# 嵌入式数据库横向性能对比测试

梁夏旭

201208

## 1、嵌入式数据库

嵌入式数据库的名称来自其独特的运行模式。这种数据库嵌入到了应用程序进程中，消除了与客户机服务器配置相关的开销。嵌入式数据库实际上是轻量级的，在运行时，它们需要较少的内存。它们由精简代码编写而成，对于嵌入式设备，速度更快，效果更理想。嵌入式运行模式允许嵌入式数据库通过SQL来轻松管理应用程序数据，而不依靠原始的文本文件。嵌入式数据库还提供零配置运行模式，这样可以启用其中一个并运行一个快照（snapshot）。

在嵌入式系统中，对数据库的操作具有定时限制的特性，这里把应用于嵌入式系统的数据库系统称为嵌入式数据库系统或嵌入式实时数据库系统（ERTDBS）。

可靠性要求是毋庸置疑的，嵌入式系统必须能够在没有人工干预的情况下，长时间不间断地运行。同时要求数据库操作具备可预知性，而且系统的大小和性能也都必须是可预知的，这样才能保证系统的性能。嵌入式系统中会不可避免地与底层硬件打交道，因此在数据管理时，也要有底层控制的能力，如什么时候会发生磁盘操作，磁盘操作的次数，如何控制等。底层控制的能力是决定数据库管理操作的关键。

目前嵌入式软件系统开发的挑战之一，体现在对各种数据的管理能否建立一套可靠、高效、稳定的管理模式，嵌入式数据库可谓应运而生。

嵌入式数据库是嵌入式系统的重要组成部分，也成为对越来越多的个性化应用开发和管理而采用的一种必不可少的有效手段。

嵌入式数据库用途广泛，如用于消费电子产品、移动计算设备、企业实时管理应用、网络存储与管理以及各种专用设备，这一市场目前正处于高速增长之中。　举简单例子，手机原来只用来打电话、发短信，现在手机增加了很多新的功能，比如彩信、音乐、摄影、视频等等，应用的功能多了，系统就变得复杂。

## 2、常用的嵌入式数据库的介绍

### 2.1 Progress

Progress软件公司2000年4月18号18时在京宣布，全面发售在Linux操作系统上运行的数据库及其部署产品。Progress在嵌入式数据库市场中拥有全球第一的占有率，世界上有超过200万人正在使用Progress软件公司的应用软件，目前部署Progress产品的站点数量已经超过100，000个。通过Progress软件公司第一个Linux版嵌入式数据库，独立软件开发商和最终用户可以在这一流行的操作系统上移植5，000多种商业应用。

Progress软件公司当时推出的产品为ProgressVersion8.3，现在已经到了10.2c版本。这是一套完善的集成开发工具、应用服务器和关系型数据库产品，提供了可扩充的多层Linux支持。Progress软件公司的Linux专用产品包括：

（1）Progress（r） App Server（tm）

这是一种可以在异构环境中部署共享应用组件的应用服务器。

（2）Progress（r） Enterprise RDBMS（tm）

为需要支持大型数据库、多处理器硬件和数千个并发用户的最苛刻的应用提供了一种可扩充的存储解决方案。

（3）ProgressVersion8.3部署产品

用于RedHat 6.0 Linux。

（4）ProgressVersion9部署产品

（5）Progress（r）Web Speed（r）Version3部署产品

（6）Progress（r）Apptivity（tm） 部署产品

（7）Progress（r）Sonic MQ（tm）部署产品

目前全球顶尖的汽车行业ERP供应商QAD支持最新的PROGRESS版本。

### 2.2 SQLite

轻量级别数据库SQLite的主要特点：

1. 支持事件，不需要配置，不需要安装，也不需要管理员；

2. 支持大部分SQL92；

3. 一个完整的数据库保存在磁盘上面一个文件，同一个数据库文件可以在不同机器上面使用，最大支持数据库到2T，字符和BLOB的支持仅限制于可用内存；

4. 整个系统少于3万行代码，少于250KB的内存占用（gcc），大部分应用比目前常见的客户端/服务端的数据库快，没有其它依赖

5. 源代码开放，代码95%有较好的注释，简单易用的API。官方带有TCL的编译版本。

### 2.3 Empress（商业数据库）

开发阶段特点：

1. 可嵌入程序，该特性使应用程序和数据库工作于统一地址空间，增强了系统的稳定性，提高了系统的效率。

2. 确定的响应时间，Empress 可以使数据的响应时间相对一致，使用者可以设定一个超时限制，如果在规定时间内没有完成插入，修改等操作，系统会报错。

3. 快速的操作Empress 提供了内核级的CAPI，称为MR， 用MR编写的应用程序在执行时不需要解析。另外在MR中加速的机制还包括优秀的加锁控制，内存管理和基于记录数量的选择功能。

4. 灵活的开发方式，Empress 提供多种开发接口，加快开发进程而无需开发者重新学习开发语言和熟悉开发环境。

5. 友好的存储方式，Empress 数据库可以放在操作系统支持的任何存储设备中，Empress的表单甚至可以分割放在不同的存储设备中，比如在内存，硬盘和CD-ROM中。

