深入浅出 MySQL 数据库开发、优化与管理维护

第一篇 开发篇

帮助的使用

在不同的 mysql 版本中，很多特性和语法有可能是不一样的，我们怎么样才能知道当前版本的语法是什么样呢？最好的办法是学会使用 mysql 的帮助。方法很简单：

按照层次看帮助

如果不知道帮助能够提供些什么，可以一层一层往下看：

命令如下：

mysql> ? contents

You asked for help about help category: "Contents"

For more information, type 'help <item>', where <item> is one of the following

categories:

Account Management

Administration

Data Definition

Data Manipulation

Data Types

Functions

Functions and Modifiers for Use with GROUP BY

Geographic Features

Language Structure

Plugins

Storage Engines

Stored Routines

Table Maintenance

Transactions

Triggers

对于列出的分类，可以进行看自己感兴趣的部分，例如：

mysql> ? data types

You asked for help about help category: "Data Types"

For more information, type 'help <item>', where <item> is one of the following

topics:

AUTO_INCREMENT

BIGINT

BINARY

BIT

BLOB

BLOB DATA TYPE

BOOLEAN

。。。。。。

对于列出的具体数据类型，可以进一步看详细情况：

mysql> ? int

Name: 'INT'

Description:

INT[(M)] [UNSIGNED] [ZEROFILL]

A normal-size integer. The signed range is -2147483648 to 2147483647.

The unsigned range is 0 to 4294967295.

快速查阅帮助

实际当中，如果我们需要快速查阅某项语法时，可以使用关键字进行快速查询。例如，我想知道 show 命令都能看些什么东西，可以用如下命令：

mysql> ? show

Name: 'SHOW'

Description:

SHOW has many forms that provide information about databases, tables,

columns, or status information about the server. This section describes

those following:

SHOW AUTHORS

SHOW CHARACTER SET [LIKE 'pattern']

SHOW COLLATION [LIKE 'pattern']

SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']

SHOW CONTRIBUTORS

SHOW CREATE DATABASE db_name

SHOW CREATE EVENT event_name

SHOW CREATE FUNCTION funcname

。。。。。。

我想知道 create table 的语法，可以命令如下：

mysql> ? create table

Name: 'CREATE TABLE'

Description:

Syntax:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name

(create_definition,...)

[table_option ...]

[partition_options]

Or:

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name

[(create_definition,...)]

[table_option ...]

[partition_options]

select_statement

、 。。。。。。

表类型（存储引擎）的选择

Mysql 存储引擎概述

mysql支持多种存储引擎，在处理不同类型的应用时，可以通过选择使用不同的存储引擎提高应用的效率，或者提供灵活的存储。

mysql的存储引擎包括：MyISAM、 InnoDB、BDB、MEMORY、MERGE、EXAMPLE、NDB Cluster、ARCHIVE、CSV、BLACKHOLE、FEDERATED等，其中InnoDB和BDB提供事务安全表，其他存储引擎都是非事务安全表。

各种存储引擎的特性

下面我们重点介绍几种常用的存储引擎并对比各个存储引擎之间的区别和推荐使用方式。

特点	Myisam	BDB	Memory	InnoDB	Archive
存储限制	没有	没有	有	64TB	没有
事务安全		支持		支持	
锁机制	表锁	页锁	表锁	行锁	行锁
B 树索引	支持	支持	支持	支持	
哈希索引			支持	支持	
全文索引	支持				
集群索引				支持	
数据缓存			支持	支持	
索引缓存	支持		支持	支持	
数据可压缩	支持				支持
空间使用	低	低	N/A	高	非常低
内存使用	低	低	中等	高	低
批量插入的速度	高	高	高	低	非常高
支持外键				支持	

最常使用的 2 种存储引擎:

1. Myisam是Mysql的默认存储引擎，当create创建新表时，未指定新表的存储引擎时，默认使用Myisam。

   每个MyISAM在磁盘上存储成三个文件。文件名都和表名相同，扩展名分别是.frm（存储表定义）、.MYD (MYData，存储数据)、.MYI (MYIndex，存储索引)。数据文件和索引文件可以放置在不同的目录，平均分布io，获得更快的速度。
2. InnoDB 存储引擎提供了具有提交、回滚和崩溃恢复能力的事务安全。但是对比 Myisam 的存储引擎，InnoDB 写的处理效率差一些并且会占用更多的磁盘空间以保留数据和索引。

如何选择合适的存储引擎

选择标准: 根据应用特点选择合适的存储引擎,对于复杂的应用系统可以根据实际情况选择多种存储引擎进行组合.

下面是常用存储引擎的适用环境:

1. MyISAM：默认的 MySQL 插件式存储引擎，它是在 Web、数据仓储和其他应用环境下最常使用的存储引擎之一
2. InnoDB：用于事务处理应用程序，具有众多特性，包括 ACID 事务支持。
3. Memory：将所有数据保存在 RAM 中，在需要快速查找引用和其他类似数据的环境下，可提供极快的访问。
4. Merge：允许 MySQL DBA 或开发人员将一系列等同的 MyISAM 表以逻辑方式组合在一起，并作为 1 个对象引用它们。对于诸如数据仓储等 VLDB 环境十分适合。

选择合适的数据类型

选择数据类型的基本原则

前提: 使用适合存储引擎。

选择原则: 根据选定的*存储引擎*,确定如何选择合适的数据类型

下面的选择方法按存储引擎分类:

1. MyISAM 数据存储引擎和数据列

   MyISAM数据表，最好使用固定长度的数据列代替可变长度的数据列。

2. MEMORY存储引擎和数据列

   MEMORY数据表目前都使用固定长度的数据行存储，因此无论使用CHAR或VARCHAR列都没有关系。两者都是作为CHAR类型处理的。

3. InnoDB 存储引擎和数据列

   建议使用 VARCHAR类型

   对于InnoDB数据表，内部的行存储格式没有区分固定长度和可变长度列（所有数据行都使用指向数据列值的头指针），因此在本质上，使用固定长度的 CHAR列不一定比使用可变长度VARCHAR列简单。因而,主要的性能因素是数据行使用的存储总量。由于CHAR平均占用的空间多于VARCHAR,因 此使用VARCHAR来最小化需要处理的数据行的存储总量和磁盘I/O是比较好的。

固定长度数据列与可变长度的数据列

char 与 varchar

- CHAR 和 VARCHAR 类型类似，但它们保存和检索的方式不同。它们的最大长度和是否尾部空格被保留等方面也不同。在存储或检索过程中不进行大小写转换。

下面的表显示了将各种字符串值保存到 CHAR(4)和 VARCHAR(4)列后的结果，说明了 CHAR 和 VARCHAR 之间的差别：

值	CHAR(4)	存储需求	VARCHAR(4)	存储需求
''	' '	4 个字节	''	1 个字节
'ab'	'ab '	4 个字节	'ab '	3 个字节
'abcd'	'abcd'	4 个字节	'abcd'	5 个字节
'abcdefgh'	'abcd'	4 个字节	'abcd'	5 个字节

请注意上表中最后一行的值只适用*不使用严格模式*时；如果 MySQL 运行在严格模式，超过列长度不的值*不保存*，并且会出现错误。

从 CHAR(4)和 VARCHAR(4)列检索的值并不总是相同，因为检索时从 CHAR 列删除了尾部的空格。通过下面的例子说明该差别：

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO vc VALUES ('ab  ', 'ab  ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT(v, '+'), CONCAT(c, '+') FROM vc;
+---------------+---------------+
| CONCAT(v, '+') | CONCAT(c, '+') |
+---------------+---------------+
```

```
| ab  +           | ab+             |
+---------------+---------------+
1 row in set (0.00 sec)
```

text 和 blob

在使用 text 和 blob 字段类型时要注意以下几点,以便更好的发挥数据库的性能.

1. **BLOB和TEXT值也会引起自己的一些问题，特别是执行了大量的删除或更新操作的时候。**删除这种值会在数据表中留下很大的"空洞",以后填入这些"空洞"的记录可能长度不同,为了提高性能,建议定期使用 OPTIMIZE TABLE 功能对这类表进行碎片整理.
2. **使用合成的（synthetic）索引。**合成的索引列在某些时候是有用的。一种办法是根据其它的列的内容建立一个散列值，并把这个值存储在单独的数据列中。接下来你就可以通过检索散列值找到数据行了。但是,我们要注意这种技术只能用于精确匹配的查询(散列值对于类似<或>=等范围搜索操作符 是没有用处的)。我们可以使用MD5()函数生成散列值，也可以使用SHA1()或CRC32()，或者使用自己的应用程序逻辑来计算散列值。请记住数值型散列值可以很高效率地存储。同样，如果散列算法生成的字符串带有尾部空格，就不要把它们存储在CHAR或VARCHAR列中，它们会受到尾部空格去除的影响。

合成的散列索引对于那些BLOB或TEXT数据列特别有用。用散列标识符值查找的速度比搜索BLOB列本身的速度快很多。

3. **在不必要的时候避免检索大型的BLOB或TEXT值。**例如，SELECT *查询就不是很好的想法，除非你能够确定作为约束条件的WHERE子句只会找到所需要的数据行。否则，你可能毫无目的地在网络上传输大量的值。这也是 BLOB或TEXT标识符信息存储在合成的索引列中对我们有所帮助的例子。你可以搜索索引列，决定那些需要的数据行，然后从合格的数据行中检索BLOB或 TEXT值。
4. **把BLOB或TEXT列分离到单独的表中。**在某些环境中，如果把这些数据列移动到第二张数据表中，可以让你把原数据表中 的数据列转换为固定长度的数据行格式，那么它就是有意义的。这会减少主表中的碎片，使你得到固定长度数据行的性能优势。它还使你在主数据表上运行 SELECT *查询的时候不会通过网络传输大量的BLOB或TEXT值。

浮点数与定点数

为了能够引起大家的重视，在介绍浮点数与定点数以前先让大家看一个例子：

```
mysql> CREATE TABLE test (c1 float(10,2),c2 decimal(10,2));
Query OK, 0 rows affected (0.29 sec)
mysql> insert into test values(131072.32,131072.32);
Query OK, 1 row affected (0.07 sec)
mysql> select * from test;
+-----------+-----------+
| c1        | c2        |
+-----------+-----------+
| 131072.31 | 131072.32 |
+-----------+-----------+
1 row in set (0.00 sec)
```

从上面的例子中我们看到 c1 列的值由 131072.32 变成了 131072.31，这就是浮点数的不精确性造成的。

在 mysql 中 float、double（或 real）是浮点数，decimal（或 numberic）是定点数。

浮点数相对于定点数的优点是在长度一定的情况下，浮点数能够表示更大的数据范围；它的缺点是会引起精度问题。

在今后关于浮点数和定点数的应用中，大家要记住以下几点：

1、浮点数存在误差问题；

2、对货币等对精度敏感的数据，应该用定点数表示或存储；

3、编程中，如果用到浮点数，要特别注意误差问题，并尽量避免做浮点数比较；

4、要注意浮点数中一些特殊值的处理。

字符集

字符集概述

字符集是一套符号和编码的规则，不论是在 oracle 数据库还是在 mysql 数据库，都

存在字符集的选择问题，而且如果在数据库创建阶段没有正确选择字符集，那么可能在后期需要更换字符集，而字符集的更换是代价比较高的操作，也存在一定的风险，所以，我们推荐在应用开始阶段，就按照需求正确的选择合适的字符集，避免后期不必要的调整。

Mysql 支持的字符集简介

mysql服务器可以支持多种字符集（可以用show character set命令查看所有mysql支持的字符集），在同一台服务器、同一个数据库、甚至同一个表的不同字段都可以指定使用不同的字符集，相比oracle等其他数据库管理系统，在同一个数据库只能使用相同的字符集，mysql明显存在更大的灵活性。

mysql的字符集包括字符集（CHARACTER）和校对规则（COLLATION）两个概念。字符集是用来定义mysql存储字符串的方式，校对规则则是定义了比较字符串的方式。字符集和校对规则是一对多的关系，MySQL支持30多种字符集的70多种校对规则。

每个字符集至少对应一个校对规则。可以用SHOW COLLATION LIKE 'utf8%';命令查看相关字符集的校对规则。

Unicode 简述

Unicode 是一种编码规范。我们在这里简述一下 Unicode 编码产生的历史。

先从 ASCII 码说起，ASCII 码也是一种编码规范，只不过 ASCII 码只能最多表示 256 个字符，是针对英文产生的，而面对中文、阿拉伯文之类的复杂文字，256 个字符显然是不够用的。于是各个国家或组织都相继制定了符合自己语言文字的标准，比如 gb2312、big5 等等。但是这种各自制定自己的标准的做法显然是有很多弊端的，于是 Unicode 编码规范应运而生。

Unicode 也是一种字符编码方法，不过它是由国际组织设计，可以容纳全世界所有语言文字的编码方案。Unicode 的学名是"Universal Multiple-Octet Coded Character Set"，简称为 UCS。UCS 可以看作是"Unicode Character Set"的缩写。

Unicode 有两套标准 UCS-2 和 UCS-4，前者用 2 个字节表示一个字符，后者用 4 个字节表示一个字符。以目前常用的 UCS-2 为例，它可以表示的字符数为 2^16=65535，基本上可以容纳所有的欧美字符和绝大多数亚洲字符。

怎样选择合适的字符集

我们建议在能够完全满足应用的前提下，尽量使用小的字符集。因为更小的字符集意味着能够节省空间、减少网络传输字节数，同时由于存储空间的较小间接的提高了系统的性能。

有很多字符集可以保存汉字，比如 utf8、gb2312、gbk、latin1 等等，但是常用的是 gb2312 和 gbk。因为 gb2312 字库比 gbk 字库小，有些偏僻字（例如：洺）不能保存，因此在选择字符集的时候一定要权衡这些偏僻字在应用出现的几率以及造成的影响，不能做出肯定答复的话最好选用 gbk。

Mysql 字符集的设置

mysql 的字符集和校对规则有 4 个级别的默认设置：服务器级、数据库级、表级和字段级。分别在不同的地方设置，作用也不相同。

服务器字符集和校对，在 mysql 服务启动的时候确定。

可以在 my.cnf 中设置：

```
[mysqld]
default-character-set=utf8
```

或者在启动选项中指定：

```
mysqld --default-character-set=utf8
```

或者在编译的时候指定：

```
./configure --with-charset=utf8
```

如果没有特别的指定服务器字符集，默认使用 latin1 作为服务器字符集。上面三种设置的方式都只指定了字符集，没有指定校对规则，这样是使用该字符集默认的校对规则，如果要使用该字符集的非默认校对规则，则需要在指定字符集的同时指定校对规则。

可以用 show variables like 'character_set_server';命令查询当前服务器的字符集和校对规则。

索引的设计和使用

Mysql 索引概述

所有 MySQL 列类型可以被索引。对相关列使用索引是提高 SELECT 操作性能的最佳途径。根据存储引擎定义每个表的最大索引数和最大索引长度。所有存储引擎支持每个表至少 16 个索引，总索引长度至少为 256 字节。大多数存储引擎有更高的限制。

在 MySQL 5.1 中，对于 MyISAM 和 InnoDB 表，前缀可以达到 1000 字节长。请注意前缀的限制应以字节为单位进行测量，而 CREATE TABLE 语句中的前缀长度解释为字符数。当为使用多字节字符集的列指定前缀长度时一定要加以考虑。

还可以创建 FULLTEXT 索引。该索引可以用于全文搜索。只有 MyISAM 存储引擎支持 FULLTEXT 索引，并且只为 CHAR、VARCHAR 和 TEXT 列。索引总是对整个列进行，不支持局部(前缀)索引。也可以为空间列类型创建索引。只有 MyISAM 存储引擎支持空间类型。空间索引使用 R-树。默认情况 MEMORY(HEAP)存储引擎使用 hash 索引，但也支持 B-树索引。

设计索引的原则

1. 搜索的索引列，不一定是所要选择的列。换句话说，最适合索引的列是出现在 WHERE 子句中的列，或连接子句中指定的列，而不是出现在 SELECT 关键字后的选择列表中的列。
2. 使用惟一索引。考虑某列中值的分布。对于惟一值的列，索引的效果最好，而具有多个重复值的列，其索引效果最差。例如，存放年龄的列具有不同值，很容易区分 各行。而用来记录性别的列，只含有“ M”和“F”，则对此列进行索引没有多大用处（不管搜索哪个值，都会得出大约一半的行）
3. 使用短索引。如果对串列进行索引，应该指定一个前缀长度，只要有可能就应该这样做。例如，如果有一个 CHAR(200) 列，如果在前 10 个或 20 个字符内，多数值是惟一的，那么就不要对整个列进行索引。对前 10 个或 20 个字符进行索引能够节省大量索引空间，也可能会使查询更快。较小的索引涉及的磁盘 I/O 较少，较短的值比较起来更快。更为重要的是，对于较短的键值，索引高速缓存中的块能容纳更多的键值，因此，MySQL 也可以在内存中容纳更多的值。这增加 了找到行而不用读取索引中较多块的可能性。（当然，应该利用一些常识。如仅用列值的第一个字符进行索引是不可能有多大好处的，

因为这个索引中不会有许多不 同的值。）

4. 利用最左前缀。在创建一个 n 列的索引时，实际是创建了 MySQL 可利用的 n 个索引。多列索引可起几个索引的作用，因为可利用索引中最左边的列集来匹配行。这样的列集称为最左前缀。（这与索引一个列的前缀不同，索引一个列的前缀是利用该的前 n 个字符作为索引值。）
5. 不要过度索引。不要以为索引“越多越好”，什么东西都用索引是错的。每个额外的索引都要占用额外的磁盘空间，并降低写操作的性能，这一点我们前面已经介绍 过。在修改表的内容时，索引必须进行更新，有时可能需要重构，因此，索引越多，所花的时间越长。如果有一个索引很少利用或从不使用，那么会不必要地减缓表 的修改速度。此外，MySQL 在生成一个执行计划时，要考虑各个索引，这也要费时间。创建多余的索引给查询优化带来了更多的工作。索引太多，也可能会使 MySQL 选择不到所要使用的最好索引。只保持所需的索引有利于查询优化。如果想给已索引的表增加索引，应该考虑所要增加的索引是否是现有多列索引的最左 索引。如果是，则就不要费力去增加这个索引了，因为已经有了。
6. 考虑在列上进行的比较类型。索引可用于“<”、“<=”、“=”、“>=”、“>”和 BETWEEN 运算。在模式具有一个直接量前缀时，索引也用于 LIKE 运算。如果只将某个列用于其他类型的运算时（如 STRCMP( )），对其进行索引没有价值。

btree 索引与 hash 索引

对于 BTREE 和 HASH 索引，当使用=、<=>、IN、IS NULL 或者 IS NOT NULL 操作符时，关键元素与常量值的比较关系对应一个范围条件。Hash 索引还有一些其它特征：它们只用于使用=或<=>操作符的等式比较(但*很快*)。优化器不能使用 hash 索引来加速 ORDER BY 操作。(该类索引不能用来按顺序搜索下一个条目）。MySQL 不能确定在两个值之间大约有多少行(这被范围优化器用来确定使用哪个索引)。如果你将一个 MyISAM 表改为 hash-索引的 MEMORY 表，会影响一些查询。只能使用整个关键字来搜索一行。(用 B-树索引，任何关键字的最左面的前缀可用来找到行)。

对于 BTREE 索引，当使用>、<、>=、<=、BETWEEN、!=或者<>，或者 LIKE '*pattern*'(其中 '*pattern*'不以通配符开始)操作符时，关键元素与常量值的比较关系对应一个范围条件。

“常量值”系指：查询字符串中的常量、同一联接中的 const 或 system 表中的列、无关联子查询的结果、完全从前面类型的子表达式组成的表达式

下面是一些 WHERE 子句中有范围条件的查询的例子：

下列范围查询适用于 btree 索引和 hash 索引

