目录



# MySQL 优化

(本文是 Monty 在 O'Reilly Open Source Convention 2000 大会上的演讲)

# 1. 我们可以且应该优化什么？

硬件
操作系统/软件库
SQL 服务器(设置和查询)
应用编程接口(API)
应用程序

# 2. 优化硬件

如果你需要庞大的数据库表(>2G)，你应该考虑使用 64 位的硬件结构，像 Alpha、Sparc 或即将推出的 IA64。因为 MySQL 内部使用大量 64 位的整数，64 位的 CPU 将提供更好的性能。

对大数据库，优化的次序一般是 RAM、快速硬盘、CPU 能力。

更多的内存通过将最常用的键码页面存放在内存中可以加速键码的更新。

如果不使用事务安全(transaction-safe)的表或有大表并且想避免长文件检查，一台 UPS 就能够在电源故障时让系统安全关闭。

对于数据库存放在一个专用服务器的系统，应该考虑 1G 的以太网。延迟与吞吐量同样重要。

# 3. 优化磁盘

为系统、程序和临时文件配备一个专用磁盘，如果确是进行很多修改工作，将更新日志和事务日志放在专用磁盘上。

低寻道时间对数据库磁盘非常重要。对与大表，你可以估计你将需要：

log(行数)/log(索引块长度/3*2/(键码长度 + 数据指针长度))+1 次寻到才能找到一行。

对于有 500000 行的表，索引 Mediun int 类型的列，需要：

log(500000) / log(1024/3*2/(3 + 2))+1=4 次寻道。

上述索引需要 500000*7*3/2=5.2M 的空间。实际上，大多数块将被缓存，所以大概只需要 1-2 次寻道。

然而对于写入（如上），你将需要 4 次寻道请求来找到在哪里存放新键码，而且一般要 2 次寻道来更新索引并写入一行。

对于非常大的数据库，你的应用将受到磁盘寻道速度的限制，随着数据量的增加呈 N log N 数据级递增。

将数据库和表分在不同的磁盘上。在 MySQL 中，你可以为此而使用符号链接。

条列磁盘(RAID 0)将提高读和写的吞吐量。带镜像的条列(RAID 0+1)将更安全并提高读取的吞吐量。写入的吞吐量将有所降低。不要对临时文件或可以很容易地重建的数据所在的磁盘使用镜像或 RAID(除了 RAID 0)。

在 Linux 上，在引导时对磁盘使用命令 hdparm -m16 -d1 以启用同时读写多个扇区和 DMA 功能。这可以将响应时间提高 5~50%。

在 Linux 上，用 async (默认)和 noatime 挂载磁盘(mount)。对于某些特定应用，可以对某些特定表使用内存磁盘，但通常不需要。

# 4. 优化操作系统

不要交换区。如果内存不足，增加更多的内存或配置你的系统使用较少内存。
不要使用 NFS 磁盘(会有 NFS 锁定的问题)。
增加系统和 MySQL 服务器的打开文件数量。(在 safe_mysqld 脚本中加入 ulimit -n #)。
增加系统的进程和线程数量。
如果你有相对较少的大表，告诉文件系统不要将文件打碎在不同的磁道上(Solaris)。
使用支持大文件的文件系统(Solaris)。
选择使用哪种文件系统。在 Linux 上的 Reiserfs 对于打开、读写都非常快。文件检查只需几秒种。

# 5. 选择应用编程接口

PERL：
可在不同的操作系统和数据库之间移植。
适宜快速原型。
应该使用 DBI/DBD 接口。
PHP：
比 PERL 易学。
使用比 PERL 少的资源。
通过升级到 PHP4 可以获得更快的速度。
C：
MySQL 的原生接口。
较快并赋予更多的控制。
低层，所以必须付出更多。
C++：
较高层次，给你更多的时间来编写应用。
仍在开发中
ODBC：
运行在 Windows 和 Unix 上。
几乎可在不同的 SQL 服务器间移植。
较慢。MyODBC 只是简单的直通驱动程序，比用原生接口慢 19%。
有很多方法做同样的事。很难像很多 ODBC 驱动程序那样运行，在不同的领域还有不同的错误。
问题成堆。Microsoft 偶尔还会改变接口。
不明朗的未来。(Microsoft 更推崇 OLE 而非 ODBC)
JDBC：
理论上可在不同的操作系统何时据库间移植。
可以运行在 web 客户端。
Python 和其他：
可能不错，可我们不用它们。

# 6. 优化应用

应该集中精力解决问题。

在编写应用时，应该决定什么是最重要的：

速度

操作系统间的可移植性

SQL 服务器间的可移植性

使用持续的连接。

缓存应用中的数据以减少 SQL 服务器的负载。

不要查询应用中不需要的列。

不要使用 SELECT * FROM table_name...

