

我们可以看到sync\_binlog的默认值为0，这里没找到sync\_binlog在哪个文件配置的，直接看下网上介绍的sync\_binlog参数作用

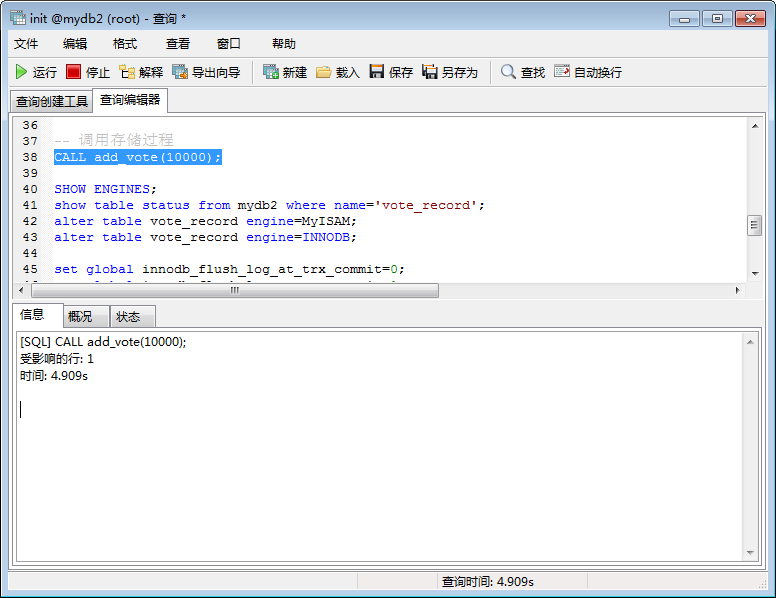
sync\_binlog =  N：   
N>0    每向二进制日志文件写入N条SQL或N个事务后，则把二进制日志文件的数据刷新到磁盘上；   
N=0    不主动刷新二进制日志文件的数据到磁盘上，而是由操作系统决定；

实验

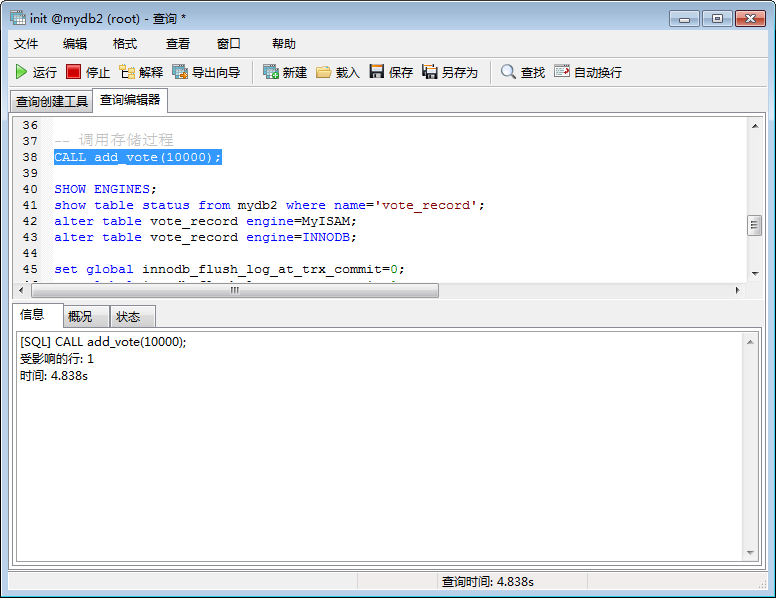
N=2,0



N=2,1



N=2,100



**推荐配置组合：**  
  
N=1,1  — 适合数据安全性要求非常高，而且磁盘IO写能力足够支持业务，比如充值消费系统；   
N=1,0  — 适合数据安全性要求高，磁盘IO写能力支持业务不富余，允许备库落后或无复制；   
N=2,0或2,m(0<m<100)  — 适合数据安全性有要求，允许丢失一点事务日志，复制架构的延迟也能接受；   
N=0,0  — 磁盘IO写能力有限，无复制或允许复制延迟稍微长点能接受，例如：日志性登记业务；   
  当两个参数设置为双1的时候，写入性能最差，sync\_binlog=N (N>1 ) innodb\_flush\_log\_at\_trx\_commit=2 时，(在当前模式下)MySQL的写操作才能达到最高性能。

**数据安全性**  
当innodb\_flush\_log\_at\_trx\_commit和sync\_binlog  都为1时是最安全的，在mysqld 服务崩溃或者服务器主机crash的情况下，binary log 只有可能丢失最多一个语句或者一个事务。但是鱼与熊掌不可兼得，都为1会导致频繁的IO操作，因此该模式也是最慢的一种方式。   
当innodb\_flush\_log\_at\_trx\_commit设置为0，mysqld进程的崩溃会导致上一秒钟所有事务数据的丢失。   
当innodb\_flush\_log\_at\_trx\_commit设置为2，只有在操作系统崩溃或者系统掉电的情况下，上一秒钟所有事务数据才可能丢失。

双1适合数据安全性要求非常高，而且磁盘IO写能力足够支持业务，比如订单,交易,充值,支付消费系统。双1模式下，当磁盘IO无法满足业务需求时，推荐的做法是innodb\_flush\_log\_at\_trx\_commit=2 ，sync\_binlog=N (N为500 或1000) 且使用带蓄电池后备电源的缓存cache，防止系统断电异常。

1. innodb\_buffer\_pool\_size

这个是Innodb最重要的参数，和MyISAM的key\_buffer\_size有相似之处，但也是有差别的。   
这个参数主要缓存innodb表的索引，数据，插入数据时的缓冲。

该参数分配内存的原则：  
这个参数默认分配只有8M，可以说是非常小的一个值。  
如果是一个专用DB服务器，那么他可以占到内存的70%-80%。  
这个参数不能动态更改，所以分配需多考虑。分配过大，会使Swap占用过多，致使Mysql的查询特慢。  
如果你的数据比较小，那么可分配是你的数据大小+10%左右做为这个参数的值。  
例如：数据大小为50M,那么给这个值分配innodb\_buffer\_pool\_size＝64M

1. Innodb\_additional\_mem\_pool\_size=16M

用来存放Innodb的内部目录这个值不用分配太大，系统可以自动调。不用设置太高。通常比较大数据设置16Ｍ够用了，如果表比较多，可以适当的增大。如果这个值自动增加，会在error log有中显示的。  
5.innodb\_log\_file\_size=256M

在日志组中每个日志文件的大小，一般是innodb\_buffer\_pool\_size的25%，官方推荐是innodb\_buffer\_pool\_size的40-50%。一般控制在几个ＬＯＧ文件相加大小在２Ｇ以内为佳。具体情况还需要看你的事务大小，数据大小为依据。说明：这个值分配的大小和数据库的写入速度，事务大小，异常重启后的恢复有很大的关系。  
6.Innodb\_log\_files\_in\_group=2

指定你有几个日志组。分配原则：一般我们可以用２－３个日值组。默认为两个。  
7.innodb\_log\_buffer\_size＝3M

事务在内存中的缓冲。分配原则：控制在2-8M.这个值不用太多的。他里面的内存一般一秒钟写到磁盘一次。具体写入方式和你的事务提交方式有关。在oracle等数据库了解这个，一般最大指定为３Ｍ比较合适。

<http://blog.csdn.net/thundermeng/article/details/50448614>

<http://www.cnblogs.com/kevingrace/p/6133818.html>

<http://www.jb51.net/article/38178.htm>

http://www.jb51.net/article/42949.htm