```
SELECT * FROM t1     WHERE key_col = 1     OR key_col IN (15,18,20);
```

下列范围查询适用于 btree 索引

```
SELECT * FROM t1     WHERE key_col > 1     AND key_col < 10;
SELECT * FROM t1     WHERE key_col LIKE 'ab%'     OR key_col BETWEEN 'bar' AND 'foo';
```

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。不使用索引，MySQL 必须从第 1 条记录开始然后读完整个表直到找出相关的行。表越大，花费的时间越多。如果表中查询的列有一个索引，MySQL 能快速到达一个位置去搜寻到数据文件的中间，没有必要看所有数据。如果一个表有 1000 行，这比顺序读取至少快 100 倍。注意如果你需要访问大部分行，顺序读取要快得多，因为此时我们避免磁盘搜索。

大多数 MySQL 索引(PRIMARY KEY、UNIQUE、INDEX 和 FULLTEXT)在 B 树中存储。只是空间列类型的索引使用 R-树，并且 MEMORY 表还支持 hash 索引。

关于什么情况下数据库会使用索引以及什么情况下数据库不会使用索引的详细解释请看优化篇的相关章节，这里就不再累述。

锁机制和事务控制

如何加锁

锁定表的语法：

LOCK TABLES

tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}

[, tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...

解锁语法：

UNLOCK TABLES

innodb 的存储引擎提供行级锁，支持共享锁和排他锁两种锁定模式，以及四种不同的隔离级别。

死锁

InnoDB自动检测事务的死锁，并回滚一个或几个事务来防止死锁。InnoDB不能在MySQL LOCK TABLES设定表锁定的地方或者涉及InnoDB之外的存储引擎设置锁定的地方检测死锁。你必须通过设定innodb_lock_wait_timeout系统变量的值来解决这些情况。如果要依靠锁等待超时来解决死锁问题，对于更新事务密集的应用，将有可能导致大量事务的锁等待，导致系统异常，所以不推荐在一个事务中混合更新不同存储类型的表，也不推荐相同类型的表采用不同的锁定方式加锁。

事务控制

MySQL通过SET AUTOCOMMIT, START TRANSACTION, COMMIT和ROLLBACK等语句支持本地事务。

语法：

START TRANSACTION | BEGIN [WORK]

COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]

SET AUTOCOMMIT = {0 | 1}

默认情况下，mysql是autocommit的，如果需要通过明确的commit和rollback来提交和回滚事务，那么需要通过明确的事务控制命令来开始事务，这是和oracle的事务管理明显不同的地方，如果应用是从oracle数据库迁移到mysql数据库，则需要确保应用中是否对事务进行了明确的管理。

START TRANSACTION或BEGIN语句可以开始一项新的事务。

COMMIT和ROLLBACK用来提交或者回滚事务。

CHAIN和RELEASE子句分别用来定义在事务提交或者回滚之后的操作，chain会立即启动一个新事物，并且和刚才的事务具有相同的隔离级别，release则会断开和客户端的连接。

SET AUTOCOMMIT可以修改当前连接的提交方式，如果设置了SET AUTOCOMMIT=0，则设置之后的所有事务都需要通过明确的命令进行提交或者回滚。

如果我们只是对某些语句需要进行事务控制，则使用START TRANSACTION开始一个事务比较方便，这样事务结束之后可以自动回到自动提交的方式，如果我们希望我们所有的事务都不是自动提交的，那么通过修改AUTOCOMMIT来控制事务比较方便，这样不用在每个事务开始的时候再执行START TRANSACTION。

time	session_1	session_2
---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ->	mysql> select * from tt3; Empty set (0.00 sec) mysql> start transaction; Query OK, 0 rows affected (0.00 sec) mysql> insert into tt3 values('1',1); Query OK, 1 row affected (0.03 sec) mysql> commit; Query OK, 0 rows affected (0.05 sec)	mysql> select * from tt3; Empty set (0.00 sec) mysql> select * from tt3; Empty set (0.00 sec) mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 1 | 1.00 | +------+------+ 1 row in set (0.00 sec)

	mysql> insert into tt3 values('2',2); Query OK, 1 row affected (0.04 sec) 这个事务是按照自动提交执行的	
		mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 1 | 1.00 | | 2 | 2.00 | +------+------+ 2 rows in set (0.00 sec)

	mysql> start transaction;
	Query OK, 0 rows affected (0.00 sec)
	
	mysql> insert into tt3 values('3',3);
	Query OK, 1 row affected (0.00 sec)
	
	mysql> commit and chain;
	Query OK, 0 rows affected (0.05 sec)
	自动开始一个新的事务
	mysql> insert into tt3 values('4',4);
	Query OK, 1 row affected (0.00 sec)	

		mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 1 | 1.00 | | 2 | 2.00 | | 3 | 3.00 | +------+------+ 3 rows in set (0.00 sec)
	mysql> commit; Query OK, 0 rows affected (0.06 sec)	
		mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 1 | 1.00 | | 2 | 2.00 | | 3 | 3.00 | | 4 | 4.00 | +------+------+ 4 rows in set (0.00 sec)

开始一个事务，会造成一个隐含的unlock tables被执行：

time	session_1	session_2
---- ----	mysql> select * from tt3; Empty set (16.65 sec)	mysql> select * from tt3; Empty set (16.65 sec)
---- ----	mysql> lock table tt3 write; Query OK, 0 rows affected (0.00 sec)	

---- ---- ---- ---- ---- ---- ---- ---- ---- ->		mysql> select * from tt3; 等待
	mysql> insert into tt3 values('1',1); Query OK, 1 row affected (0.07 sec)	等待
	mysql> rollback; Query OK, 0 rows affected (0.00 sec)	等待
	mysql> start transaction; Query OK, 0 rows affected (0.00 sec)	等待
		mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 1 | 1.00 | +------+------+ 1 row in set (37.71 sec) 开始一个事务时，表锁被释放。 对 lock 方式加的表锁,不能通过 rollback 进行回滚。

因此，在同一个事务中，最好不使用不同存储引擎的表，否则rollback时需要对非事务类型的表进行特别的处理，因为commit、rollback只能对事务类型的表进行提交和回滚。

通常情况下，只对提交的事务纪录到二进制的日志中，但是如果一个事务中包含非事务类型的表，那么回滚操作也会被记录到二进制日志中，以确保非事务类型表的更新可以被复制到从的数据库中。

和oracle的事务管理相同，所有的DDL语句是不能回滚的，并且部分的DDL语句会造成隐式的提交。

在事务中可以通过定义savepoint，指定回滚事务的一个部分，但是不能指定提交事务的一个部分。对于复杂的应用，可以定义多个不同的savepoint，满足不同的条件时，回滚

不同的savepoint。需要注意的是，如果定义了相同名字的savepoint，则后面定义的savepoint会覆盖之前的定义。对于不再需要使用的savepoint，可以通过release savepoint命令删除savepoint，删除后的savepoint，不能再执行rollback to savepoint命令。

下面我们例子就是模拟回滚事务的一个部分，通过定义savepoint来指定需要回滚的事务的位置。

time	session_1	session_2
---- ---- ---- ---- ---- ---- ---- ---- ----	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 2 | 2.00 | | 3 | 3.00 | | 4 | 4.00 | +------+------+ 3 rows in set (0.00 sec)	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 2 | 2.00 | | 3 | 3.00 | | 4 | 4.00 | +------+------+ 3 rows in set (0.00 sec)
---- ---- ---- ---- ---- ->	mysql> start transaction; Query OK, 0 rows affected (0.00 sec) mysql> delete from tt3 where id = '2'; Query OK, 1 row affected (0.00 sec)	

	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 3 | 3.00 | | 4 | 4.00 | +------+------+ 3 rows in set (0.00 sec)	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 2 | 2.00 | | 3 | 3.00 | | 4 | 4.00 | +------+------+ 3 rows in set (0.00 sec)
	mysql> savepoint test; Query OK, 0 rows affected (0.00 sec) mysql> delete from tt3 where id = '3'; Query OK, 1 row affected (0.00 sec)	
	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 4 | 4.00 | +------+------+ 3 rows in set (0.00 sec)	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 2 | 2.00 | | 3 | 3.00 | | 4 | 4.00 | +------+------+ 3 rows in set (0.00 sec)
	mysql> rollback to savepoint test; Query OK, 0 rows affected (0.00 sec)	

	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 3 | 3.00 | | 4 | 4.00 | +------+------+ 2 rows in set (0.00 sec)	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 2 | 2.00 | | 3 | 3.00 | | 4 | 4.00 | +------+------+ 3 rows in set (0.00 sec)
	mysql> commit; Query OK, 0 rows affected (0.05 sec)	
	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 3 | 3.00 | | 4 | 4.00 | +------+------+ 2 rows in set (0.00 sec)	mysql> select * from tt3; +------+------+ | id | name | +------+------+ | 3 | 3.00 | | 4 | 4.00 | +------+------+ 2 rows in set (0.00 sec)

SQL 中的安全问题

SQL 注入简述

SQL Injection 攻击具有很大的危害，攻击者可以利用它读取、修改或者删除数据库内的数据，获取数据库中的用户名和密码等敏感信息，甚至可以 获得数据库管理员的权限。如果能够再利用 SQLServer 扩展存储过程和自定义扩展存储过程来执行一些系统命令，攻击者还可以获得该系统的控制权。而且，SQL Injection 也很难防范。网站管理员无法通过安装系统补丁或者进行简单的安全配置进行自我保护，一般的防火墙也无法拦截 SQL Injection 攻击。

SQL Injection 原理：

结构化查询语言(SQL)是一种用来和数据库交互的文本语言。SQL Injection 就是利用某些数据库的外部接口把用户数据插入到实际的数据库操作语言(SQL)当中，从而达到入侵数据库乃至操作系统的目的。它的产生主要是由于程序对用户输入 的数据没有进行严格的过滤，导致非法数据库查询语句的执行。

如下面的用户登陆验证程序：

$sql = "SELECT * FROM user WHERE username='$username' AND password='$password'";

$result = mysql_db_query($dbname, $sql);

如果我们提交如下 url：

http://127.0.0.1/injection/user.php?username=angel' or '1=1

那么就可以成功登陆系统，但是很显然这并不是我们预期的，同样我们也可以利用 sql 的注释语句实现 sql 注入，如下面的例子：

http://127.0.0.1/injection/user.php?username=angel'/*

http://127.0.0.1/injection/user.php?username=angel'%23

这样就把后面的语句给注释掉了！说说这两种提交的不同之处，我们提交的第一句是利用逻辑运算，第二、三句是根据 mysql 的特性，mysql 支持/*和#两种注释格式，所以我们提交的时候是把后面的代码注释掉，值得注意的是由于编码问题，在 IE 地址栏里提交#会变成空的，所以我们在地址栏提交的时候，应该提交%23，才会变成#，就成功注释了， 这个比逻辑运算简单得多了。

开发中可以采取的措施

prepareStatement + Bind-variable

对 Java ,Jsp 开发的应用,可以使用 prepareStatement + Bind-variable 来防止 sql 注入,另外从 PHP 5 开始，也在扩展的 mysqli 中支持 prepared statements，所以在使用这类语言作数据库开发时,强烈建议使用 prepareStatement + Bind-variable 来实现,而尽量不要使用拼接的 sql。

使用应用程序提供的转换函数：

很多应用程序接口都提供了对特殊的字符进行转换的函数，恰当的使用这些函数，可以防止应用程序用户输入使应用程序生成不期望的效果的语句的数值。

MySQL C API：使用 mysql_real_escape_string() API 调用。

MySQL++ ：使用 escape 和 quote 修饰符。

PHP ：使用 mysql_real_escape_string()函数（适用于 PHP 4.3.0，之前的版本请使用 mysql_escape_string(), PHP 4.0.3 之前的版本请使用 addslashes()）。从 PHP 5 开始,可以使用扩展的 mysqli，这是对 MYSQL 新特性的一个扩展支持，其中的一个优点就是支持 prepared statements。

Perl DBI ：使用 placeholders 或者 quote()方法。

Ruby DBI ：使用 placeholders 或者 quote()方法。

Java JDBC ：使用 PreparedStatement 和 placeholders。

自己定义函数进行校验

如果现有的转换函数仍然不能满足要求，则需要自己编写函数进行输入校验。输入验证是一个很复杂的问题。输入验证的途径可以分为以下几种：

整理数据使之变得有效；

拒绝已知的非法输入；

只接受已知的合法的输入。

所以如果想要获得最好的安全状态，目前最好的解决办法就是对用户提交或者可能改变的数据进行简单分类，分别应用正则表达式来对用户提供的输入数据进行严格的检测和验证。

SQL Mode 及相关问题

Mysql SQL Mode 简介

MySQL 服务器能够工作在不同的 SQL 模式下，并能针对不同的客户端以不同的方式应用这些模式。这样，应用程序就能对服务器操作进行量身定制以满足自己的需求。这类模式

定义了 MySQL 应支持的 SQL 语法，以及应该在数据上执行何种确认检查。这样，就能在众多不同的环境下、与其他数据库服务器一起更容易地使用 MySQL。可以使用“--sql-mode="modes"”选项，通过启动 **mysqld** 来设置默认的 SQL 模式。从 MySQL 4.1 开始，也能在启动之后，使用 SET [SESSION|GLOBAL] sql_mode='modes'语句，通过设置 sql_mode 变量更改模式。

通常在 linux 下安装完 mysql 后，默认的 sql-mode 值是空，在这种情形下 mysql 执行的是一种不严格的检查，例如日期字段可以插入'0000-00-00 00:00:00'这样的值，还有如果要插入的字段长度超过列定义的长度，那么 mysql 不会终止操作，而是会自动截断后面的字符继续插入操作，如下例：

```
mysql> create table t5 (c1 char(3));
mysql> insert into t5 values('abcd');
mysql> select * from t5;
+------+
| c1   |
+------+
| abc  |
+------+
1 row in set (0.00 sec)
```

我们发现插入的字符被自动截断了，但是如果我们本意希望如果长度超过限制就报错，那么我们可以设置 sql_mode 为 STRICT_TRANS_TABLES，如下：

```
mysql> set session sql_mode='STRICT_TRANS_TABLES'
```

这样我们再执行同样的操作，mysql 就会告诉我们插入的值太长，操作被终止，如下：

```
mysql> insert into t5 values('abcd');
ERROR 1406 (22001): Data too long for column 'c1' at row 1
```

经常使用的 sql_mode 值如下表：

Sql_mode 值	描述
ANSI	更改语法和行为，使其更符合标准 SQL。
STRICT_TRANS_TABLES	如果不能将给定的值插入到事务表中，则放弃该语句。对于非事务表，如果值出现在单行语句或多行语句的第 1 行，则放弃该语句。本节后

	面给出了更详细的描述。
TRADITIONAL	Make MySQL 的行为象“传统”SQL 数据库系统。该模式的简单描述是当在列中插入不正确的值时“给出错误而不是警告”。**注释**：一旦发现错误立即放弃 INSERT/UPDATE。如果你使用非事务存储引擎，这种方式不是你想要的，因为出现错误前进行的数据更改不会“滚动”，结果是更新“只进行了一部分”。

说明：如果把 sql_mode 的值设置成后面的两个值（也就是我们说的严格模式），那么当在列中插入或更新不正确的值时，mysql 将会给出错误，并且放弃 insert/update 操作。在我们的一般应用中建议使用这两种模式，而不是使用默认的空或 ANSI 模式。但是需要注意的问题是，如果数据库运行在严格模式下，并且你的存储引擎不支持事务，那么有数据不一致的风险存在，比如一组 sql 中有两个 dml 语句，如果后面的一个出现了问题，但是前面的已经操作成功，那么 mysql 并不能回滚前面的操作。因此说设置 sql_mode 需要应用人员权衡各种得失，从而得到一个合适的选择。

Sql_mode 的值还有很多，这里不再累述，可以参考相关的手册。

SQL Mode 与可移植性

如果 mysql 与其它异构数据库之间有数据移植的需求的话，那么下面的 sql_mode 的组合设置可以达到相应的效果：

数据库	Sql_mode 值
DB2	PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS
MAXDB	PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、 NO_AUTO_CREATE_USER
MSSQL	PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、 NO_FIELD_OPTIONS
ORACLE	PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS、NO_AUTO_CREATE_USER
POSTGRESQL	PIPES_AS_CONCAT、ANSI_QUOTES、IGNORE_SPACE、NO_KEY_OPTIONS、NO_TABLE_OPTIONS、NO_FIELD_OPTIONS

SQL Mode 与数据效验

SQL Mode 还可以实现对数据效验和转移等功能如:

- 效验日期数据合法性.
- 在 INSERT 或 UPDATE 过程中，如果被零除(或 MOD(X，0))，则产生错误
- 将‘″’视为识别符引号(‘`’引号字符)
- 禁用反斜线字符(‘\’)做为字符串内的退出字符。启用 NO_BACKSLASH_ESCAPES 模式，反斜线则成为普通字符。
- 将||视为字符串连接操作符（＋）(同 CONCAT())，而不视为 OR。

常用 SQL 技巧

检索包含最大/最小值的行

MIN([DISTINCT] *expr*), MAX([DISTINCT] *expr*)

返回 *expr* 的最小值和最大值。 MIN() 和 MAX() 的取值可以是一个字符串参数；在这些情况下， 它们返回最小或最大字符串值。DISTINCT 关键词可以被用来查找 *expr* 的不同值的最小或最大值，然而，这产生的结果与省略 DISTINCT 的结果相同。

若找不到匹配的行，MIN()和 MAX()返回 NULL 。

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
    ->       FROM student
    ->       GROUP BY student_name;
```