测试应用的所有部分，但将大部分精力放在在可能最坏的合理的负载下的测试整体应用。通过以一种模块化的方式进行，你应该能用一个快速"哑模块"替代找到的瓶颈，然后很容易地标出下一个瓶颈。

如果在一个批处理中进行大量修改，使用 LOCK TABLES。例如将多个 UPDATES 或 DELETES 集中在一起。

# 7. 应该使用可移植的应用

Perl DBI/DBD

ODBC

JDBC

Python(或其他有普遍 SQL 接口的语言)

你应该只使用存在于所有目的 SQL 服务器中或可以很容易地用其他构造模拟的 SQL 构造。www.mysql.com 上的 Crash-me 页可以帮助你。

为操作系统/SQL 服务器编写包装程序来提供缺少的功能。

# 8. 如果你需要更快的速度

应该找出瓶颈(CPU、磁盘、内存、SQL 服务器、操作系统、API 或应用)并集中全力解决。

使用给予你更快速度/灵活性的扩展。

逐渐了解 SQL 服务器以便能为你的问题使用可能最快的 SQL 构造并避免瓶颈。

优化表布局和查询。

使用复制以获得更快的选择(select)速度。

如果你有一个慢速的网络连接数据库，使用压缩客户/服务器协议。

不要害怕时应用的第一个版本不能完美地移植，在你解决问题时，你总是可以在以后优化它。

# 9. 优化 MySQLD

挑选编译器和编译选项。

为你的系统寻找最好的启动选项。

通读 MySQL 参考手册并阅读 Paul DuBios 的《MySQL》一书。(已有中文版-译注)

多用 EXPLAIN SELECT、SHOW VARIABLES、SHOW STATUS 和 SHOW PROCESSLIST。

了解查询优化器的工作原理。

优化表的格式。

维护你的表(myisamchk、CHECK TABLE、 OPTIMIZE TABLE)

使用 MySQL 的扩展功能以让一切快速完成。

如果你注意到了你将在很多场合需要某些函数，编写 MySQL UDF 函数。

不要使用表级或列级的 GRANT，除非你确实需要。

购买 MySQL 技术支持以帮助你解决问题:)

# 10. 编译和安装 MySQL

通过位你的系统挑选可能最好的编译器，你通常可以获得 10-30%的性能提高。

在 Linux/Intel 平台上，用 pgcc(gcc 的奔腾芯片优化版)编译 MySQL。然而，二进制代码将只能运行在 Intel 奔腾 CPU 上。

对于一种特定的平台，使用 MySQL 参考手册上推荐的优化选项。

一般地，对特定 CPU 的原生编译器(如 Sparc 的 Sun Workshop)应该比 gcc 提供更好的性能，但不总是这样。

用你将使用的字符集编译 MySQL。

静态编译生成 mysqld 的执行文件(用--with-mysqld-ldflags=all-static)并用 strip sql/mysqld 整理最终的执行文件。