巧用 rand()/rand(n)提取随机行

可按照如下的随机顺序检索数据行，如下：

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

ORDER BY RAND()同 LIMIT 的结合可以从一组列中选择随机样本，如下：

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d
       ORDER BY RAND() LIMIT 1000;
```

利用 group by 的 with rollup 子句做统计

```
create table sales
(
      year       int not null,
      country varchar(20) not null,
      product varchar(32) not null,
      profit    int
);
 select year, sum(profit) from sales group by year;
   select year, sum(profit) from sales group by year with rollup;
     select year, country, product, sum(profit) from sales group by year, country, product;
   select year, country, product, sum(profit) from sales group by year, country, product with rollup;
 insert into sales values(2004,'china','tnt2004',2001);
 insert into sales values(2004,'china','tnt2004',2002);
 insert into sales values(2004,'china','tnt2004',2003);
 insert into sales values(2005,'china','tnt2005',2004);
 insert into sales values(2005,'china','tnt2005',2005);
 insert into sales values(2005,'china','tnt2005',2006);
 insert into sales values(2005,'china','tnt2005',2007);
 insert into sales values(2005,'china','tnt2005',2008);
 insert into sales values(2005,'china','tnt2005',2009);
 insert into sales values(2006,'china','tnt2006',2010);
 insert into sales values(2006,'china','tnt2006',2011);
 insert into sales values(2006,'china','tnt2006',2012);
```

```
 select year, country, product, sum(profit)
   from sales
   group by year, country, product;

select year, country, product, sum(profit)
from sales
group by year, country, product with rollup;

SELECT year, country, product, SUM(profit)
FROM sales
GROUP BY year, country, product WITH ROLLUP
LIMIT 5;
```

当你使用 ROLLUP时，你不能同时使用 ORDER BY子句进行结果排序。换言之， ROLLUP 和ORDER BY 是互相排斥的

LIMIT 用在 ROLLUP 后面

用 bit group functions 做统计

Group by 统计语句的一般用法:

```
SELECT student_name, AVG([DISTINCT] expr)
    ->       FROM student
    ->       GROUP BY student_name;
```

在本小节,主要介绍 group by 语句和 bit_and ,bit_or 函数共同使用完成统计工作,先来了解一下 bit_and 和 bit_or 函数,举例如下:

```
mysql> CREATE TABLE `ta` (
    ->   `id` smallint(5) unsigned NOT NULL default '0',
    ->   KEY `id` (`id`)
    -> ) TYPE=MyISAM;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> INSERT INTO `ta` VALUES("1"),("2"),("3"),("4");
Query OK, 8 rows affected (0.00 sec)
Records: 4   Duplicates: 0   Warnings: 0
mysql> SELECT BIT_OR(id) from ta;
+------------+
| BIT_OR(id) |
+------------+
|          7 |
+------------+
1 row in set (0.00 sec)
#      ..0001
#      ..0010
#      ..0011
#      ..0100
#   OR ..0000
#   ---------
#      ..0111
mysql> SELECT BIT_AND(id) from ta;
+-------------+
| BIT_AND(id) |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)
#       ..0001
#       ..0010
#       ..0011
#       ..0100
#   AND ..1111
#   ----------
```

..0000

熟悉了 bit_and 和 bit_or 后我们一起来学习一下 bit_and ,bit_or 和 group by 函数共同使用进行统计工作.

针对 上面的表 ta 我们增加字段 cust_type 并按这个字段的值进行分类统计

```
alter table ta    add column    cust_type varchar(100);
update ta set cust_type ='2'    where id>3;
update ta set cust_type ='1'    where cust_type is null;
mysql> SELECT cust_type,BIT_OR(id) from ta    group by cust_type;
+-----------+------------+
| cust_type | BIT_OR(id) |
+-----------+------------+
| 1                |                3 |
| 2                |                4 |
+-----------+------------+
2 rows in set (0.00 sec)

mysql> SELECT cust_type,BIT_and(id) from ta    group by cust_type;
+-----------+-------------+
| cust_type | BIT_and(id) |
+-----------+-------------+
| 1                |                 0 |
| 2                |                 4 |
+-----------+-------------+
2 rows in set (0.00 sec)
```

BIT_AND(*expr*)

返回 *expr* 中所有比特的 bitwise AND 。计算执行的精确度为 64 比特(BIGINT) 。

若找不到匹配的行，则这个函数返回 18446744073709551615 。(这是无符号 BIGINT 值，所有比特被设置为 1）。

BIT_OR(*expr*)

返回 *expr* 中所有比特的 bitwise OR。计算执行的精确度为 64 比特(BIGINT) 。

若找不到匹配的行，则函数返回 0 。

BIT_XOR(*expr*)

返回 *expr* 中所有比特的 bitwise XOR。计算执行的精确度为 64 比特(BIGINT) 。

若找不到匹配的行，则函数返回 0 。

其他需注意的问题

数据库名、表名大小写问题

在 MySQL 中，数据库对应数据目录中的目录。数据库中的每个表至少对应数据库目录中的一个文件(也可能是多个，取决于存储引擎)。因此，所使用操作系统的大小写敏感性决定了数据库名和表名的大小写敏感性。这说明在大多数 Unix 中数据库名和表名对大小写敏感，而在 Windows 中对大小写不敏感。一个显著的例外情况是 Mac OS X，它基于 Unix 但使用默认文件系统类型(HFS+)，对大小写不敏感。然而，Mac OS X 也支持 UFS 卷，该卷对大小写敏感，就像 Unix 一样。

注释：尽管在某些平台中数据库名和表名对大小写不敏感，不应在同一查询中使用不同的大小写来引用给定的数据库或表。下面的查询不会工作，因为它同时引用了表 my_tables 和 as MY_tables：

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

列、索引、存储子程序和触发器名在任何平台上对大小写不敏感，列的别名也不敏感。默认情况，表别名在 Unix 中对大小写敏感，但在 Windows 或 Mac OS X 中对大小写不敏感。下面的查询在 Unix 中不会工作，因为它同时引用了别名 a 和 A：

```
mysql> SELECT col_name FROM tbl_name AS a
    -> WHERE a.col_name = 1 OR A.col_name = 2;
```

然而，该查询在 Windows 中是可以的。要想避免出现差别，最好采用一致的转换，例如总是用小写创建并引用数据库名和表名。在大多数移植和使用中建议使用该转换。

在 MySQL 中如何在硬盘上保存和使用表名和数据库名由 lower_case_tables_name 系统变量确定，可以在启动 **mysqld** 时设置。lower_case_tables_name 可以采用下面的任一值：

值	含义
0	使用 CREATE TABLE 或 CREATE DATABASE 语句指定的大写和小写在硬盘上保存表名和数据库名。名称比较对大小写敏感。在 Unix 系统中的默认设置即如此。请注意如果在大小写不敏感的文件系统上用--lower-case-table-names=0 强制设为 0，并且使用不同的大小写访问 MyISAM 表名，会导致索引破坏。
1	表名在硬盘上以小写保存，名称比较对大小写敏感。MySQL 将所有表名转换为小写以便存储和查找。该行为也适合数据库名和表的别名。该值为 Windows 和 Mac OS X 系统中的默认值。
2	表名和数据库名在硬盘上使用 CREATE TABLE 或 CREATE DATABASE 语句指定的大小写进行保存，但 MySQL 将它们转换为小写以便查找。名称比较对大小写敏感。**注释：** *只*在对大小写不敏感的文件系统上适用！InnoDB 表名以小写保存，例如 lower_case_tables_name=1。

在 Windows 和 Mac OS X 中，lower_case_tables_name 的 默认值是 1。

如果只在一个平台上使用 MySQL，通常不需要更改 lower_case_tables_name 变量。然而，如果你想要在对大小写敏感不同的文件系统的平台之间转移表，会遇到困难。例如，

在 Unix 中，my_tables 和 MY_tables 是两个不同的表，但在 Windows 中，这两个表名相同。要想避免由于数据库或表名的大小写造成的数据转移问题，可使用两个选项：

在任何系统中可以使用 lower_case_tables_name=1。使用该选项的不利之处是当使用 SHOW TABLES 或 SHOW DATABASES 时，看不出名字原来是用大写还是小写。

在 Unix 中使用 lower_case_tables_name=0，在 Windows 中使用 lower_case_tables_name=2。这样了可以保留数据库名和表名的大小写。不利之处是必须确保在 Windows 中查询总是用正确大小写引用数据库名和表名。如果将查询转移到 Unix 中，由于在 Unix 中大小写很重要，如果大小写不正确，它们不工作。

例外：如果你正使用 InnoDB 表，在任何平台上均应将 lower_case_tables_name 设置为 1，以强制将名转换为小写。

请注意在 Unix 中将 lower_case_tables_name 设置为 1 之前，重启 **mysqld** 之前，必须先将旧的数据库名和表名转换为小写

使用外键需注意的地方

在 MySQL 中，InnoDB 表支持对外部关键字约束条件的检查。

对于除 InnoDB 类型的表，当使用 REFERENCES tbl_name(col_name)子句定义列时可以使用外部关键字，该子句没有实际的效果，只作为备忘录或注释来提醒，你目前正定义的列指向另一个表中的一个列。执行该语句时，实现下面很重要：

MySQL 不执行表 *tbl_name* 中的动作，例如作为你正定义的表中的行的动作的响应而删除行；换句话说，该句法不会致使 ON DELETE 或 ON UPDATE 行为（如果你在 REFERENCES 子句中写入 ON DELETE 或 ON UPDATE 子句，将被忽略）。

该句法可以创建一个 *column*；但不创建任何索引或关键字。

如果用该句法定义 InnoDB 表，将会导致错误。

你可以使用作为联接列创建的列，如下所示：

```
CREATE TABLE person (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
    PRIMARY KEY (id)
);

INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
```

```
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Antonio Paz        |
|  2 | Lilliana Angelovska |
+----+--------------------+

SELECT * FROM shirt;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+

SELECT s.* FROM person p, shirt s
  WHERE p.name LIKE 'Lilliana%'
    AND s.owner = p.id
    AND s.color <> 'white';

+----+-------+--------+-------+
```

```
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

按照这种方式使用，REFERENCES 子句不会显示在 SHOW CREATE TABLE 或 DESCRIBE 的输出中:

```
SHOW CREATE TABLE shirt\G
*************************** 1. row ***************************
Table: shirt
Create Table: CREATE TABLE `shirt` (
`id` smallint(5) unsigned NOT NULL auto_increment,
`style` enum('t-shirt','polo','dress') NOT NULL,
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,
PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

在列定义中，按这种方式使用 REFERENCES 作为注释或“提示”适用于表 MyISAM 和 BerkeleyDB。

第二篇 优化篇

SQL 优化

优化 SQL 的一般步骤

通过 show status 和应用特点了解各种 SQL 的执行频率

通过 SHOW STATUS 可以提供服务器状态信息，也可以使用 mysqladmin extended-status 命令获得。SHOW STATUS 可以根据需要显示 session 级别的统计结果和 global 级别的统计结果。

以下几个参数对 Myisam 和 Innodb 存储引擎都计数：

1. Com_select 执行 select 操作的次数，一次查询只累加 1；
2. Com_insert 执行 insert 操作的次数，对于批量插入的 insert 操作，只累加一次；
3. Com_update 执行 update 操作的次数；
4. Com_delete 执行 delete 操作的次数；

以下几个参数是针对 Innodb 存储引擎计数的，累加的算法也略有不同：

1. Innodb_rows_read select 查询返回的行数；
2. Innodb_rows_inserted 执行 Insert 操作插入的行数；
3. Innodb_rows_updated 执行 update 操作更新的行数；
4. Innodb_rows_deleted 执行 delete 操作删除的行数；

通过以上几个参数，可以很容易的了解当前数据库的应用是以插入更新为主还是以查询操作为主，以及各种类型的 SQL 大致的执行比例是多少。对于更新操作的计数，是对执行次数的计数，不论提交还是回滚都会累加。

对于事务型的应用，通过 Com_commit 和 Com_rollback 可以了解事务提交和回滚的情况，对于回滚操作非常频繁的数据库，可能意味着应用编写存在问题。

此外，以下几个参数便于我们了解数据库的基本情况：

1. Connections 试图连接 Mysql 服务器的次数

2. Uptime 服务器工作时间
3. Slow_queries 慢查询的次数

定位执行效率较低的 SQL 语句：

可以通过以下两种方式定位执行效率较低的 SQL 语句：

1. 可以通过慢查询日志定位那些执行效率较低的 sql 语句，用 --log-slow-queries[=file_name]选项启动时，mysqld 写一个包含所有执行时间超过 long_query_time 秒的 SQL 语句的日志文件。可以链接到管理维护中的相关章节。
2. 慢查询日志在查询结束以后才纪录,所以在应用反映执行效率出现问题的时候查询慢查询日志并不能定位问题，可以使用 show processlist 命令查看当前 MySQL 在进行的线程，包括线程的状态，是否锁表等等，可以实时的查看 SQL 执行情况，同时对一些锁表操作进行优化。

通过 EXPLAIN 分析低效 SQL 的执行计划：

通过以上步骤查询到效率低的 SQL 后，我们可以通过 explain 或者 desc 获取 MySQL 如何执行 SELECT 语句的信息，包括 select 语句执行过程表如何连接和连接的次序。

explain 可以知道什么时候必须为表加入索引以得到一个使用索引来寻找记录的更快的 SELECT。

```
mysql> explain select sum(moneys) from sales a,companys b where a.company_id = b.id
and a.year = 2006;
+---------------+---------+----------+---------------+---------------+---------
-+----------+---------------+
| select_type   | table | type  | possible_keys| key           | key_len   |   rows
    | Extra     |
+---------------+---------+----------+---------------+---------------+---------
-+----------+---------------+
```

```
| SIMPLE      | b      | index | PRIMARY       | PRIMARY       | 4       |    1 |  Using  index
    |
| SIMPLE      | a      | ALL   | NULL          | NULL          | NULL    |   12 |  Using  where
    |
+--------------+---------+----------+---------------+---------------+---------
-+----------+---------------+
2 rows in set (0.02 sec)
```

select_type: select 类型

table: 输出结果集的表

type: 表示表的连接类型

当表中仅有一行是type的值为system是最佳的连接类型;

当select操作中使用索引进行表连接时type的值为ref;

当select的表连接没有使用索引时，经常会看到type的值为ALL，表示对该表进行了全表扫描，这时需要考虑通过创建索引来提高表连接的效率。

possible_keys: 表示查询时,可以使用的索引列.

key: 表示使用的索引

key_len: 索引长度

rows: 扫描范围

Extra: 执行情况的说明和描述

确定问题，并采取相应的优化措施：

经过以上步骤，基本可以确认问题出现的原因，可以根据情况采取相应的措施，进行优化提高执行的效率。

例如上面的例子，我们确认是对 a 表的全表扫描导致效率的不理想，我们对 a 表的 year 字段创建了索引，查询需要扫描的行数明显较少。

```
mysql> explain select sum(moneys) from sales a,companys b where a.company_id = b.id
and a.year = 2006;
```

```
+--------------+---------+----------+---------------+---------------+---------
-+----------+---------------+
| select_type  | table | type  | possible_keys| key           | key_len   |   rows
    | Extra     |
+--------------+---------+----------+---------------+---------------+---------
-+----------+---------------+
| SIMPLE    | b     | index | PRIMARY   | PRIMARY   | 4     |    1 |  Using  index
    |
| SIMPLE    | a     | ref       | year      | year      | 4     |    3 |Using where
    |
+--------------+---------+----------+---------------+---------------+---------
-+----------+---------------+
2 rows in set (0.02 sec)
```

索引问题

索引的存储分类

myisam 表的数据文件和索引文件是自动分开的；innodb 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。

创建索引语法如下：

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
[USING index_type]
ON tbl_name (index_col_name,...)
index_col_name:
col_name [(length)] [ASC | DESC]
```

索引的存储类型目前只有两种（btree 和 hash），具体和表的模式相关：

```
myisam     btree
innodb      btree
memory/heap hash，btree
```

mysql 目前不支持函数索引，只能对列的前一部分（length）进行索引，例：

create index ind_test on table1(name(5)),

对于 char 和 varchar 列，使用前缀索引将大大节省空间。

10.2.2 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。

查询要使用索引最主要的条件是查询条件中需要使用索引关键字，如果是多列索引，那么只有查询条件使用了多列关键字最左边的前缀时，才可以使用索引，否则将不能使用索引。

下列情况下，Mysql 不会使用已有的索引：

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 key_part1 均匀分布在 1 和 100 之间，下列查询中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 heap 表并且 where 条件中不用＝索引列，其他> 、<、 >=、 <=均不使用索引；
3. 如果不是索引列的第一部分；
4. 如果 like 是以%开始；
5. 对 where 后边条件为字符串的一定要加引号，字符串如果为数字 mysql 会自动转为字符串，但是不使用索引。

10.2.3 查看索引使用情况

如果索引正在工作，Handler_read_key 的值将很高，这个值代表了一个行被索引值读的次数，很低的值表明增加索引得到的性能改善不高，因为索引并不经常使用。

Handler_read_rnd_next 的值高则意味着查询运行低效，并且应该建立索引补救。这个值的含义是在数据文件中读下一行的请求数。如果你正进行大量的表扫描，该值较高。通常说明表索引不正确或写入的查询没有利用索引。

语法：

mysql> show status like 'Handler_read%';

两个简单实用的优化方法

定期分析表：

ANALYZE TABLE

语法：

ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...

本语句用于分析和存储表的关键字分布。在分析期间，使用一个读取锁定对表进行锁定。这对于 MyISAM, BDB 和 InnoDB 表有作用。对于 MyISAM 表，本语句与使用 myisamchk -a 相当。

CHECK TABLE

语法：

CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}

检查一个或多个表是否有错误。CHECK TABLE 对 MyISAM 和 InnoDB 表有作用。对于 MyISAM 表，关键字统计数据被更新。

CHECK TABLE 也可以检查视图是否有错误，比如在视图定义中被引用的表已不存在。

CHECKSUM TABLE

语法：

CHECKSUM TABLE tbl_name [, tbl_name] ... [QUICK | EXTENDED]

报告一个表校验和。

使用 optimize table：

OPTIMIZE TABLE

语法：

OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...

如果已经删除了表的一大部分，或者如果您已经对含有可变长度行的表（含有

VARCHAR, BLOB 或 TEXT 列的表）进行了很多更改，则应使用 OPTIMIZE TABLE。被删除的记录被保持在链接清单中，后续的 INSERT 操作会重新使用旧的记录位置。您可以使用 OPTIMIZE TABLE 来重新利用未使用的空间，并整理数据文件的碎片。

OPTIMIZE TABLE 只对 MyISAM, BDB 和 InnoDB 表起作用。

常用 SQL 的优化

大批量插入数据：

1. 对于 Myisam 类型的表，可以通过以下方式快速的导入大量的数据。

   ALTER TABLE tblname DISABLE KEYS;

   loading the data

   ALTER TABLE tblname ENABLE KEYS;

   这两个命令用来打开或者关闭 Myisam 表非唯一索引的更新。在导入大量的数据到一个非空的 Myisam 表时，通过设置这两个命令，可以提高导入的效率。对于导入大量数据到一个空的 Myisam 表，默认就是先导入数据然后才创建索引的，所以不用进行设置。()

2. 而对于 Innodb 类型的表，这种方式并不能提高导入数据的效率。对于 Innodb 类型的表，我们有以下几种方式可以提高导入的效率：

a. 因为 Innodb 类型的表是按照主键的顺序保存的，所以将导入的数据按照主键的顺序排列，可以有效的提高导入数据的效率。如果 Innodb 表没有主键，那么系统会默认创建一个内部列作为主键，所以如果可以给表创建一个主键，将可以利用这个优势提高导入数据的效率。

b. 在导入数据前执行 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，可以提高导入的效率。

c. 如果应用使用自动提交的方式，建议在导入前执行 SET AUTOCOMMIT=0，关闭自动提交，导入结束后再执行 SET AUTOCOMMIT=1，打开自动提交，也可以提高导入的效率。

优化 insert 语句：

3. 如果你同时从同一客户插入很多行，使用多个值表的 INSERT 语句。这比使用分开

INSERT 语句快(在一些情况中几倍)。

Insert into test values(1,2),(1,3),(1,4)…

4. 如果你从不同客户插入很多行，能通过使用 INSERT DELAYED 语句得到更高的速度。Delayed 的含义是让 insert 语句马上执行，其实数据都被放在内存的队列中，并没有真正写入磁盘；这比每条语句分别插入要快的多；LOW_PRIORITY 刚好相反，在所有其他用户对表的读写完后才进行插入；
5. 将索引文件和数据文件分在不同的磁盘上存放（利用建表中的选项）；
6. 如果进行批量插入，可以增加 bulk_insert_buffer_size 变量值的方法来提高速度，但是，这只能对 myisam 表使用；
7. 当从一个文本文件装载一个表时，使用 LOAD DATA INFILE。这通常比使用很多 INSERT 语句快 20 倍；
8. 根据应用情况使用 replace 语句代替 insert；
9. 根据应用情况使用 ignore 关键字忽略重复记录。

优化 group by 语句：

默认情况下，MySQL 排序所有 GROUP BY col1，col2，....。查询的方法如同在查询中指定 ORDER BY col1，col2，...。如果显式包括一个包含相同的列的 ORDER BY 子句，MySQL 可以毫不减速地对它进行优化，尽管仍然进行排序。

如果查询包括 GROUP BY 但你想要避免排序结果的消耗，你可以指定 ORDER BY NULL 禁止排序。

例如：

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

优化 order by 语句：

在某些情况中，MySQL 可以使用一个索引来满足 ORDER BY 子句，而不需要额外的排序。where 条件和 order by 使用相同的索引，并且 order by 的顺序和索引顺序相同，并且 order by 的字段都是升序或者都是降序。

例如：下列 sql 可以使用索引。

```
SELECT * FROM t1 ORDER BY key_part1,key_part2,... ;
SELECT * FROM t1 WHERE key_part1=1 ORDER BY key_part1 DESC, key_part2 DESC;
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 DESC;
```

但是以下情况不使用索引：

```
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
--order by 的字段混合 ASC 和 DESC
SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
--用于查询行的关键字与 ORDER BY 中所使用的不相同
SELECT * FROM t1 ORDER BY key1, key2;
--对不同的关键字使用 ORDER BY：
```

优化 join 语句：

Mysql4.1 开始支持 SQL 的子查询。这个技术可以使用 SELECT 语句来创建一个单列的查询结果，然后把这个结果作为过滤条件用在另一个查询中。使用子查询可以一次性的完成很多逻辑上需要多个步骤才能完成的 SQL 操作，同时也可以避免事务或者表锁死，并且写起来也很容易。但是，有些情况下，子查询可以被更有效率的连接(JOIN).. 替代。

假设我们要将所有没有订单记录的用户取出来，可以用下面这个查询完成：

```
SELECT * FROM customerinfo WHERE CustomerID NOT in (SELECT CustomerID FROM salesinfo )
```

如果使用连接(JOIN).. 来完成这个查询工作，速度将会快很多。尤其是当 salesinfo 表中对 CustomerID 建有索引的话，性能将会更好，查询如下：

```
SELECT * FROM customerinfo
LEFT JOIN salesinfoON customerinfo.CustomerID=salesinfo.CustomerID
WHERE salesinfo.CustomerID IS NULL
```

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。

mysql 如何优化 or 条件：

对于 or 子句，如果要利用索引，则 or 之间的每个条件列都必须用到索引；如果没有索引，则应该考虑增加索引。

查询优先还是更新（insert、update、delete）优先：

MySQL 还允许改变语句调度的优先级，它可以使来自多个客户端的查询更好地协作，这样单个客户端就不会由于锁定而等待很长时间。改变优先级还可以确保特定类型的查询被处理得更快。

我们首先应该确定应用的类型，判断应用是以查询为主还是以更新为主的，是确保查询效率还是确保更新的效率，决定是查询优先还是更新优先。

下面我们提到的改变调度策略的方法主要是针对 Myisam 存储引擎的，对于 Innodb 存储引擎，语句的执行是由获得行锁的顺序决定的。

MySQL 的默认的调度策略可用总结如下：

1. 写入操作优先于读取操作。
2. 对某张数据表的写入操作某一时刻只能发生一次，写入请求按照它们到达的次序来处理。
3. 对某张数据表的多个读取操作可以同时地进行。

MySQL 提供了几个语句调节符，允许你修改它的调度策略：

1. LOW_PRIORITY关键字应用于DELETE、INSERT、LOAD DATA、REPLACE和UPDATE。
2. HIGH_PRIORITY关键字应用于SELECT和INSERT语句。
3. DELAYED关键字应用于INSERT和REPLACE语句。

如果写入操作是一个 LOW_PRIORITY（低优先级）请求，那么系统就不会认为它的优先级高于读取操作。在这种情况下，如果写入者在等待的时候，第二个读取者到达了，那么就允许第二个读取者插到写入者之前。只有在没有其它的读取者的时候，才允许写入者开始操作。这种调度修改可能存在 LOW_PRIORITY 写入操作永远被阻塞的情况。

SELECT 查询的 HIGH_PRIORITY（高优先级）关键字也类似。它允许 SELECT 插入正

在等待的写入操作之前，即使在正常情况下写入操作的优先级更高。另外一种影响是，高优先级的 SELECT 在正常的 SELECT 语句之前执行，因为这些语句会被写入操作阻塞。

如果你希望所有支持 LOW_PRIORITY 选项的语句都默认地按照低优先级来处理，那么请使用--low-priority-updates 选项来启动服务器。通过使用 INSERT HIGH_PRIORITY 来把 INSERT 语句提高到正常的写入优先级，可以消除该选项对单个 INSERT 语句的影响。

使用 SQL 提示：

SELECT SQL_BUFFER_RESULTS ...

将强制 MySQL 生成一个临时结果集。只要所有临时结果集生成后，所有表上的锁定均被释放。这能在遇到表锁定问题时或要花很长时间将结果传给客户端时有所帮助。

当处理一个会让客户端耗费点时间才能处理的大结果集时，可以考虑使用 SQL_BUFFER_RESULT 提示字。这样可以告诉 MySQL 将结果集保存在一个临时表中，这样可以尽早的释放各种锁。

- USE INDEX

在你查询语句中表名的后面，添加 USE INDEX 来提供你希望 MySQ 去参考的索引列表，就可以让 MySQL 不再考虑其他可用的索引。

Eg:SELECT * FROM mytable USE INDEX (mod_time, name) ...

- IGNORE INDEX

如果你只是单纯的想让 MySQL 忽略一个或者多个索引，可以使用 IGNORE INDEX 作为 Hint。

Eg:SELECT * FROM mytale IGNORE INDEX (priority) ...

- FORCE INDEX

为强制 MySQL 使用一个特定的索引，可在查询中使用 FORCE INDEX 作为 Hint。

Eg:SELECT * FROM mytable FORCE INDEX (mod_time) ...

其他优化措施

1. 使用持久的连接数据库以避免连接开销。
2. 经常检查所有查询确实使用了必要的索引。
3. 避免在频繁更新的表上执行复杂的 SELECT 查询，以避免与锁定表有关的由于读、写

冲突发生的问题。

4. 对于没有删除的行操作的 MyISAM 表，插入操作和查询操作可以并行进行，因为没有删除操作的表查询期间不会阻塞插入操作．对于确实需要执行删除操作的表，尽量在空闲时间进行批量删除操作，避免阻塞其他操作。
5. 充分利用列有默认值的事实。只有当插入的值不同于默认值时，才明确地插入值。这减少 MySQL 需要做的语法分析从而提高插入速度。
6. 对经常访问的可以重构的数据使用内存表，可以显著提高访问的效率。
7. 通过复制可以提高某些操作的性能。可以在复制服务器中分布客户的检索以均分负载。为了防止备份期间对应用的影响，可以在复制服务器上执行备份操作。
8. 表的字段尽量不使用自增长变量，在高并发情况下该字段的自增可能对效率有比较大的影响，推荐通过应用来实现字段的自增长。

优化数据库对象

优化表的数据类型

表需要使用何种数据类型，是需要根据应用来判断的。虽然应用设计的时候需要考虑字段的长度留有一定的冗余，但是不推荐让很多字段都留有大量的冗余，这样即浪费存储也浪费内存。

我们可以使用 PROCEDURE ANALYSE()对当前已有应用的表类型的判断，该函数可以对数据表中的列的数据类型提出优化建议，可以根据应用的实际情况酌情考虑是否实施优化。

语法：

```
SELECT * FROM tbl_name PROCEDURE ANALYSE();
SELECT * FROM tbl_name PROCEDURE ANALYSE(16,256);
```

输出的每一列信息都会对数据表中的列的数据类型提出优化建议。第二个例子告诉 PROCEDURE ANALYSE()不要为那些包含的值多于 16 个或者 256 字节的 ENUM 类型提出建议。如果没有这样的限制，输出信息可能很长；ENUM 定义通常很难阅读。

在对字段类型进行优化时，可以根据统计信息并结合应用的实际情况对其进行优化。

通过拆分，提高表的访问效率

这里我们所说的拆分，主要是针对 Myisam 类型的表，拆分的方法可以分成两种情况：

1. 纵向拆分：

纵向拆分是只按照应用访问的频度，将表中经常访问的字段和不经常访问的字段拆分成两个表，经常访问的字段尽量是定长的，这样可以有效的提高表的查询和更新的效率。

2. 横向拆分：

横向拆分是指按照应用的情况，有目的的将数据横向拆分成几个表或者通过分区分到多个分区中，这样可以有效的避免 Myisam 表的读取和更新导致的锁问题。

逆规范化

数据库德规范化设计强调数据的独立性，数据应该尽可能少地冗余，因为存在过多的冗余数据，这就意味着要占用了更多的物理空间，同时也对数据的维护和一致性检查带来了问题。

但是对于查询操作很多的应用，一次查询可能需要访问多表进行，如果通过冗余纪录在相同表中，更新的代价增加不多，但是查询操作效率可以有明显提高，这种情况就可以考虑通过冗余数据来提高效率。

使用冗余统计表

使用 create temporary table 语法，它是基于 session 的表，表的数据保存在内存里面，当 session 断掉后，表自然消除。

对于大表的统计分析，如果统计的数据量不大，利用 insert。。。select 将数据移到临时表中比直接在大表上做统计要效率更高。

选择更合适的表类型

1、如果应用出现比较严重的锁冲突，请考虑是否更改存储引擎到 innodb，行锁机制可以有效的减少锁冲突的出现。

2、如果应用查询操作很多，且对事务完整性要求不严格，则可以考虑使用 Myisam 存储引擎。

更多存储引擎选择的原则，请参考开发篇的相关章节。

锁问题

获取锁等待情况

可以通过检查 table_locks_waited 和 table_locks_immediate 状态变量来分析系统上的表锁定争夺：

```
mysql> show status like 'Table%';
+----------------------------+---------+
| Variable_name              | Value   |
+----------------------------+---------+
| Table_locks_immediate      | 105     |
| Table_locks_waited         | 3       |
+----------------------------+---------+
2 rows in set (0.00 sec)
```

可以通过检查 Innodb_row_lock 状态变量来分析系统上的行锁的争夺情况：

```
mysql> show status like 'innodb_row_lock%';
+-------------------------------------+---------+
| Variable_name                       | Value   |
+-------------------------------------+---------+
| Innodb_row_lock_current_waits       | 0       |
| Innodb_row_lock_time                | 2001    |
| Innodb_row_lock_time_avg            | 667     |
| Innodb_row_lock_time_max            | 845     |
| Innodb_row_lock_waits               | 3       |
+-------------------------------------+---------+
5 rows in set (0.00 sec)
```

另外，针对 Innodb 类型的表，如果需要察看当前的锁等待情况，可以设置 InnoDB

Monitors，然后通过 Show innodb status 察看，设置的方式是：

CREATE TABLE innodb_monitor(a INT) ENGINE=INNODB;

监视器可以通过发出下列语句来被停止：

DROP TABLE innodb_monitor;

设置监视器后，在 show innodb status 的显示内容中，会有详细的当前锁等待的信息，包括表名、锁类型、锁定记录的情况等等，便于进行进一步的分析和问题的确定。打开监视器以后，默认情况下每 15 秒会向日志中记录监控的内容，如果长时间打开会导致.err 文件变得非常的巨大，所以我们在确认问题原因之后，要记得删除监控表以关闭监视器。或者通过使用--console 选项来启动服务器以关闭写日志文件。

什么情况下使用表锁

表级锁在下列几种情况下比行级锁更优越：

1. 很多操作都是读表。
2. 在严格条件的索引上读取和更新，当更新或者删除可以用单独的索引来读取得到时：
3. UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
4. DELETE FROM tbl_name WHERE unique_key_col=key_value;
5. SELECT 和 INSERT 语句并发的执行，但是只有很少的 UPDATE 和 DELETE 语句。
6. 很多的扫描表和对全表的 GROUP BY 操作，但是没有任何写表。

什么情况下使用行锁

行级锁定的优点：

1. 当在许多线程中访问不同的行时只存在少量锁定冲突。
2. 回滚时只有少量的更改。
3. 可以长时间锁定单一的行。

行级锁定的缺点：

1. 比页级或表级锁定占用更多的内存。
2. 当在表的大部分中使用时，比页级或表级锁定速度慢，因为你必须获取更多的锁。
3. 如果你在大部分数据上经常进行 GROUP BY 操作或者必须经常扫描整个表，比其它锁定

明显慢很多。

4. 用高级别锁定，通过支持不同的类型锁定，你也可以很容易地调节应用程序，因为其锁成本小于行级锁定。

insert …select …带来的问题

当使用 insert...select...进行记录的插入时，如果 select 的表是 innodb 类型的，不论 insert 的表是什么类型的表，都会对 select 的表的纪录进行锁定。

对于那些从 oracle 迁移过来的应用，需要特别的注意，因为 oracle 并不存在类似的问题，所以在 oracle 的应用中 insert...select...操作非常的常见。例如：有时候会对比较多的纪录进行统计分析，然后将统计的中间结果插入到另外一个表，这样的操作因为进行的非常少，所以可能并没有设置相应的索引。如果迁移到 mysql 数据库后不进行相应的调整，那么在进行这个操作期间，对需要 select 的表实际上是进行的全表扫描导致的所有记录的锁定，将会对应用的其他操作造成非常严重的影响。

究其主要原因，是因为 mysql 在实现复制的机制时和 oracle 是不同的，如果不进行 select 表的锁定，则可能造成从数据库在恢复期间插入结果集的不同，造成主从数据的不一致。如果不采用主从复制，关闭 binlog 并不能避免对 select 纪录的锁定，某些文档中提到可以通过设置 innodb_locks_unsafe_for_binlog 来避免这个现象，当这个参数设置为 true 的时候，将不会对 select 的结果集加锁，但是这样的设置将可能带来非常严重的隐患。如果使用这个 binlog 进行从数据库的恢复，或者进行主数据库的灾难恢复，都将可能和主数据库的执行效果不同。

因此，我们并不推荐通过设置这个参数来避免 insert...select...导致的锁，如果需要进行可能会扫描大量数据的 insert...select 操作，我们推荐使用 select...into outfile 和 load data infile 的组合来实现，这样是不会对纪录进行锁定的。

next-key 锁对并发插入的影响

在行级锁定中，InnoDB 使用一个名为 next-key locking 的算法。InnoDB 以这样一种方式执行行级锁定：当它搜索或扫描表的索引之时，它对遇到的索引记录设置

共享或独占锁定。因此，行级锁定事实上是索引记录锁定。

InnoDB 对索引记录设置的锁定也映像索引记录之前的“间隙”。如果一个用户对一个索引上的记录 R 有共享或独占的锁定，另一个用户 不能紧接在 R 之前以索引的顺序插入一个新索引记录。这个间隙的锁定被执行来防止所谓的“幽灵问题”。

可以用 next-key 锁定在你的应用程序上实现一个唯一性检查：如果你以共享模式读数据，并且没有看到你将要插入的行的重复，则你可以安全地插入你的行，并且知道在读过程中对你的行的继承者设置的 next-key 锁定与此同时阻止任何人对你的行插入一个重复。因此，the next-key 锁定允许你锁住在你的表中并不存在的一些东西。

隔离级别对并发插入的影响

REPEATABLE READ 是 InnoDB 的默认隔离级别。带唯一搜索条件使用唯一索引的 SELECT ... FOR UPDATE, SELECT ... LOCK IN SHARE MODE, UPDATE 和 DELETE 语句只锁定找到的索引记录，而不锁定记录前的间隙。用其它搜索条件，这些操作采用 next-key 锁定，用 next-key 锁定或者间隙锁定锁住搜索的索引范围，并且阻止其它用户的新插入。