注意，既然 MySQL 不使用 C++扩展，不带扩展支持编译 MySQL 将赢得巨大的性能提高。

如果操作系统支持原生线程，使用原生线程(而不用 mit-pthreads)。

用 MySQL 基准测试来测试最终的二进制代码。

# 11. 维护

如果可能，偶尔运行一下 OPTIMIZE table，这对大量更新的变长行非常重要。

偶尔用 myisamchk -a 更新一下表中的键码分布统计。记住在做之前关掉 MySQL。

如果有碎片文件，可能值得将所有文件复制到另一个磁盘上，清除原来的磁盘并拷回文件。

如果遇到问题，用 myisamchk 或 CHECK table 检查表。

用 mysqladmin -i10 precesslist extended-status 监控 MySQL 的状态。

用 MySQL GUI 客户程序，你可以在不同的窗口内监控进程列表和状态。

使用 mysqladmin debug 获得有关锁定和性能的信息。

# 12. 优化 SQL

扬 SQL 之长，其它事情交由应用去做。使用 SQL 服务器来做：

- 找出基于 WHERE 子句的行。
- JOIN 表
- GROUP BY
- ORDER BY
- DISTINCT

不要使用 SQL 来做：

- 检验数据(如日期)
- 成为一只计算器

技巧：

- 明智地使用键码。
- 键码适合搜索，但不适合索引列的插入/更新。

保持数据为数据库第三范式，但不要担心冗余信息或这如果你需要更快的速度，创建总结表。

在大表上不做 GROUP BY，相反创建大表的总结表并查询它。

UPDATE table set count=count+1 where key_column=constant 非常快。

对于大表，或许最好偶尔生成总结表而不是一直保持总结表。

充分利用 INSERT 的默认值。

# 13. 不同 SQL 服务器的速度差别（以秒计）

| 通过键码读取 2000000 行： | NT | Linux |
|---|---|---|
| mysql | 367 | 249 |
| mysql_odbc | 464 | |
| db2_odbc | 1206 | |
| informix_odbc | 121126 | |
| ms-sql_odbc | 1634 | |
| oracle_odbc | 20800 | |
| solid_odbc | 877 | |
| sybase_odbc | 17614 | |

| 插入 350768 行： | NT | Linux |
|---|---|---|
| mysql | 381 | 206 |
| mysql_odbc | 619 | |
| db2_odbc | 3460 | |
| informix_odbc | 2692 | |
| ms-sql_odbc | 4012 | |
| oracle_odbc | 11291 | |
| solid_odbc | 1801 | |

```
+-------------------------+--------+---------+
|sybase_odbc                | 4802   |         |
+-------------------------+--------+---------+
```

在上述测试中，MySQL 配置 8M 高速缓存运行，其他数据库以默认安装运行。

# 14. 重要的 MySQL 启动选项

back_log 如果需要大量新连接，修改它。
thread_cache_size 如果需要大量新连接，修改它。
key_buffer_size 索引页池，可以设成很大。
bdb_cache_size BDB 表使用的记录和键吗高速缓存。
table_cache 如果有很多的表和并发连接，修改它。
delay_key_write 如果需要缓存所有键码写入，设置它。
log_slow_queries 找出需花大量时间的查询。
max_heap_table_size 用于 GROUP BY
sort_buffer 用于 ORDER BY 和 GROUP BY
myisam_sort_buffer_size 用于 REPAIR TABLE
join_buffer_size 在进行无键吗的联结时使用。

# 15. 优化表

MySQL 拥有一套丰富的类型。你应该对每一列尝试使用最有效的类型。

ANALYSE 过程可以帮助你找到表的最优类型：SELECT * FROM table_name PROCEDURE ANALYSE()。

对于不保存 NULL 值的列使用 NOT NULL，这对你想索引的列尤其重要。

将 ISAM 类型的表改为 MyISAM。

如果可能，用固定的表格式创建表。

不要索引你不想用的东西。

利用 MySQL 能按一个索引的前缀进行查询的事实。如果你有索引 INDEX(a,b)，你不需要在 a 上的索引。

不在长 CHAR/VARCHAR 列上创建索引，而只索引列的一个前缀以节省存储空间。CREATE TABLE table_name (hostname CHAR(255) not null, index(hostname(10)))