在持续读中，有一个与 READ COMMITTED 隔离级别重要的差别：在这个级别，在同一事务内所有持续读读取由第一次读所确定的同一快照。这个惯例意味着如果你在同一事务内发出数个无格式 SELECT 语句，这些 SELECT 语句对相互之间也是持续的。

READ COMMITTED 隔离级别是一个有些象 Oracle 的隔离级别。所有 SELECT ... FOR UPDATE 和 SELECT ... LOCK IN SHARE MOD 语句仅锁定索引记录，而不锁定记录前的间隙，因而允许随意紧挨着已锁定的记录插入新记录。UPDATE 和 DELETE 语句使用一个带唯一搜索条件的唯一的索引仅锁定找到的索引记录，而不包括记录前的间隙。

在范围类型 UPDATE 和 DELETE 语句，InnoDB 必须对范围覆盖的间隙设置 next-key 锁定或间隙锁定以及其它用户做的块插入。这是很必要的，因为要让 MySQL 复制和恢复起作用，“幽灵行”必须被阻止掉。

如果应用是从基于 ORACLE 的应用迁移到 MYSQL 数据库的，那么建议使用该隔离级别提供数据库服务，因为该隔离级别是最接近 ORACLE 的默认隔离级别的，迁移可能遇到的锁问题最小。

如何减少锁冲突

1. 对 Myisam 类型的表：
   1) Myisam 类型的表可以考虑通过改成 Innodb 类型的表来减少锁冲突。
   2) 根据应用的情况，尝试横向拆分成多个表或者改成 Myisam 分区对减少锁冲突也会有一定的帮助。
2. 对 Innodb 类型的表：
   1) 首先要确认，在对表获取行锁的时候，要尽量的使用索引检索纪录，如果没有使用索引访问，那么即便你只是要更新其中的一行纪录，也是全表锁定的。要确保 sql 是使用索引来访问纪录的，必要的时候，请使用 explain 检查 sql 的执行计划，判断是否按照预期使用了索引。
   2) 由于 mysql 的行锁是针对索引加的锁，不是针对纪录加的锁，所以虽然是访问不同行的纪录，但是如果是相同的索引键，是会被加锁的。应用设计的时候也要注意，这里和 Oracle 有比较大的不同。
   3) 当表有多个索引的时候，不同的事务可以使用不同的索引锁定不同的行，当表有主键或者唯一索引的时候，不是必须使用主键或者唯一索引锁定纪录，其他普通索引同样可以用来检索纪录，并只锁定符合条件的行。
   4) 用 SHOW INNODB STATUS 来确定最后一个死锁的原因。查询的结果中，包括死锁的事务的详细信息，包括执行的 SQL 语句的内容，每个线程已经获得了什么锁，在等待什么锁，以及最后是哪个线程被回滚。详细的分析死锁产生的原因，可以通过改进程序有效的避免死锁的产生。
   5) 如果应用并不介意死锁的出现，那么可以在应用中对发现的死锁进行处理。
   6) 确定更合理的事务大小，小事务更少地倾向于冲突。
   7) 如果你正使用锁定读，（SELECT ... FOR UPDATE 或 ... LOCK IN SHARE MODE），试着用更低的隔离级别，比如 READ COMMITTED。
   8) 以固定的顺序访问你的表和行。则事务形成良好定义的查询并且没有死锁。

优化 Mysql Server

查看 Mysql server 当前参数

1. 查看服务器参数默认值：

   mysqld --verbose --help

2. 查看服务器参数实际值：

   shell> mysqladmin variables

   或者

   mysql> SHOW VARIABLES;

3. 查看服务器运行状态值：

   shell> mysqladmin extended-status

   或者

   mysql>SHOW STATUS;

影响 Mysql 性能的重要参数

key_buffer_size 的设置

说明：键缓存(变量 key_buffer_size)被所有线程共享；服务器使用的其它缓存则根据需要分配。此参数只适用于 myisam 存储引擎。

使用方法：

mysql5.1 以前只允许使用一个系统默认的 key_buffer

mysql5.1 以后提供了多个 key_buffer，可以将指定的表索引缓存入指定的 key_buffer，这样可以更小的降低线程之间的竞争，相关语法如下：

例如，下面的语句将表 t1、t2 和 t3 的索引分配给名为 hot_cache 的 键高速缓冲：

mysql> CACHE INDEX t1, t2, t3 IN hot_cache;

可以用 SET GLOBAL 参数设置语句或使用服务器启动选项设置在 CACHE INDEX 语句中

引用的键高速缓冲的大小来创建键高速缓冲。例如：

mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;

要想删除键高速缓冲，将其大小设置为零：

mysql> SET GLOBAL keycache1.key_buffer_size=0;

请注意不能删除默认键高速缓冲。删除默认键高速缓冲的尝试将被忽略

CACHE INDEX 在一个表和 键高速缓冲之间建立一种联系，但每次服务器重启时该联系被丢失。如果你想要每次服务器重启时该联系生效，一个发办法是使用选项文件：包括配置 键高速缓冲的变量设定值，和一个 init-file 选项用来命名包含待执行的 CACHE INDEX 语句的一个文件。例如：

key_buffer_size = 4G

hot_cache.key_buffer_size = 2G

cold_cache.key_buffer_size = 2G

init_file=/path/to/data-directory/mysqld_init.sql

每次服务器启动时执行mysqld_init.sql中的语句。该文件每行应包含一个SQL语句。下面的例子分配几个表，分别对应 hot_cache 和 cold_cache：

CACHE INDEX a.t1, a.t2, b.t3 IN hot_cache

CACHE INDEX a.t4, b.t5, b.t6 IN cold_cache

要想将索引预装到缓存中，使用 LOAD INDEX INTO CACHE 语句。例如，下面的语句可以预装表 t1 和 t2 索引的非叶节点(索引块)：

mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;

键高速缓冲可以通过更新其参数值随时重新构建。例如：

mysql> SET GLOBAL cold_cache。key_buffer_size=4*1024*1024;

如果你很少使用 MyISAM 表，那么也保留低于 16-32MB 的 key_buffer_size 以适应给予磁盘的临时表索引所需。

table_cache 的设置

说明：数据库中打开表的缓存数量。table_cache 与 max_connections 有关。例如，对于 200 个并行运行的连接，应该让表的缓存至少有 200 * N，这里 N 是可以执行的查询的一个联接中表的最大数量。还需要为临时表和文件保留一些额外的文件描述符。

设置技巧：

可以通过检查 mysqld 的状态变量 Opened_tables 确定表缓存是否太小：

```
mysql> SHOW STATUS LIKE 'Opened_tables';
+--------------+-------+
| Variable_name | Value |
+--------------+-------+
| Opened_tables | 2741  |
+--------------+-------+
```

如果值很大，即使你没有发出许多 FLUSH TABLES 语句，也应增加表缓存的大小。

innodb_buffer_pool_size 的设置：

缓存 InnoDB 数据和索引的内存缓冲区的大小。你把这个值设得越高，访问表中数据需要得磁盘 I/O 越少。在一个专用的数据库服务器上，你可以设置这个参数达机器物理内存大小的 80%。尽管如此，还是不要把它设置得太大，因为对物理内存的竞争可能在操作系统上导致内存调度。

innodb_flush_log_at_trx_commit 的设置：

0：日志缓冲每秒一次地被写到日志文件，并且对日志文件做到磁盘操作的刷新，但是在一个事务提交不做任何操作。

1：在每个事务提交时，日志缓冲被写到日志文件，对日志文件做到磁盘操作的刷新。

2：在每个提交，日志缓冲被写到文件，但不对日志文件做到磁盘操作的刷新。对日志文件每秒刷新一次。

默认值是 1，也是最安全的设置，即每个事务提交的时候都会从 log buffer 写到日志文件，而且会实际刷新磁盘，但是这样性能有一定的损失。如果可以容忍在数据库崩溃的时候损失一部分数据，那么设置成 0 或者 2 都会有所改善。设置成 0，则在数据库崩溃的时候会丢失那些没有被写入日志文件的事务，最多丢失 1 秒钟的事务，这种方式是最不安全的，也是效率最高的。设置成 2 的时候，因为只是没有刷新到磁盘，但是已经写入日志文件，所以只要操作系统没有崩溃，那么并没有丢失数据，比设置成 0 更安全一些。

在 mysql 的手册中，为了确保事务的持久性和复制设置的耐受性、一致性，都是建议将这个参数设置为 1 的。

innodb_additional_mem_pool_size：

InnoDB 用来存储数据目录信息和其它内部数据结构的内存池的大小。默认值是 1MB。应用程序里的表越多，你需要在这里分配越多的内存。如果 InnoDB 用光了这个池内的内存，InnoDB 开始从操作系统分配内存，并且往 MySQL 错误日志写警告信息。没有必要给这个缓冲池分配非常大的空间，在应用相对稳定的情况下，这个缓冲池的大小也相对稳定。

innodb_table_locks：

InnoDB 重视 LOCK TABLES，直到所有其它线程已经释放他们所有对表的锁定，MySQL 才从 LOCK TABLE .. WRITE 返回。默认值是 1，这意为 LOCK TABLES 让 InnoDB 内部锁定一个表。在使用 AUTOCOMMIT=1 的应用里，InnoDB 的内部表锁定会导致死锁。可以通过设置 innodb_table_locks=0 来消除这个问题。

innodb_lock_wait_timeout：

Mysql 可以自动的监测行锁导致的死锁并进行相应的处理，但是对于表锁导致的死锁不能自动的监测，所以该参数主要被用来在出现类似情况的时候对锁定进行的后续处理。默认值是 50 秒，根据应用的需要进行调整。

innodb_support_xa：

通过该参数设置是否支持分布式事务，默认值是 ON 或者 1，表示支持分布式事务。如果确认应用中不需要使用分布式事务，则可以关闭这个参数，减少磁盘刷新的次数并获得更好的 InnoDB 性能。

innodb_doublewrite：

默认地，InnoDB 存储所有数据两次，第一次存储到 doublewrite 缓冲，然后存储到确实的数据文件。如果对性能的要求高于对数据完整性的要求，那么可以通过 --skip-innodb-doublewrite 关闭这个设置。

innodb_log_buffer_size:

默认的设置在中等强度写入负载以及较短事务的情况下，服务器性能还可以。如果存在更新操作峰值或者负载较大，就应该考虑加大它的值了。如果它的值设置太高了，可能会浪费内存 -- 它每秒都会刷新一次，因此无需设置超过 1 秒所需的内存空间。通常 8-16MB 就足够了。越小的系统它的值越小。

innodb_log_file_size：

在高写入负载尤其是大数据集的情况下很重要。这个值越大则性能相对越高，但是要注意到可能会增加恢复时间。

I/O 问题

磁盘搜索是巨大的性能瓶颈。当数据量变得非常大以致于缓存性能变得不可能有效时，该问题变得更加明显。对于大数据库，其中你或多或少地随机访问数据，你可以确信对读取操作需要至少一次硬盘搜索，写操作需要多次硬盘搜索。要想使该问题最小化，应使用搜索次数较少的磁盘。

使用磁盘阵列或虚拟文件卷分布 I/O

分条意味着你有许多磁盘，将第 1 个块放到第 1 个硬盘，第 2 个块放到第 2 个磁盘，并且第 N 块在（N mod number_of_disks）磁盘上等等。这意味着如果正常数据大小小于分条大小（或完全匹配），能够得到最佳性能。分条完全取决于操作系统和分条大小，因此用不同的条纹大小对应用程序进行基准测试。

分条的不同速度完全依赖于参数。依赖于怎样设置条纹参数和硬盘数量，可以根据不同数量级别得到不同的标准。你必须进行选择以便优化随机或顺序存取。

1. 为了高可靠性你可能想使用 RAID 0+1（条纹加镜像），但在这种情况下，需要 2*N 块磁盘来保持 N 个磁盘的数据。如果你肯为它花钱，这可能是最好的选项。然而，你可能还必须投资一部分资金到卷管理软件中以便有效地管理它。
2. 一个较好的选择是根据数据类型的重要性程度改变 RAID 级别。例如，保存可以在 RAID 0 硬盘上重新生成的不太重要的数据，但保存 真正重要数据（例如主机信息和日志）到 RAID 0+1 或 RAID N 硬盘。如果你有许多写操作，RAID N 可能会存在问题，因为需要时间来更新校验位。

使用 Symbolic Links 分布 I/O

可以将表和数据库从数据库目录移动到其它的位置并且用指向新位置的符号链接进行替换。推荐的方法只需要将数据库通过符号链接指到不同的磁盘。符号链接表仅作为是最后的办法。

符号链接一个数据库的方法是，首先在一些有空闲空间的硬盘上创建一个目录，然后从 MySQL 数据目录中创建它的一个符号链接。

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test /path/to/datadir
```

注意：只有 MyISAM 表完全支持符号链接。对于其它表类型，如果试图在操作系统中的文件上用前面的任何语句使用符号链接，可能会出现奇怪的问题。

对于 MyISAM 表的符号链接的处理如下：

1. 在数据目录指，一定会有表定义文件、数据文件和索引文件。数据文件和索引文件可以移到别处和在数据目录中符号链接替代。表定义文件不能进行符号链接替换。
2. 可以分别通过符号链接将数据文件和索引文件指到不同的目录。

3. 如果 mysqld 没有运行，符号链接可以从服务器命令行使用 ln -s 手动完成。同样，通过使用 DATA DIRECTORY 和 INDEX DIRECTORY 选项创建表，你可以指示运行的 MySQL 服务器执行符号链接。
4. myisamchk 不用数据文件或索引文件替换符号链接。它直接工作在符号链接指向的文件。任何临时文件创建在数据文件或索引文件所处的目录中。
5. 注释：当你删掉一个表时，如果该表使用了符号链接，符号链接和该符号链接指向的文件都被删除掉。这就是你不应以系统 root 用户运行 mysqld 或允许系统用户对 MySQL 数据库目录有写访问权限的原因。
6. 如果你用 ALTER TABLE ... RENAME 重命名一个表并且不将表移到另一个数据库，数据库目录中的符号链接被重新命名为一个新名字并且数据文件和索引文件也相应地重新命名。
7. 如果你用 ALTER TABLE ... RENAME 移动一个表到另一个数据库，表移动到另一个数据库目录。旧的符号链接和其所指向的文件被删除。换句话说，新表不再被链接。
8. 如果不使用符号链接，你应对 mysqld 使用--skip-symbolic-links 选项以确保没有人能够使用 mysqld 来删除或重新命名数据目录之外的文件。

表符号链接还不支持以下操作：

1. ALTER TABLE 忽略 DATA DIRECTORY 和 INDEX DIRECTORY 表选项。
2. BACKUP TABLE 和 RESTORE TABLE 不考虑符号链接。
3. .frm 文件必须绝不能是一个符号链接（如前面所述，只有数据和索引文件可以是符号链接）。如果试图这样做（例如，生成符号链接）会产生不正确的结果。

应用优化

使用连接池

对于访问数据库来说，建立连接的代价比较昂贵，因此，我们有必要建立"连接池"以提高访问的性能。我们可以把连接当作对象或者设备，池中又有许多已经建立的连接，访问本来需要与数据库的连接的地方，都改为和池相连，池临时分配连接供访问使用，结果返回后，访问将连接交还。

减少对 Mysql 的访问

避免对同一数据做重复检索：

应用中需要理清楚对数据库的访问逻辑，需要对相同表的访问，尽量集中在相同 sql 访问，一次提取结果，减少对数据库的重复访问。

使用 mysql query cache：

作用：查询缓存存储 SELECT 查询的文本以及发送给客户端的相应结果。如果随后收到一个相同的查询，服务器从查询缓存中重新得到查询结果，而不再需要解析和执行查询。

适用范围：不发生数据更新的表。当表更改（包括表结构和表数据）后，查询缓存值的相关条目被清空。

查询缓存的主要参数设置：

show variables like ‘%query_cache%’;

have_query_cache 表明服务器在安装使已经配置了高速缓存

query_cache_size 表明缓存区大小，单位为 M

query_cache_type 的变量值从 0 到 2，含义分别为

0 或者 off（缓存关闭）

1 或者 on（缓存打开，使用 sql_no_cache 的 select 除外）

2 或者 demand（只有带 sql_cache 的 select 语句提供高速缓存）

在 SHOW STATUS 中，你可以监视查询缓存的性能：

变量	**含义**
Qcache_queries_in_cache	在缓存中已注册的查询数目
Qcache_inserts	被加入到缓存中的查询数目
Qcache_hits	缓存采样数数目
Qcache_lowmem_prunes	因为缺少内存而被从缓存中删除的查询数目
Qcache_not_cached	没有被缓存的查询数目（不能被缓存的，或由于 QUERY_CACHE_TYPE)
Qcache_free_memory	查询缓存的空闲内存总数
Qcache_free_blocks	查询缓存中的空闲内存块的数目

Qcache_total_blocks	查询缓存中的块的总数目

加 cache 层：

Cache（高速缓存）、Memory（内存）、Hard disk（硬盘）都是数据存取单元，但存取速度却有很大差异，呈依次递减的顺序。对于 CPU 来说，它可以从距离自己最近的 Cache 高速地存取数据，而不是从内存和硬盘以低几个数量级的速度来存取数据。而 Cache 中所存储的数据，往往是 CPU 要反复存取的数据，有特定的机制（或程序）来保证 Cache 内数据的命中率（Hit Rate）。因此，CPU 存取数据的速度在应用高速缓存后得到了巨大的提高。

因为将数据写入高速缓存的任务由 Cache Manager 负责，所以对用户来说高速缓存的内容肯定是只读的。需要你做的工作很少，程序中的 SQL 语句和直接访问 DBMS 时没有分别，返回的结果也看不出有什么差别。而数据库厂商往往会在 DB Server 的配置文件中提供与 Cache 相关的参数，通过修改它们，可针对我们的应用优化 Cache 的管理。

负载均衡

利用 mysql 复制分流查询操作：

利用 mysql 的主从复制可以有效的分流更新操作和查询操作，具体的实现是一个主服务器，承担更新操作，多台从服务器，承担查询操作，主从之间通过复制实现数据的同步。多台从服务器一方面用来确保可用性，一方面可以创建不同的索引满足不同查询的需要。

对于主从之间不需要复制全部表的情况，可以通过在主的服务器上搭建一个虚拟的从服务器，将需要复制到从服务器的表设置成 blackhole 引擎，然后定义 replicate-do-table 参数只复制这些表，这样就过滤出需要复制的 binlog，减少了传输 binlog 的带宽。因为搭建的虚拟的从服务器只起到过滤 binlog 的作用，并没有实际纪录任何数据，所以对主数据库服务器的性能影响也非常的有限。

通过复制分流查询的存在的问题是主数据库上更新频繁或者网络出现问题的时候，主从之间的数据可能存在差异，造成查询结果的异议，应用在设计的时候需要有所考虑。

采用分布式数据库架构：

mysql 从 5.0.3 开始支持分布式事务，当前分布式事务只对 Innodb 存储引擎支持。分布式的数据库架构适合大数据量，负载高的情况，有良好的扩展性和高可用性。通过在多

台服务器之间分布数据实现在多台服务器之间的负载平均，提高了访问的执行效率。具体实现的时候，可以使用 mysql 的 Cluster 功能（NDB 引擎）或者自己编写程序来实现全局事务。

第三篇 管理维护篇

mysql 安装升级

安装

安装方法比较

	rpm	二进制	源码
优点	安装简单，适合初学者学习使用	1. 安装简单 2. 可以地安装到任何路径下，灵活性好 3. 一台服务器可以安装多个 mysql	1. 在实际安装的操作系统进行可根据需要定制编译，最灵活 2. 性能最好 3. 一台服务器可以安装多个 mysql
缺点	1. 需要单独下载客户端和服务器 2. 安装路径不灵活，默认路径不能修改，一台服务器只能安装一个 mysql	1) 已经经过编译，性能不如源码编译的好 2) 不能灵活定制编译参数	1. 安装过程较复杂 2. 编译时间长
文件布局	/usr/bin 客户端程序和脚本 /usr/sbin mysqld 服务器 /var/lib/mysql 日志文件，数据库	bin 客户端程序和 mysqld 服务器 data 日志文件，数据库 docs	bin 客户端程序和脚本 include/mysql 包含(头)文件 info Info 格式的文档

	/usr/share/doc/packag es 文档 /usr/include/mysql 包含(头)文件 /usr/lib/mysql 库 /usr/share/mysql 错误消息和字符集文件 /usr/share/sql-bench 基准程序	文档，ChangeLog include 包含(头)文件 lib 库 scripts mysql_install_db share/mysql 错误消息 文件 sql-bench 基准程序	lib/mysql 库 libexec mysqld 服务器 share/mysql 错误消息文件 sql-bench 基准程序和 crash-me 测 试 var 数据库和日志文件

rpm 安装步骤

大多数情况下，下载 MySQL-server 和 MySQL-client 就够用了，安装方法如下：

```
shell> rpm -ivh MySQL-server-VERSION.i386.rpm
shell> rpm -ivh MySQL-client-VERSION.i386.rpm
```

二进制安装步骤

root 登陆，执行如下步骤：

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> cd /home/mysql
shell>tar -xzvf  /home/mysql/mysql-VERSION-OS.tar.gz
shell> ln -s mysql-VERSION-OS.tar.gz mysql
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root:mysql  .
shell> chown -R mysql:mysql data
shell> bin/mysqld_safe --user=mysql &
```

源码安装步骤

root 登陆，执行如下步骤：

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> gunzip < mysql-VERSION.tar.gz | tar -xvf -
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> cd /usr/local/mysql
shell> bin/mysql_install_db --user=mysql
shell> chown -R root  .
shell> chown -R mysql var
shell> chgrp -R mysql .
bin/mysql_install_db --user=mysql
```

源码安装的性能考虑：

去掉不需要的模块：

源码安装由于可以灵活的进行数据库的定制编译，因此有更强的灵活性。某些编译选项可以大大增强我们数据库的性能。

执行如下命令可以看到所有编译的配置选项：

```
shell> ./configure --help
```

如果只安装客户端，可以执行如下命令：

```
shell> ./configure --without-server
```

如果你不想要位于“/usr/local/var”目录下面的日志(log)文件和数据库，使用类似于下列 configure 命令的一个：

shell>./configure--prefix=/usr/local/mysql

shell>./configure--prefix=/usr/local localstatedir=/usr/local/mysql/data

第一个命令改变安装前缀以便将所有内容安装到“/usr/local/mysql”下面而非默认的“/usr/local”。第二个命令保留默认安装前缀，但是覆盖了数据库目录默认目录(通常是“/usr/local/var”)并且把它改为/usr/local/mysql/data。编译完 MySQL 后，可以通过选项文件更改这些选项

修改 socket 的默认位置：

shell> ./configure\-- with-unix-socket-

path=/usr/local/mysql/tmp/mysql.sock

只选择要使用的字符集：

改变安装后的默认字符集和排序规则：

shell> ./configure -- with-charset=*CHARSET*

./configure --with-collation=*COLLATION*

选择需要安装的字符集：

shell>./configure --with-extra-charsets=*LIST*

list 可以是下面任何一项：

空格间隔的一系列字符集名

complex -，以包括不能动态装载的所有字符集

all -，以将所有字符集包括进二进制

使用 pgcc 编译：

pgcc 2.90.29 或更新版：

CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc \

CXXFLAGS="-O3 -mpentiumpro -mstack-align-double \

-felide-constructors -fno- exceptions -fno-rtti"

使用静态编译以提高性能：

shell>./configure --with-client-ldflags=-all-static\

--with-mysqld-ldflags=-all-static

mysql 升级

方法 1 最简单，适合于任何存储引擎（不一定速度最快）

安装新数据库

将老数据库导出为文本，导入到新数据库上

shell> mysqladmin -h hostname -P port -u user -p passwd create db_name

shell> mysqldump --opt db_name | mysql -h hostname -P port -u user -p passwd db_name

注：如果网络较慢，可以在导出选项中加上--compress 来减少网络传输

升级权限表

将原库中的 mysql 数据库目录全部 cp 过来覆盖新库中 mysql 数据库

在 shell 里面执行 mysql_fix_privilege_tables 命令升级权限表

shell>mysql_fix_privilege_tables

重启数据库服务

方法 2 适合于任何存储引擎，速度较快

安装新数据库

旧库中创建保存输出文件的目录并备份数据库：

shell> mkdir DUMPDIR

shell>mysqldump --tab=DUMPDIR db_name

将 DUMPDIR 目录中的文件转移到目标机上相应的目录中并将文件装载到 MySQL：

shell> mysqladmin create db_name # create database

shell> cat DUMPDIR/*.sql | mysql db_name # create tables in database

shell> mysqlimport db_name DUMPDIR/*.txt # load data into tables

（实际测试的时候，发现 txt 要放到 data 下才能执行，否则提示文件找不到）

升级权限表

将原库中的 mysql 数据库目录全部 cp 过来覆盖新库中 mysql 数据库

在 shell 里面执行 mysql_fix_privilege_tables 命令升级权限表

shell>mysql_fix_privilege_tables

重启数据库服务

方法 3 适合于 myisam 表，速度最快

安装新数据库

将原库中的数据目录下的所有文件（.frm，.MYD，MYI）cp 到新库下的相应目录下

升级权限表

将原库中的 mysql 数据库目录全部 cp 过来覆盖新库中 mysql 数据库

在 shell 里面执行 mysql_fix_privilege_tables 命令升级权限表

shell>mysql_fix_privilege_tables

flush tables 或者重启数据库服务生效

mysql 降级

对于 myisam 存储引擎，直接将数据文件 cp 到低版本数据库上的数据目录下

如果发生表格式冲突，或者是其他存储引擎的表，用 mysqldump 导出文本后导入低版本的数据库

Mysql 日志管理

错误日志：

记录内容：

包含了当**mysqld**启动和停止时，以及服务器在运行过程中发生任何严重错误时的相关信息

文件位置和格式：

可以用--log-error[=*file_name*]选项来指定**mysqld**保存错误日志文件的位置。如果没有给定*file_name*值，**mysqld**使用错误日志名*host_name.err* 并在数据目录中写入日志文件

BINLOG：

记录内容：

二进制日志包含了所有更新了数据或者已经潜在更新了数据（例如，没有匹配任何行的一个DELETE）的所有语句。语句以“事件”的形式保存，它描述数据更改

文件位置和格式：

当用--log-bin[=*file_name*]选项启动时，**mysqld**写入包含所有更新数据的SQL命令的日志文件。如果未给出*file_name*值， 默认名为-bin后面所跟的主机名。如果给出了文件名，但没有包含路径，则文件被写入数据目录

查看binlog内容：

shell> **mysqlbinlog log-file**

删除日志：

RESET MASTER;//删除所有binlog日志，新日志编号从头开始

PURGE MASTER LOGS TO 'mysql-bin.010';//删除mysql-bin.010之前所有日志

PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26'

;// 删除2003-04-02 22:46:26之前产生的所有日志

相关选项：

1. --binlog-do-db=*db_name*

   告诉主服务器，如果当前的数据库(即USE选定的数据库)是*db_name*，应将更新记录到二进制日志中。其它所有没有明显指定的数据库 被忽略

2. --binlog-ignore-db=*db_name*

   告诉主服务器，如果当前的数据库(即USE选定的数据库)是*db_name*，不应将更新保存到二进制日志中

   要想记录或忽视多个数据库，使用多个选项，为每个数据库指定相应的选项。

3. -innodb-safe-binlog

使用此选项和sync_binlog＝N（每写N次日志同步磁盘）全局变量将使得事务能够记录的更加安全

4. 具有SUPER权限的客户端可以通过SET SQL_LOG_BIN=0语句禁止将自己的语句记入二进制记录

查询日志

记录内容：

记录了客户端的所有语句，而二进制日志不包含只查询数据的语句

文件位置和格式：

用--log[=*file_name*]或-l [*file_name*]选项启动它。如果没有给定*file_name*的值，默认名是*host_name*.log。

慢查询日志：

记录内容：

记录包含所有执行时间超过long_query_time秒的SQL语句的日志文件。获得初使表锁定的时间不算作执行时间。

文件位置和格式

用--log-slow-queries[=*file_name*]选项启动它。如果没有给出*file_name*值， 默认为主机名，后缀为-slow.log。如果给出了文件名，但不是绝对路径名，文件则写入数据目录。

快速查看：

使用 **mysqldumpslow** 命令获得日志中显示的查询摘要来处理慢查询日志，例如：

[zzx@bj37 data]$ mysqldumpslow bj37-slow.log

其他选项：

在MySQL 5.1中,通过--log-slow-admin-statements服务器选项，你可以请求将慢管理语句，例如OPTIMIZE TABLE、ANALYZE TABLE和 ALTER TABLE写入慢查询日志

数据备份与恢复：

备份/恢复策略：

1. 要定期做 mysql 备份，并考虑系统可以承受的恢复时间。
2. 确保 mysql 打开 log-bin，有了 binarylog，mysql 才可以在必要的时候做完整恢复，或基于时间点的恢复，或基于位置的恢复。
3. 要经常做备份恢复测试，确保备份是有效的，并且是可以恢复的。

冷备份：

备份：

1. 停掉 mysql 服务，在操作系统级别备份 mysql 的数据文件。
2. 重启 mysql 服务，备份重启以后生成的 binlog。

恢复：

1. 停掉 mysql 服务，在操作系统级别恢复 mysql 的数据文件。
2. 重启 mysql 服务，使用 mysqlbinlog 恢复自备份以来的 binlog。

逻辑备份：

备份：

1. 选择在系统空闲时，比如在夜间，使用 mysqldump -F(flush-logs)备份数据库。

   ```
   mysqldump -u root -p*** pointcard -F > pointcard.sql
   ```

2. 并备份 mysqldump 开始以后生成的 binlog。

恢复：

1. 停掉应用，执行 mysql 导入备份文件.

   ```
   mysql -u root -p*** pointcard < pointcard.sql
   ```

2. 使用 mysqlbinlog 恢复自 mysqldump 备份以来的 binlog。

   ```
   mysqlbinlog $HOME/data/mysql-bin.123456 | mysql -u root -p***
   ```

单个表的备份：

备份：

1. 方法 1：

mysql > select * into outfile '/tmp/order_tab' fields-terminated-by=',' from order_tab;

2. 方法 2：

mysqldump -u root -p*** -T /tmp pointcard order_tab --fields-terminated-by=',';

恢复：

1. 方法 1：

mysql > load data [local] infile '/tmp/order_tab' into table order_tab fields-terminated-by=',';

2. 方法 2：

mysqlimport -u root -p*** [--local] pointcatd order_tab.txt --fields-terminated-by=',';

注意：如果导入和导出是跨平台操作的（windows 和 linux），那么要注意设置参数 line-terminated-by， windows 上设置为 line-terminated-by='\r\n'，linux 上设置为 line-terminated-by='\n'.

使用备份工具 ibbackup：

ibbackup 是 innodb 公司（www.innodb.com)的一个热备份工具，专门对 innodb 存储引擎进行物理热备份，此工具是收费的，不能免费使用。现在 innodb 公司已经被 oracle 收购

使用方法：

编辑用于启动的配置文件my.cnf和用于备份的配置文件my2.cnf

my.cnf的例子如下：

[mysqld]

datadir = /home/heikki/data

innodb_data_home_dir = /home/heikki/data

innodb_data_file_path = ibdata1:10M:autoextend

innodb_log_group_home_dir = /home/heikki/data

set-variable = innodb_log_files_in_group=2

set-variable = innodb_log_file_size=20M

如果想备份到/home/heikki/backup，则my2.cnf的例子如下：

[mysqld]

datadir = /home/heikki/backup

innodb_data_home_dir = /home/heikki/backup

innodb_data_file_path = ibdata1:10M:autoextend

innodb_log_group_home_dir = /home/heikki/backup

set-variable = innodb_log_files_in_group=2

set-variable = innodb_log_file_size=20M

开始备份

ibbackup my.cnf my2.cnf

如果需要恢复，则进行日志重做

ibbackup --apply-log my2.cnf

恢复后重启数据库服务

./bin/mysqld_saft --defaults-file=my2.cnf &

时间点恢复：

1. 如果上午 10 点发生了误操作，可以用以下语句用备份和 binglog 将数据恢复到故障前：

mysqlbinlog --stop-date="2005-04-20 9:59:59" /var/log/mysql/bin.123456 | mysql -u *root* -p*mypwd*

2. 跳过故障时的时间点，继续执行后面的 binlog，完成恢复

```
mysqlbinlog --start-date="2005-04-20 10:01:00"
/var/log/mysql/bin.123456| mysql -u root -pmypwd \
```

位置恢复：

和时间点恢复类似，但是更精确，步骤如下：

mysqlbinlog --start-date="2005-04-20 9:55:00" --stop-date="2005-04-20 10:05:00" /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql

该命令将在/tmp 目录创建小的文本文件，编辑此文件，找到出错语句前后的位置号，例如前后位置号分别是 368312 和 368315。恢复了以前的备份文件后，你应从命令行输入下面内容：