对每个表使用最有效的表格式。

在不同表中保存相同信息的列应该有同样的定义并具有相同的列名。

# 16. MySQL 如何次存储数据

数据库以目录存储。

表以文件存储。

列以变长或定长格式存储在文件中。对 BDB 表，数据以页面形式存储。

支持基于内存的表。

数据库和表可在不同的磁盘上用符号连接起来。

在 Windows 上，MySQL 支持用.sym 文件内部符号连接数据库。

MySQL 表类型

HEAP 表：固定行长的表，只存储在内存中并用 HASH 索引进行索引。

ISAM 表：MySQL 3.22 中的早期 B-tree 表格式。

MyIASM：IASM 表的新版本，有如下扩展：

二进制层次的可移植性。

NULL 列索引。

对变长行比 ISAM 表有更少的碎片。

支持大文件。

更好的索引压缩。

更好的键吗统计分布。

更好和更快的 auto_increment 处理。

来自 Sleepcat 的 Berkeley DB(BDB)表：事务安全(有 BEGIN WORK/COMMIT|ROLLBACK)。

# 17. MySQL 行类型（专指 IASM/MyIASM 表）

如果所有列是定长格式(没有 VARCHAR、BLOB 或 TEXT)，MySQL 将以定长表格式创建表，否则表以动态长度格式创建。

定长格式比动态长度格式快很多并更安全。

动态长度行格式一般占用较少的存储空间，但如果表频繁更新，会产生碎片。

在某些情况下，不值得将所有 VARCHAR、BLOB 和 TEXT 列转移到另一个表中，只是获得主表上的更快速度。

-利用 myiasmchk（对 ISAM，pack_iasm），可以创建只读压缩表，这使磁盘使用率最小，但使用慢速磁盘时，这非常不错。压缩表充分地利用将不再更新的日志表

# 18. MySQL 缓存

1、MySQL 高速缓存（所有线程共享，一次性分配）

键码缓存：key_buffer_size，默认 8M。

表缓存：table_cache，默认 64。

线程缓存：thread_cache_size，默认 0。

主机名缓存：可在编译时修改，默认 128。

内存映射表：目前仅用于压缩表。

注意：MySQL 没有运行高速缓存，而让操作系统处理。

2、MySQL 缓存区变量（非共享，按需分配）

sort_buffer：ORDER BY/GROUP BY

record_buffer：扫描表。

join_buffer_size：无键联结

myisam_sort_buffer_size：REPAIR TABLE

net_buffer_length:对于读 SQL 语句并缓存结果。

tmp_table_size：临时结果的 HEAP 表大小。

# 19. MySQL 表高速缓存工作原理

每个 MyISAM 表的打开实例(instance)使用一个索引文件和一个数据文件。如果表被两个线程使用或在同一条查询中使用两次，MyIASM 将共享索引文件而是打开数据文件的另一个实例。

如果所有在高速缓存中的表都在使用，缓存将临时增加到比表缓存尺寸大些。如果是这样，下一个被释放的表将被关闭。

你可以通过检查 mysqld 的 Opened_tables 变量以检查表缓存是否太小。如果该值太高，你应该增大表高速缓存。

# 20. MySQL 扩展/优化-提供更快的速度

使用优化的表类型（HEAP、MyIASM 或 BDB 表）。
对数据使用优化的列。
如果可能使用定长行。
使用不同的锁定类型（SELECT HIGH_PRIORITY，INSERT LOW_PRIORITY）
Auto_increment
REPLACE (REPLACE INTO table_name VALUES (...))
INSERT DELAYED
LOAD DATA INFILE / LOAD_FILE()
使用多行 INSERT 一次插入多行。
SELECT INTO OUTFILE
LEFT JOIN, STRAIGHT JOIN
LEFT JOIN ，结合 IS NULL
ORDER BY 可在某些情况下使用键码。
如果只查询在一个索引中的列，将只使用索引树解决查询。
联结一般比子查询快（对大多数 SQL 服务器亦如此）。
LIMIT

```
SELECT * from table1 WHERE a > 10 LIMIT 10,20
DELETE * from table1 WHERE a > 10 LIMIT 10
```

foo IN (常数列表) 高度优化。
GET_LOCK()/RELEASE_LOCK()
LOCK TABLES
INSERT 和 SELECT 可同时运行。
UDF 函数可装载进一个正在运行的服务器。
压缩只读表。
CREATE TEMPORARY TABLE
CREATE TABLE .. SELECT
带 RAID 选项的 MyIASM 表将文件分割成很多文件以突破某些文件系统的 2G 限制。
Delay_keys
复制功能

# 21. MySQL 何时使用索引

对一个键码使用>, >=, =, <, <=, IF NULL 和 BETWEEN

```
SELECT * FROM table_name WHERE key_part1=1 and key_part2 > 5;
SELECT * FROM table_name WHERE key_part1 IS NULL;
```