```
mysqlbinlog --stop-position="368312" /var/log/mysql/bin.123456 \
    | mysql -u root -pmypwd
 mysqlbinlog --start-position="368315" /var/log/mysql/bin.123456 \
    | mysql -u root -pmypwd \
```

上面的第 1 行将恢复到停止位置为止的所有事务。下一行将恢复从给定的起始位置直到二进制日志结束的所有事务。因为 **mysqlbinlog** 的输出包括每个 SQL 语句记录之前的 SET TIMESTAMP 语句，恢复的数据和相关 MySQL 日志将反应事务执行的原时间。

MyISAM 表修复：

一张损坏的表的症状通常是查询意外中断并且能看到下述错误：

- “*tbl_name*.frm”被锁定不能更改。
- 不能找到文件“*tbl_name*.MYI”（Errcode：*nnn*)。
- 文件意外结束。
- 记录文件被毁坏。
- 从表处理器得到错误*nnn*

解决方法如下：

方法一：

myisamchk -r tablename

上面的方法几乎能解决所有问题，如果不行，则使用：

myisamchk -o tablename

方法二：

1) CHECK TABLE *tbl_name* [, *tbl_name*] ... [*option*] ...

option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}

2) REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE

tbl_name [, *tbl_name*] ... [QUICK] [EXTENDED] [USE_FRM]

Mysql 安全：

正确设置目录权限：

设置目录权限的原则是软件和数据分开，具体如下：

1. 将 mysql 安装在单独的用户下
2. 安装时，以 root 用户进行安装，mysql 的软件默认都为 root 权限
3. 安装完毕后，将数据目录权限设置为实际运行 mysql 的用户权限，比如：

Chown -R mysql:mysql /home/mysql/data

尽量避免以 root 权限运行 mysql：

将 4.1 的目录权限设置完毕后，启动、停止 mysql 以及日常的维护工作都可以在 mysql 用户下进行,没有必要 su 到 root 后再用—user=mysql 来启动和关闭 mysql，这样就没有必要授权维护人员 root 权限，而且最重要的一定是因为任何具有 FILE 权限的用户能够用 root 创建文件。

删除匿名帐号：

安装完毕 mysql 后，会自动安装一个空帐号，普通用户只需要执行 mysql 命令即可登陆 mysql，给系统造成隐患，建议删除此空帐号：

drop user ''@'localhost';

drop user ''@' localhost.localdomain' ;

给 mysql root 帐号设置口令：

Mysql 安装完毕后，root 默认口令为空，需要马上修改 root 口令：

[zzx@localhost data]$ mysql -uroot

mysql> set password=password('123');

Query OK, 0 rows affected (0.00 sec)

设置安全密码并定期修改：

尽量使用安全密码，建议使用 6 位以上字母、数字、下画线和一些特殊字符组合而成的字符串

只授予帐号必须的权限：

只需要赋予普通用户必须的权限，比如：

Grant select,insert,update,delete on tablename to ‘username’@’hostname’ ;

除 root 外，任何用户不应有 mysql 库 user 表的存取权限：

如果拥有 mysql 库中 user 表的存取权限（select、update、insert、delete），就可以轻易的增加、修改、删除其他的用户权限，造成系统的安全隐患。

不要把 FILE、PROCESS 或 SUPER 权限授予管理员以外的帐号：

FILE 权限可以被滥用于将服务器主机上 MySQL 能读取的任何文件读入到数据库表中。包括任何人可读的文件和服务器数据目录中的文件。可以使用 SELECT 访问数据库表，然后将其内容传输到客户端上。

不要向非管理用户授予 FILE 权限。有这权限的任何用户能在拥有 **mysqld** 守护进程权限的文件系统那里写一个文件！为了更加安全，由 SELECT ... INTO OUTFILE 生成的所有文件对每个人是可写的，并且你不能覆盖已经存在的文件。

file 权限也可以被用来读取任何作为运行服务器的 Unix 用户可读取或访问的文件。使用该权限，你可以将任何文件读入数据库表。这可能被滥用，例如，通过使用 LOAD DATA 装载“/etc/passwd”进一个数据库表，然后能用 SELECT 显示它。

PROCESS 权限能被用来察看当前执行的查询的明文文本，包括设定或改变密码的查询。

SUPER 权限能用来终止其它用户或更改服务器的操作方式。比如 kill 进程

不要将 PROCESS 或 SUPER 权限授给非管理用户。**mysqladmin processlist** 的输出显示出当前执行的查询正文，如果另外的用户发出一个 UPDATE user SET password=PASSWORD('not_secure')查询，被允许执行那个命令的任何用户可能看得到

load data local 带来的安全问题：

由 MySQL 服务器启动文件从客户端向服务器主机的传输。理论上，打过补丁的服务器可以告诉客户端程序传输服务器选择的文件，而不是客户用 LOAD DATA 语句指定的文件。这样服务器可以访问客户端上客户有读访问权限的任何文件。

在 Web 环境中，客户从 Web 服务器连接，用户可以使用 LOAD DATA LOCAL 来读取 Web 服务器进程有读访问权限的任何文件(假定用户可以运行 SQL 服务器的任何命令)。在这种环境中，MySQL 服务器的客户实际上是 Web 服务器，而不是连接 Web 服务器的用户运行的程序。

解决方法：

可以用--local-infile=0 选项启动 **mysqld** 从服务器端禁用所有 LOAD DATA LOCAL 命令。

对于 **mysql** 命令行客户端，可以通过指定--local-infile[=1]选项启用 LOAD DATA LOCAL,或通过--local-infile=0 选项禁用。类似地,对于**mysqlimport**,--local

or -L 选项启用本地数据文件装载。在任何情况下，成功进行本地装载需要服务器启用相关选项。

尽量避免通过 symlinks 访问表：

不要允许使用表的符号链接。(可以用--skip-symbolic-links 选项禁用)。如果你用 root 运行 **mysqld** 则特别重要，因为任何对服务器的数据目录有写访问权限的人则能够删除系统中的任何文件！

使用 merge 存储引擎潜藏的安全漏洞：

Merge 表在某些版本中可能存在以下安全漏洞：

用户 A 赋予表 T 的权限给用户 B

用户 B 创建一个包含 T 的 merge 表，做各种操作

用户 A 收回对 T 的权限

安全隐患：用户 B 通过 merge 表仍然可以访问表 A 中的数据

防止 DNS 欺骗：

如果你不信任你的 DNS，你应该在授权表中使用 IP 数字而不是主机名。在任何情况下，你应该非常小心地使用包含通配符的主机名来创建 授权表条目！

drop table 命令并不收回以前的相关访问授权：

drop 表的时候，其他用户对此表的权限并没有被收回，这样导致重新创建同名的表时，以前其他用户对此表的权限会自动赋予，导致权限外流。

因此，要在删除表时，同时取消其他用户在此表上的相应权限。

使用 SSL：

下面列出了规定 SSL、证书文件和密钥文件使用的选项。它们可以位于命令行中或选项文件中。

- --ssl

对于服务器，该选项规定该服务器允许 SSL 连接。对于客户端程序，它允许客户使用 SSL 连接服务器。单单该选项不足以使用 SSL 连接。还必须指定--ssl-ca、--ssl-cert 和--ssl-key 选项。

通常从反向使用该选项表示不应使用 SSL。要想实现，将选项指定为--skip-ssl 或--ssl=0。

请注意使用--ssl 不*需要*SSL 连接。例如，如果编译的服务器或客户不支持 SSL，则使用普通的未加密的连接。

确保使用 SSL 连接的安全方式是使用含 REQUIRE SSL 子句的 GRANT 语句在服务器上创建一个账户。然后使用该账户来连接服务器，服务器和客户端均应启用 SSL 支持。

- --ssl-ca=*file_name*

含可信 SSL CA 的清单的文件的路径。

- --ssl-capath=*directory_name*

包含 pem 格式的可信 SSL CA 证书的目录的路径。

- --ssl-cert=*file_name*

SSL 证书文件名，用于建立安全连接。

- --ssl-cipher=*cipher_list*

允许的用于 SSL 加密的密码的清单。*cipher_list* 的格式与 OpenSSL ciphers 命令相同。

示例：--ssl-cipher=ALL:-AES:-EXP

- --ssl-key=*file_name*

SSL 密钥文件名，用于建立安全连接。

如果可能，给所有用户加上访问 IP 限制：

给所有用户加上 ip 限制将拒绝所有未知的主机进行的连接，保证只有受信任的主机才可以进行连接。例如：

Grant select on dbname.* to ‘username’@’ip’ identified by ‘passwd’;

严格控制操作系统帐号和权限：

在数据库服务器上要严格控制操作系统的帐号和权限，比如：

锁定 mysql 用户

其他任何用户都采取独立的帐号登陆，管理员通过普通用户管理 mysql；或者通过 root su 到 mysql 用户下进行管理。

禁止修改 mysql 用户下的任何资源

增加防火墙：

购买防火墙。这样可以保护你防范各种软件中至少 50%的各种类型的攻击。把 MySQL 放到防火墙后或隔离区(DMZ)。

其他安全设置选项：

allow-suspicious-udfs：

该选项控制是否可以载入主函数只有 xxx 符的用户定义函数。默认情况下，该选项被关闭，并且只能载入至少有辅助符的 UDF。这样可以防止从未包含合法 UDF 的共享对象文件载入函数。

old-passwords：

强制服务器为新密码生成短(pre-4.1)密码哈希。当服务器必须支持旧版本客户端程序时，为了保证兼容性这很有用。

safe-user-create：

如果启用，用户不能用 GRANT 语句创建新用户，除非用户有 mysql.user 表的 INSERT 权限。如果你想让用户具有授权权限来创建新用户，你应给用户授予下面的权限：

mysql> GRANT INSERT(user) ON mysql.user TO '*user_name*'@'*host_name*';

这样确保用户不能直接更改权限列，必须使用 GRANT 语句给其它用户授予该权限。

secure-auth：

不允许鉴定有旧(pre-4.1)密码的账户。

skip-grant-tables：

这个选项导致服务器根本不使用权限系统。这给每个人以*完全访问*所有的数据库的权力！（通过执行 **mysqladmin flush-privileges** 或 **mysqladmin reload** 命令，或执行 FLUSH PRIVILEGES 语句，你能告诉一个正在运行的服务器再次开始使用授权表。）

skip-networking：

在网络上不允许 TCP/IP 连接。所有到 mysqld 的连接必须经由 Unix 套接字进行

skip-show-database：

使用该选项，只允许有 SHOW DATABASES 权限的用户执行 SHOW DATABASES 语句，该语句显示所有数据库名。不使用该选项，允许所有用户执行 SHOW DATABASES，但只显示用户有 SHOW DATABASES 权限或部分数据库权限的数据库名。请注意全局权限指数据库的权限。

Mysql 复制：

Mysql 复制概述：

复制是将主数据库的 DML 操作通过日志传到从服务器上，使得从服务器实现了对主服务器的远程备份，并且可以通过应用使得在主服务器繁忙的时候分担一部分负载。mysql 支持同时向多台从服务器进行复制。

缺点：不能保证主从同步，只能实现异步复制。

安装配置：

1. 正常安装主从服务器。确保主服务器开启 log-bin，主从服务器的 server_id 取不同的数字。

2. 在主服务器上，设置一个复制账户，并授予 REPLICATION SLAVE 权限：

mysql > GRANT REPLICATION rep ON *.* TO 'rep'@'slave_host' IDENTIFIED BY 'repl_pass';

3. 在主服务器上，设置读锁定有效：

mysql > FLUSH TABLES WITH READ LOCK;

然后得到主服务器上当前的二进制日志名和偏移量值：

```
mysql> SHOW MASTER STATUS;
+------------------+-----------+-----------------+----------------
-------+
| File             | Position  | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+-----------+-----------------+----------------
-------+
| bj72-bin.000013 | 27050310 |              |                  |
+------------------+-----------+-----------------+----------------
-------+
```

4. 在从服务器上，做相应设置：

```
mysql> CHANGE MASTER TO
->     MASTER_HOST='master_host',
```

```
->      MASTER_PORT=master_port,
->      MASTER_USER='rep',
->      MASTER_PASSWORD=' rep_pass ',
->      MASTER_LOG_FILE=' bj72-bin.000013',
->      MASTER_LOG_POS=27050310;
```

5. 在从服务器上，启动 slave 线程：

   mysql> START SLAVE;

   这时 slave 上执行 show processlist 命令将显示类似如下进程：

   10436 | system user 。。。|Waiting for master to send event | NULL

   这表明 slave 已经连接上 master，并开始接受并执行日志。

6. 在主服务器上，重置读锁定：

   mysql> UNLOCK TABLES;

日常管理维护：

经常查看 slave 状态

```
mysql> SHOW SLAVE STATUS\G;
                。。。。。
          Slave_IO_Running: Yes
         Slave_SQL_Running: Yes
           。。。。。。
1 row in set (0.00 sec)
```

主要检查 Slave_IO_Running 和 Slave_SQL_Running 这两个进程状态是否是 yes，这两个进程的含义如下：

Slave_IO_Running：此进程负责 slave 从 master 服务器上读取 binlog 日志，并写入 slave 服务器上的中继日志中

Slave_SQL_Running：此进程负责读取并且执行中继日志中的 binlog 日志

只要其中有一个进程的状态是 no，则表示复制进程停止，错误原因可以从 Last_Errno 后面看到

怎样强制主服务器阻塞更新直到从服务器同步？

可以使用下面的步骤：

1. 在主服务器上，执行这些语句：

   mysql> FLUSH TABLES WITH READ LOCK;

   mysql> SHOW MASTER STATUS;

   记录 SHOW 语句的输出的日志名和偏移量。这些是复制坐标。

2. 在从服务器上，发出下面的语句，其中 Master_POS_WAIT()函数的参量是前面步骤中的得到的复制坐标值：

   mysql> SELECT MASTER_POS_WAIT('log_name', log_offset);

   SELECT 语句阻塞直到从服务器达到指定的日志文件和偏移量。此时，从服务器与主服务器同步，语句返回。

3. 在主服务器上，发出下面的语句允许主服务器重新开始处理更新：

   mysql> UNLOCK TABLES;

master 执行的语句在 slave 上执行失败怎么办？

1. 确定是否从服务器的表与主服务器的不同。尽力了解发生的原因。然后让从服务器的表与主服务器的一样并运行 START SLAVE。
2. 如果前面的步骤不工作或不适合，尽力了解手动更新是否安全(如果需要)，然后忽视来自主服务器的下一个语句。如果你确定可以跳过来自主服务器的下一个语句，执行下面的语句：

   mysql> SET GLOBAL SQL_slave_SKIP_COUNTER = n;

   mysql> START SLAVE;

   如果来自主服务器的下一个语句不使用 AUTO_INCREMENT 或 LAST_INSERT_ID()，n 值应为 1。否则，值应为 2。使用 AUTO_INCREMENT 或 LAST_INSERT_ID()的语句使用值 2 的原因是它们从主服务器的二进制日志中取两个事件。

Slave 上出现 log event entry exceeded max_allowed_packet 错误怎么办？

在主从服务器上增加 max_allowed_packet 参数的大小：

mysql> SET @@global.max_allowed_packet=16777216;

同时在 my.cnf 里，设置 max_allowed_packet = 16M, 保证下次重新启动后继续有效。

多主复制时，自动增长变量的冲突问题

在单主复制时，系统参数 auto_increment_increment 和 auto_increment_offset 可以采用默认设置，但是多主复制时，要定制 auto_increment_increment 和 auto_increment_offset 的设置，保证多主之间不会有重复冲突。比如两个 master 的情况可以如下设置：

Master1 上：auto_increment_increment = 2，auto_increment_offset = 1;（1,3,5,7…序列）

Master2 上：auto_increment_increment = 2，auto_increment_offset = 0;（0,2,4,6…序列）

怎么样知道 slave 上现在复制到什么地方了

可以查看 SHOW SLAVE STATUS 语句的 Seconds_Behind_Master 列的结果

需要注意的问题：

Mysql Cluster：

Mysql Cluster 概述：

MySql 自 4.1.x 开始推出 MySql Cluster 功能（NDB 引擎），开始 NDB 引擎只是支持基于内存的数据表，到现在 5.1beta 版时，开始支持基于磁盘的数据表。

理论上，MySql Cluster 通过数据的分布式存储和可扩展的系统架构，可以满足更大规模的应用；通过冗余策略，大大提高了系统的可靠性和数据的有效性。

虽然早在 5.0 时就有公司将 MySql Cluster 用于正式生产环境，但是更多的测试（包括我们自己的测试）表明，MySql Cluster 在性能和可靠性上还有待于完善，我们期待 MySql5.1 正式版发布时，MySql Cluster 在性能和可靠性上能够有重大的改进。

Mysql Cluster 架构：

mysql cluster 由三部分构成：

管理节点

管理节点需要config.ini文件，该文件通知节点有多少需要维护的副本，需要在每个数据节点上为数据和索引分配多少内存，数据节点的位置，在每个数据节点上保存数据的磁盘位置，以及SQL节点的位置。管理节点只能有一个，要求配置不高。

sql 节点

前台通过 sql 节点来访问 mysql cluster 的数据节点里面的数据。可以有多个 sql 节点，通过每个 sql 节点查询到的数据都是一致的。而且 sql 节点性能越好。

数据节点

用来存放 cluster 里面的数据，可以有多个数据节点。每个数据节点可以有多个镜像 copy

这三种节点物理上可以在不同的服务器上，也可以在同一台服务器上

安装配置：

管理节点配置步骤：

mkdir /var/lib/mysql-cluster

cd /var/lib/mysql-cluster

vi config.ini

对于我们的典型设置，config.ini 文件应类似于：

[NDBD DEFAULT]

NoOfReplicas=2　　#每个数据节点的镜像数量

DataMemory=500M　　# How much memory to allocate for data storage

IndexMemory=300M　　# How much memory to allocate for index storage

[TCP DEFAULT]

portnumber=2202　#数据节点的默认连接端口

[NDB_MGMD]　　#配置管理节点

id=1

hostname=192.168.7.187

datadir=/home/zzx2/mysql-cluster

[NDBD]

id=2

HostName=192.168.7.187

DataDir=/home/zzx2/mysql/data

[NDBD]

id=3

hostname=192.168.7.55

datadir=/home/zzx/mysql/data

[NDBD]

id=4

HostName=192.168.7.187

DataDir=/home/zzx3/mysql/data

[NDBD]

id=5

hostname=192.168.7.55

datadir=/home1/zzx1/mysql/data

[MYSQLD]

hostname=192.168.7.187

[MYSQLD]

hostname=192.168.7.55

[MYSQLD]# Options for mysqld process:

其中，每个节点都要有一个独立的 id 号，具体说明如下：

[NDB_MGMD]表示管理节点的配置,只能有一个

[NDBD]表示数据节点的配置，可以有多个

[MYSQLD] 表示 sql 节点的配置，可以有多个。也可以是空节点

sql 节点和数据节点的配置：

```
# Options for mysqld process:
[MYSQLD]
ndbcluster                      # run NDB engine
ndb-connectstring=192.168.0.10  # location of MGM node
# Options for ndbd process:
[MYSQL_CLUSTER]
ndb-connectstring=192.168.0.10  # location of MGM node
```

管理维护：

Cluster 的启动

在管理主机上，从系统shell发出下述命令以启动MGM节点进程：

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```

在每台数据节点主机上，对于首次启动，运行下述命令启动NDBD进程：

```
shell> ndbd --initial --ndb-connectstring=192.168.7.187:1186
```

注意，仅应在首次启动ndbd时，或在备份／恢复或配置变化后重启ndbd时使用“--initial”参数，这很重要。原因在于，该参数会使节点删除由早期ndbd实例创建的、用于恢复的任何文件，包括恢复用日志文件

sql节点顺序启动mysql服务

```
node 1: ./bin/mysqld_safe &
......
node n: ./bin/mysqld_safe &
```

在任何一个sql节点运行mysql客户端，即可以连接到mysql cluster数据库

注意：创建表必须用 engine＝ndb 选项来使得新创建的表是 ndb 表。否则数据将不会存储在 ndb 数据节点中，并且其他 sql 节点将看不到此表

Cluster 的关闭

shell> **ndb_mgm -e shutdown**

数据备份和恢复：

备份步骤：

启动管理服务器(ndb_mgm)

执行命令：start backup

管理服务器回复“备份 backup_id 开始”，其中，backup_id 是该备份的唯一 ID（如果未作其他配置，该 ID 还将保存在簇日志中）。

管理服务器发出消息“备份 backup_id 完成”，通知备份操作已结束。

备份路径：$MYSQL_HOME/data/BACKUP/BACKUP-备份 ID

恢复步骤：

1. 进入单用户模式（先运行 ndb_mgm）

   NDB> ENTER SINGLE USER MODE 5

2. ndb_restore -b 2 -n 2 -c host=192.168.7.187:1186

   -m -r /home/zzx2/mysql/data/BACKUP/BACKUP-2

   b：备份 id

   n：节点 id

   m：恢复表定义

   r：恢复路径

   注意：－m 参数是恢复表定义使用，只需要第一个节点带此参数就可以，否则会报错：

   Table or index with given name already exists

   Restore: Failed to restore table: cluster/def/NDB$BLOB_2_3 ... Exiting

3. 退出单用户模式

NDB> EXIT SINGLE USER MODE

Oracle 向 Mysql 数据迁移：

数据类型的差异：

在迁移 oracle 数据到 mysql 数据库时,首先要清楚 2 个数据类型的差异,并且在迁移前确定 oracle 中的数据类型在 mysql 数据库中使用什么样的数据类型来替换最为合适,在转换时对一些不确定的数据类型可以通过必要的测试来进行检测和确认.

这里给出了常用的几种数据类型的对照:

Oracle 数据类型	mysql 数据类型
Date	datetime
Char	char
varchar2	varchar
Clob	text 最大长度为65,535(2^{16} - 1)字符的TEXT列。
Number	MEDIUMINT 或 INT 或DECIMAL--针对带有浮点数的数据

另外,可以通过 Mysql Migration Toolkit 工具,将 oracle 的数据类型转换为 mysql 数据类型。

这个工具在 mysql 网站上提供下载.但是此工具目前只有 windows 版本。

直接使用 Mysql Migration Toolkit 工具 将 oracle 数据导入到 mysql 数据库,会出现很多问题,建议对存在大量 clob 字段的数据最好不要使用此工具进行数据的导入。

利用导出文本迁移：

导出为 insert sql 文本

生成 sql 文件,直接使用 mysql 命令进行导入.

例如:

使用 oracle 的 select 语句生成 sql 文件

SQL>spool test.sql