当使用不以通配符开始的 LIKE

```
SELECT * FROM table_name WHERE key_part1 LIKE 'jani%'
```

在进行联结时从另一个表中提取行时

```
SELECT * from t1,t2 where t1.col=t2.key_part
```

找出指定索引的 MAX()或 MIN()值

```
SELECT MIN(key_part2),MAX(key_part2) FROM table_name where key_part1=10
```

一个键码的前缀使用 ORDER BY 或 GROUP BY

```
SELECT * FROM foo ORDER BY key_part1,key_part2,key_part3
```

在所有用在查询中的列是键码的一部分时间

```
SELECT key_part3 FROM table_name WHERE key_part1=1
```

# 22. MySQL 何时不使用索引

如果 MySQL 能估计出它将可能比扫描整张表还要快时，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
```

如果使用 HEAP 表且不用=搜索所有键码部分。

在 HEAP 表上使用 ORDER BY。

如果不是用键码第一部分

```
SELECT * FROM table_name WHERE key_part2=1
```

如果使用以一个通配符开始的 LIKE

```
SELECT * FROM table_name WHERE key_part1 LIKE '%jani%'
```

搜索一个索引而在另一个索引上做 ORDER BY

```
SELECT * from table_name WHERE key_part1 = # ORDER BY key2
```

# 23. 学会使用 EXPLAIN

对于每一条你认为太慢的查询使用 EXPLAIN!

```
mysql> explain select t3.DateOfAction, t1.TransactionID
    -> from t1 join t2 join t3
    -> where t2.ID = t1.TransactionID and t3.ID = t2.GroupID
    -> order by t3.DateOfAction, t1.TransactionID;
+-------+--------+---------------+---------+---------+-----------------+------+--------------------------------+
| table | type   | possible_keys | key     | key_len | ref             | rows | Extra                          |
+-------+--------+---------------+---------+---------+-----------------+------+--------------------------------+
| t1    | ALL    | NULL          | NULL    |    NULL | NULL            |   11 | Using temporary; Using filesort |
| t2    | ref    | ID            | ID      |       4 | t1.TransactionID |   13 |                                |
| t3    | eq_ref | PRIMARY       | PRIMARY |       4 | t2.GroupID      |    1 |                                |
+-------+--------+---------------+---------+---------+-----------------+------+--------------------------------+
```

ALL 和范围类型提示一个潜在的问题。

# 24. 学会使用 SHOW PROCESSLIST

使用 SHOW processlist 来发现正在做什么：

```
+----+-------+-----------+----+---------+------+--------------+-----------------------------------+
| Id | User  | Host      | db | Command | Time | State        | Info                              |
+----+-------+-----------+----+---------+------+--------------+-----------------------------------+
| 6  | monty | localhost | bp | Query   | 15   | Sending data | select * from station,station as s1 |
| 8  | monty | localhost |    | Query   | 0    |              | show processlist                  |
+----+-------+-----------+----+---------+------+--------------+-----------------------------------+
```

在 mysql 或 mysqladmin 中用 KILL 来杀死溜掉的线程。

# 25. 如何知晓 MySQL 解决一条查询

运行项列命令并试图弄明白其输出：

```
SHOW VARIABLES;
SHOW COLUMNS FROM ...\G
EXPLAIN SELECT ...\G
FLUSH STATUS;
SELECT ...;
SHOW STATUS;
```

# 26. MySQL 非常不错

日志

在进行很多连接时，连接非常快。

同时使用 SELECT 和 INSERT 的场合。

在不把更新与耗时太长的选择结合时。

在大多数选择/更新使用唯一键码时。

在使用没有长时间冲突锁定的多个表时。

在用大表时(MySQL 使用一个非常紧凑的表格式)。

# 27. MySQL 应避免的事情

用删掉的行更新或插入表，结合要耗时长的 SELECT。

在能放在 WHERE 子句中的列上用 HAVING。

不使用键码或键码不够唯一而进行 JOIN。

在不同列类型的列上 JOIN。

在不使用=匹配整个键码时使用 HEAP 表。

在 MySQL 监控程序中忘记在 UPDATE 或 DELETE 中使用一条 WHERE 子句。如果想这样做，使用 mysql 客户程序的--i-am-a-dummy 选项。

# 28. MySQL 各种锁定

内部表锁定

LOCK TABLES（所有表类型适用）

GET LOCK()/RELEASE LOCK()

页面锁定（对 BDB 表）

ALTER TABLE 也在 BDB 表上进行表锁定

LOCK TABLES 允许一个表有多个读者和一个写者

一般 WHERE 锁定具有比 READ 锁定高的优先级以避免让写入方干等。对于不重要的写入方，可以使用 LOW_PRIORITY 关键字让锁定处理器优选读取方。