```
set head off
set pagesize 0
set recsep OFF
set wrap off
set feedback off
set linesize 200
set termout off
set trimspool on
select 'insert into test  values(' ||id||');' from test;
spool off;
```

传输要导入的 sql 文件到 mysql 服务器

执行下列命令导入:

```
mysql -u root test <test.sql
```

导出为固定格式文本

将导入数据生成一定格式的文本文件(利用 spool 或者一些工具，例如 golden)，然后使用 LOAD DATA INFILE 语句,高速地从文件中读取行，并装入指定表中。文件名称必须为一个文字字符串。

例如:

```
mysql> USE test;
mysql>LOAD DATA INFILE '/home/bjguan/tt.txt' INTO TABLE test.tt FIELDS TERMINATED BY '\t' ;
```

或

```
mysql -e "LOAD DATA INFILE '/home/bjguan/tt.txt' INTO TABLE tt FIELDS TERMINATED BY '\t' " test -u root
```

利用工具软件迁移：

Mysql 官方有自己开发的迁移工具 Mysql Migration Toolkit，但是在实际迁移过程中发现，此工具对于转义字符的处理问题很多，对于含有 varchar 和 clob 的字段经常出错，因此不建议使用

使用 DBA 组开发的迁移工具：

为了解决迁移过程中出现的各种问题，dba 组开发了一个迁移工具，主要功能是对 oracle 的数据进行导出，导出的时候，对各种转义字符都做了相应处理，将导出的文件直接 load data infile 到目标表即可。

使用步骤：

1、服务器上必须安装 jre，如果用 oci 连接，则还要有 oracle 客户端

2、将附件中的包和类加入 classpath

java Migration ’连接串’ ’目标路径’ ’字段分隔符’ ’记录分隔符’ ‘table1，table2.....’ feedbackBig feedbackSmall

其中： Migration 为类名称

连接串为 oracle 的 jdbc 连接字符串，如果为 thin 连接，用以下格式：

jdbc:oracle:thin:@ip:port:dbname,username,passwd

如果为 oci 连接，则格式为：

jdbc:oracle:oci8:@tns

目标路径为要生成的文件要保存到的目标目录，例如/home/zzx/,后面的/一定要写啊，在目标目录下一个表产生一 个文件

字段分隔符为一个字符，例如！

记录分隔符为一个字符，李然＃

table1、table2 表示导出多个表，多个表之间用逗号分隔开，

feedbackBig/small 分别指不包含/包含 clob 字段的表每次写入文件的记录条数，如果出现内存溢出，则把相应参数调小，对小表，则可以增大相应参数，使得导出更快！

后三个参数都可以不写，默认值分别为导出全部表，50000 和 3000

举一个完整的例子：

java Migration ’jdbc:oracle:thin:@202.106.168.88:1521:riji，user，passwd’ ’/home/yellowpage/zzx/’ ’!’ ’#’

java Migration ’jdbc:oracle:oci8:@riji , user , passwd’ ’/home/yellowpage/zzx/’ ’!’ #’

3、生成的文件大家直接在 mysql 服务器上运行 load 命令即可导入

数据迁移常见问题：

字符集问题：

从 oracle 导出时的客户端字符集要等于要导入的 mysql 数据库字符集以保证数据的一致性：

Oracle：NLS_LANG

MYSQL:default-character-set

特殊字符处理：

导入过程中出错很重要的一点是对转义字符的处理不正确而出错，在 dba 组开发的迁移程序中，对所有字符串中含有我们定义的域分隔符、行分隔符和转义字符’\’的字符，全部加上了转义。这样，我们就可以保证我们定义的分隔符可以正常发挥作用。在 load data 语法中，虽然也可以手工指定各种分隔符和转义字符，但是如果导出文件的字符中含有和指定分隔符相同的字符时，会经常出错。

日期字段的处理：

Oracle 中导出日期的格式，一定要和 mysql 支持的格式一致，否则导入的时候会出错，mysql 支持的日期格式如下：

常见格式指定 DATETIME、DATE 和 TIMESTAMP 值：

’YYYY-MM-DD HH:MM:SS’或’YY-MM-DD HH:MM:SS’格式的字符串。允许“不严格”语法：任何标点符都可以用做日期部分或时间部分之间的间割符。例如，’98-12-31 11:30:45’、’98.12.31 11+30+45’、’98/12/31 11*30*45’和’98@12@31 11^30^45’是等价的。

'YYYY-MM-DD'或'YY-MM-DD'格式的字符串。这里也允许使用“不严格的”语法。例如，'98-12-31'、'98.12.31'、'98/12/31'和'98@12@31'是等价的。

'YYYYMMDDHHMMSS'或'YYMMDDHHMMSS'格式的没有间割符的字符串，假定字符串对于日期类型是有意义的。例如，'19970523091528'和'970523091528'被解释为'1997-05-23 09:15:28'，但'971122129015'是不合法的(它有一个没有意义的分钟部分)，将变为'0000-00-00 00:00:00'。

'YYYYMMDD'或'YYMMDD'格式的没有间割符的字符串，假定字符串对于日期类型是有意义的。例如，'19970523'和'970523'被解释为 '1997-05-23'，但'971332'是不合法的(它有一个没有意义的月和日部分)，将变为'0000-00-00'。

YYYYMMDDHHMMSS 或 YYMMDDHHMMSS 格式的数字，假定数字对于日期类型是有意义的。例如，19830905132800 和 830905132800 被解释为 '1983-09-05 13:28:00'。

YYYYMMDD 或 YYMMDD 格式的数字，假定数字对于日期类型是有意义的。例如，19830905 和 830905 被解释为'1983-09-05'。

如何使迁移过程不被 SQL 错误中断：

启动 mysql 客户端进行连接的时候加上-f ，-f 可以强制出错的时候，记录错误但不终止 load 过程；--show-warnings 显示出错原因

如何查找产生 warnings 的原因：

启动 mysql 客户端的时候加上-v --show-warnings 参数，-v 显示出错的数据。可以用 tee 工具将 load 过程中的错误详细记录在文本中。

应急处理：

一般处理流程：

如果数据库在运行过程中出现任何异常，一般按照如下步骤解决：

查看错误日志。错误日志一般放在数据库的 data 目录下

通过 perror 工具查看错误号，判断错误出现的原因

寻找解决方案

忘记 root 密码：

如果你忘记了 MySQL 的 root 用户的口令，你可以用下列过程恢复它。

通过发送一个 kill(不是 kill -9)到 mysqld 服务器来关闭 mysqld 服务器。pid 被保存在一个.pid 文件中,通常在 MySQL 数据库目录中(你必须是一个 UNIX root 用户或运行服务器的相同用户做这个)：

kill `cat /mysql-data-directory/hostname.pid`

使用--skip-grant-tables 选项重启 mysqld。

用 mysql -h hostname mysql 连接 mysqld 服务器并且用一条 GRANT 命令改变口令。你也可以用 mysqladmin -h hostname -u user password 'new password' 进行。

用 mysqladmin -h hostname flush-privileges 或用 SQL 命令 FLUSH PRIVILEGES 来装载权限表()

表损坏如何处理：

一张损坏的表的症状通常是查询意外中断并且能看到下述错误：

“*tbl_name*.frm”被锁定不能更改。

不能找到文件“*tbl_name*.MYI”（Errcode：*nnn*)。

文件意外结束。

记录文件被毁坏。

从表处理器得到错误*nnn*

解决方法如下：

方法一：

myisamchk -r tablename

上面的方法几乎能解决所有问题，如果不行，则使用：

myisamchk -o tablename

方法二：

CHECK TABLE *tbl_name* [, *tbl_name*] ... [*option*] ...

option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}

REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE

tbl_name [, *tbl_name*] ... [QUICK] [EXTENDED] [USE_FRM]

MyISAM 表超过 4G 无法访问：

mysql5.0 以前的中的 myisam 存储引擎默认的表大小只支持到 4G，如果大于 4G，可以执行以下命令来扩大表的存储能力：

alter table weblogentry MAX_ROWS=1000000000 AVG_ROW_LENGTH=15000;

执行以下命令可以查看更改前后的表状态：

myisamchk －dv tablename

其中：

Max datafile length: 4294967294 Max keyfile length: 4398046510079

表明了本表实际支持的最大 MYD size 和最大 MYI size

数据目录磁盘空间不足怎么办？

如果建表前时候预测到 data 目录下的空间不足，则在建表时用如下选项指定数据目录和索引目录到磁盘充足的空间：

DATA DIRECTORY = '*absolute path to directory*'

INDEX DIRECTORY = '*absolute path to directory*'

如果表已经创建，则可以将表的数据文件和索引文件 mv 到磁盘充足的分区上，然后在原文件处创建符号链接即可。当然，mv 前最好停机或者将表锁定，以防止表的更改。

如何禁止 DNS 反向解析？

show processlist 命令出现了非常多个进程，但是这些进程很奇怪，类似于：

unauthenticated user | 192.168.5.71:57857 | NULL | Connect | NULL | login | NULL

这些并不是我们正常的进程，原来这是 mysql 的一个 bug，是由于反复解析 ip 和 dns 造成的，启动的时候加上--skip-name-resolve 选项就可以避免

Mysql 管理中一些常用的命令和技巧：

参数设置方法：

1) 如果对服务器参数不熟悉，建议从$MYSQL_HOME/support-files 下面按照需要 cp 合适的配置文件为数据库配置文件，例如：

   cp my-large.cnf /etc/my.cnf

2) session 级修改（只对本 session 有效）：

   set para_name=value;

3) 全局级修改（对所有新的连接都有效,但是数据库重启后失效）

   set global para_name=value;

4) 永久修改

   将参数在 my.cnf 中增加或者修改

mysql.sock 丢失后怎么连接数据库？

请注意，如果你指定 localhost 作为一个主机名，mysqladmin 默认使用 Unix 套接字文件连接,而不是 TCP/IP。从 MySQL 4.1 开始,通过--protocol= TCP | SOCKET | PIPE | MEMORY}选项，你可以显示地指定连接协议 ，举例如下：

socket 连接：

[zzx@zzx mysql]$ mysql -uroot

ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/home/zzx/mysql/mysql.sock' (2)

tcp 连接：

```
[zzx@zzx mysql]$ mysql --protocol=TCP -uroot -p -P3307 -hlocalhost
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 73 to server version: 5.0.15-standard
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

同一台机器运行多个 mysql：

最简单的方法，将每个 mysql 安装在不同的用户下面，例如 mysql1 和 mysql2，每个用户下面，分别执行如下操作：

```
export MYSQL_HOME=/home/mysql1/mysql
shell> groupadd mysql
shell> useradd -g mysql mysql1
shell> cd /home/mysql1
shell>tar -xzvf  /home/mysql1/mysql-VERSION-OS.tar.gz
shell> ln -s mysql-VERSION-OS.tar.gz mysql
shell> cd mysql
cp support-files/my-large.cnf(根据实际情况选择)  ./my.cnf
```

vi my.cnf ，主要修改[client]和[mysqld]下面的 port 和 socket，并指定字符集，例如：

```
[client]
port            = 3307
socket          = /home/mysql1/mysql/data/mysql.sock
# The MySQL server
[mysqld]
default-character-set = utf8
port            = 3307
socket          = /home/mysql1/mysql/data/mysql.sock
。。。。。。
shell> scripts/mysql_install_db --user=mysql1
```

shell> chown -R root:mysql .

shell> chown -R mysql1:mysql data

shell> bin/mysqld_safe --user=mysql &

mysql2 用户执行的和 mysql1 类似，不同的是指定不同的 MYSQL_HOME，不同的 port、socket 即可

查看用户权限：

怎么样查看用户权限，最简单的方法，通过如下语句：

mysql> show grants for 'test1'@'localhost';

如果要通过权限表来查看，比较复杂：

在 5.0 以下，按照以下顺序来查看：

user->db->tables_priv->columns_priv，权限范围依次递减。和参数的设置不同，权限设置的原则是：

全局权限覆盖局部权限

首先，从 user 表中查看 user 和 host 对应的那些权限值，比如：

select_priv="Y"

说明此用户组具有对所有数据库的所有表的 select 权限，此时，再单独对某个数据库设置 select 权限已经没有意义

如果 user 表中的 select_priv="N"，则接着查看 db 表中对应用户组的权限，如果存在一条记录如下：

Host	Db	User	Select_priv	Insert_priv	Update_priv	Delete_priv	Create_priv	Drop_priv	Grant_priv	References_priv	Index_priv	Alter_priv	Create_tmp_table_priv	Lock_tables_priv
localhost	test2	test1	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y

则表示 test1@localhost 用户组对 test2 数据库中的所有表具有所有权限（除了

grant)，此时单独对此数据库内的表进行权限设置已经没有意义；如果没有此记录或者对应权限不是“N”，则接着查询 tables_priv 表，此表中的记录决定了对数据库中实际表的权限；如果 tables_priv 内记录的权限都是 Y，则对表内的任何列单独设置权限已经没有意义，如果 tables_priv 没有对应表的记录或者对应权限不是“N”，则接着查询 columnss_priv 表的记录。

一步一般类推，最后得出某个用户组的权限。

在 mysql 5.0 以后，多了一个数据字典库 information_schema，通过这个库里面的 USER_PRIVILEGES、SCHEMA_PRIVILEGES、TABLE_PRIVILEGES、COLUMN_PRIVILEGES 表可以得到同样的结论。

修改用户密码：

方法 1：

可以用 mysqladmin 命令在命令行指定密码：

```
shell> mysqladmin -u user_name -h host_name password "newpwd"
```

方法 2：

为账户赋予密码的另一种方法是执行 SET PASSWORD 语句：

```
mysql> SET PASSWORD FOR 'jeffrey'@'%' = PASSWORD('biscuit');
```

如果是更改自己的密码，可以省略 for 语句：

```
mysql> SET PASSWORD = PASSWORD('biscuit');
```

方法 3：

你还可以在全局级别使用 GRANT USAGE 语句(在*.*)来指定某个账户的密码而不影响账户当前的权限：

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'%' IDENTIFIED BY 'biscuit';
```

方法 4：

直接更改数据库的 user 表：

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('%','jeffrey',PASSWORD('biscuit'));
mysql> FLUSH PRIVILEGES;
```

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('bagel')
    -> WHERE Host = '%' AND User = 'francis';
mysql> FLUSH PRIVILEGES;
```

注意：更改密码时候一定要使用 password 函数（mysqladmin 和 grant 两种方式不用写，会自动加上）

怎样灵活的指定连接的主机：

在 user 表 Host 值的指定方法：

- Host 值可以是主机名或 IP 号，或'localhost'指出本地主机。
- 你可以在 Host 列值使用通配符字符“%”和“_”。

Host 值'%'匹配任何主机名，空 Host 值等价于'%'。它们的含义与 LIKE 操作符的模式匹配操作相同。例如，'%'的 Host 值与所有主机名匹配，而'%.mysql.com'匹配 mysql.com 域的所有主机。

到底匹配哪个符合条件的用户：

例如以下两个用户：

'thomas.loc.gov' 'fred' fred, 从 thomas.loc.gov 连接

'%' 'fred' fred, 从任何主机连接

当从主机 thomas.loc.gov 进行连接的时候，上面两个用户显然都满足条件，该选择哪个呢？

如果有多个匹配，服务器必须选择使用哪个条目。按照下述方法解决问题：

服务器在启动时读入 user 表后进行排序。

然后当用户试图连接时，以排序的顺序浏览条目

服务器使用与客户端和用户名匹配的第一行。

当服务器读取表时，它首先以最具体的 Host 值排序。主机名和 IP 号是最具体的。'%'意味着“任何主机”并且是最不特定的。有相同 Host 值的条目首先以最具体的 User 值排序(空 User 值意味着“任何用户”并且是最不特定的)。例如下例是排序前和排序后的结果：

```
+-----------+----------+-
| Host      | User     | …
+-----------+----------+-
| %         | root     | …
| %         | jeffrey  | …
| localhost | root     | …
| localhost |          | …
+-----------+----------+-
```

排序前

```
+-----------+----------+-
| Host      | User     | …
+-----------+----------+-
| localhost | root     | … ...
| localhost |          | … ...
| %         | jeffrey  | … ...
| %         | root     | … ...
+-----------+----------+-
```

排序后

- 记住：明确指定用户名的用户不一定是被匹配的用户

不进入 mysql，怎样运行 sql 语句？

使用--execute（-e）选项：

```
mysql -u root -p -e "SELECT User, Host FROM User" mysql
```

可以按这种方式传递多个 SQL 语句，用分号隔开：

```
shell> mysql -u root -p -e "SELECT Name FROM Country WHERE Name LIKE
'AU%';SELECT COUNT(*) FROM City" world
Enter password: ******
+-----------+
| Name      |
```

```
+-----------+
| Australia |
| Austria   |
+-----------+
+----------+
| COUNT(*) |
+----------+
|     4079 |
+----------+
```

请注意长形式(--execute)后面必须紧跟一个等号(=)。

客户端怎么访问内网数据库？

oracle 的客户端可以通过 cman 来访问内网中的 oralce 数据库，mysql 能实现类似功能吗？可以，假设服务器如下：

中转服务器 ip：202.108.15.160（192.168.161.38）

内网服务器 ip：192.168.161.39，在端口 3306 上起着 mysql 服务

客户端：windows，secureCRT

- 在中转服务器上增加 ssh turnal，具体操作如下：

1. 点击 session 的属性
2. 点击 connection－>port_forwarding
3. 点击 add 按钮：name 中随便起个名字；local 下的 ip 写上 127.0.0.1，port 随便起一个未使用的 port，例如 9999；remote 下面的的 hostname 写上 192.168.161.39，端口写上 3306；点击 ok 设置成功

- 在 192.168.161.39 的 mysql 内增加一个用户 test，host 设置为 192.168.161.38
- grant select on dbname.* to test@192.168.161.38 identified by '123';

客户端执行 mysql -h127.0.0.1 -P3306 -utest -p123，连接成功

1. 本站属于公益技术站点，目的是宣传和推广计算机设计与开发技术，为国内技术发展做一定的贡献!
2. 我们的目标和口号：源码网,做最好的源码下载站！
3. 我们的站点：

   总站：

   论坛：bbs.codepub.com

   学院：info.codepub.com

   博客：my.codepub.com

   聚合：rss.codepub.com
4. 欢迎各种形式的合作共赢！
5. 站长：无色幽默
6. E-mail/MSN:wuse#codepub.com
7. QQ:24721515
8. 备注：什么是 CP，就是 CodePub 的简写。