```
UPDATE LOW_PRIORITY SET value=10 WHERE id=10;
```

# 29. 给 MySQL 更多信息以更好地解决问题的技巧

注意你总能去掉(加注释)MySQL 功能以使查询可移植：

SELECT /*! SQL_BUFFER_RESULTS */ ...

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。

SELECT SQL_SMALL_RESULT ... GROUP BY ...

告诉优化器结果集将只包含很少的行。

SELECT SQL_BIG_RESULT ... GROUP BY ...

告诉优化器结果集将包含很多行。

SELECT STRAIGHT_JOIN ...

强制优化器以出现在 FROM 子句中的次序联结表。

SELECT ... FROM table_name [USE INDEX (index_list) | IGNORE INDEX (index_list)] table_name2

强制 MySQL 使用/忽略列出的索引。

# 30. 事务的例子

MyIASM 表如何进行事务处理:

```
mysql> LOCK TABLES trans READ, customer WRITE;
mysql> select sum(value) from trans where customer_id=some_id;
mysql> update customer set total_value=sum_from_previous_statement
           where customer_id=some_id;
mysql> UNLOCK TABLES;
```

BDB 表如何进行事务:

```
mysql> BEGIN WORK;
mysql> select sum(value) from trans where customer_id=some_id;
mysql> update customer set total_value=sum_from_previous_statement
           where customer_id=some_id;
mysql> COMMIT;
```

注意你可以通过下列语句回避事务:

```
UPDATE customer SET value=value+new_value WHERE customer_id=some_id;
```

# 31. 使用 REPLACE 的例子

REPLACE 的功能极像 INSERT，除了如果一条老记录在一个唯一索引上具有与新纪录相同的值，那么老记录在新纪录插入前则被删除。不使用

```
SELECT 1 FROM t1 WHERE key=#
IF found-row
  LOCK TABLES t1
  DELETE FROM t1 WHERE key1=#
  INSERT INTO t1 VALUES (...)
  UNLOCK TABLES t1;
ENDIF
```

而用

```
REPLACE INTO t1 VALUES (...)
```

# 32. 一般技巧

使用短主键。联结表时使用数字而非字符串。

当使用多部分键码时，第一部分应该时最常用的部分。

有疑问时，首先使用更多重复的列以获得更好地键码压缩。

如果在同一台机器上运行 MySQL 客户和服务器，那么在连接 MySQL 时则使用套接字而不是 TCP/IP（这可以提高性能 7.5%）。可在连接 MySQL 服务器时不指定主机名或主机名为 localhost 来做到。

如果可能，使用--skip-locking(在某些 OS 上为默认)，这将关闭外部锁定并将提高性能。

使用应用层哈希值而非长键码：

```
SELECT * FROM table_name WHERE hash=MD5(concat(col1,col2)) AND
col_1='constant' AND col_2='constant'
```

在文件中保存需要以文件形式访问的 BLOB，在数据库中只保存文件名。
删除所有行比删除一大部分行要快。
如果 SQL 不够快，研究一下访问数据的较底层接口。

# 33. 使用 MySQL 3.23 的好处

MyISAM：可移植的大表格式
HEAP：内存中的表
Berkeley DB：支持事务的表。
众多提高的限制
动态字符集
更多的 STATUS 变量
CHECK 和 REPAIR 表
更快的 GROUP BY 和 DISTINCT
LEFT JOIN ... IF NULL 的优化
CREATE TABLE ... SELECT
CREATE TEMPORARY table_name (...)
临时 HEAP 表到 MyISAM 表的自动转换
复制
mysqlhotcopy 脚本

# 34. 正在积极开发的重要功能

改进事务处理
失败安全的复制
正文搜索
多个表的删除(之后完成多个表的更新)
更好的键码缓存
原子 RENAME (RENAME TABLE foo as foo_old, foo_new as foo)
查询高速缓存
MERGE TABLES
一个更好的 GUI 客户程序