6. 微型内核结构 Empress 高度单元化， 可根据需要选择需要的单元，从而缩小产品中Empress 数据库所占用的资源。

7. 宽广的平台支持，Empress 支持多种硬件平台和软件平台， 也可移植到客户要求的硬件平台或操作系统。

技术优势：

1. 微型内核结构，占用少量内存空间，特别适合紧凑性的设计

2．一周7天，每天24小时连续工作，无需任何额外操作免维护

3. 内核级CAPI接口，使运行速度最大化

4. 高度灵活的SQL接口

5. 优秀的掉电恢复能力

6. 强壮的交易和锁存机制

7. 支持SCSI，RAID，IDE，RAM，CD-RW，DVD-ROM，CF，等存储介质

8. 支持Unicode码

9. 引擎可加载于磁盘和内存

### 2.4 eXtremeDB

eXtremeDB特点：

1. 内存数据库，eXtremeDB将数据以程序直接使用的格式保存在主内存之中，不仅剔除了文件I/O的开销，也剔除了文件系统数据库所需的缓冲和Cache机制。其结果是每个交易一微秒甚至更少的极限速度，相比于类磁盘数据库而言，速度成百上千倍地提高。作为内存数据库，eXtremeDB不仅性能高，而且数据存储的效率也非常高。为了提高性能并方便程序使用，数据在eXtremeDB中不做任何压缩，100M的空间可以保存高达70M以上的有效数据，这是其他数据库所不可想象的。

2. 混合数据库，eXtremeDB不仅可以建立完全运行在主内存的内存数据库，更可以建立磁盘/内存混合介质的数据库。在eXtremeDB，我们把这种建立在磁盘、内存或磁盘+内存的运行模式称为eXtremeDB Fusion融合数据库。eXtremeDB Fusion兼顾数据管理的实时性与安全性要求，是实时数据管理的台阶性进步。

3. 嵌入式数据库，eXtremeDB内核以链接库的形式包含在应用程序之中，其开销只有50KB~130KB。无论在嵌入式系统还是在实时系统之中，eXtremeDB都天然地嵌入在应用程序之中，在最终用户毫不知情的情况下工作。eXtremeDB的这种天然嵌入性对实时数据管理至关重要：各个进程都直接访问eXtremeDB数据库，避免了进程间通信，从而剔除了进程间通信的开销和不确定性。同时， eXtremeDB独特的数据格式方便程序直接使用的，剔除了数据复制及数据翻译的开销，缩短了应用程序的代码执行路径。

4. 由应用定制的API，应用程序对eXtremeDB数据库的操作接口是根据应用数据库设计而自动产生，不仅提升了性能，也剔除了通用接口所必不可少的动态内存分配，从而提高了应用系统的可靠性。定制过程简单方便，由高级语言定制eXtremeDB数据库中的表格、字段、数据类型、事件触发、访问方法等应用特征，通过eXtremeDB预编译器自动产生访问该数据库的C/C++ API接口。

5. 可预测的数据管理

eXtremeDB独特的体系结构，保证了数据管理的可预测性。eXtremeDB不仅更快、更小，而且更确定。在80双核CPU的服务器上，eXtremeDB在1TB内存里保存15B条记录；无论记录数多少，eXtremeDB可以在八十分之一微秒的时间内提取一条记录。

### 2.5 Firebird嵌入服务器版

Firebird嵌入服务器版（Embedded Server），从Interbase开源衍生出的Firebird，充满了勃勃生机。虽然它的体积比前辈Interbase缩小了几十倍，但功能并无阉割。为了体现Firebird短小精悍的特色，开发小组在增加了超级服务器版本之后，又增加了嵌入版本，最新版本为2.0。

Firebird的嵌入版有如下特色：

1、数据库文件与Firebird网络版本完全兼容，差别仅在于连接方式不同，可以实现零成本迁移。

2、数据库文件仅受操作系统的限制，且支持将一个数据库分割成不同文件，突破了操作系统最大文件的限制，提高了IO吞吐量。

3、完全支持SQL92标准，支持大部分SQL-99标准功能。

4、丰富的开发工具支持，绝大部分基于Interbase的组件，可以直接使用于Firebird。

5、支持事务、存储过程、触发器等关系数据库的所有特性。

6、可自己编写扩展函数（UDF）。

### 2.6 mSQL介绍

mSQL（mini SQL）是一个单用户数据库管理系统，个人使用免费，商业使用收费。由于它的短小精悍，使其开发的应用系统特别受到互联网用户青睐。mSQL（mini SQL）是一种小型的关系数据库，性能不是太好，对SQL语言的支持也不够完全，但在一些网络数据库应用中是足够了。由于mSQL较简单，在运行简单的SQL语句时速度比MySQL略快，而MySQL在线程和索引上下了功夫，运行复杂的SQL语句时比mSQL，PostgreSQL等都要快一些。最新版本是2005年5月8日发布的3.7.MSQL的标志是一个鹿。

mSQL的技术特点：安全性方面，mSQL通过ACL文件设定各主机上各用户的访问权限，缺省是全部可读/写。mSQL缺乏 ANSI SQL 的大多数特征，它仅仅实现了一个最最少的API，没有事务和参考完整性。mSQL与Lite（一种类似C的脚本语言，与分发一起发行）紧密结合，可以得到一个称为 W3-mSQL的一个网站集成包，它是JDBC、ODBC、Perl和PHP API。

### 2.7 Berkeley DB

技术特点：

1. Berkeley DB是一个开放源代码的内嵌式数据库管理系统，能够为应用程序提供高性能的数据管理服务。应用它程序员只需要调用一些简单的API就可以完成对数据的访问和管理。（不使用SQL语言）

2. Berkeley DB为许多编程语言提供了实用的API接口，包括C、C++、Java、Perl、Tcl、Python和PHP等。所有同数据库相关的操作都由Berkeley DB函数库负责统一完成。

3. Berkeley DB轻便灵活（Portable），可以运行于几乎所有的UNIX和Linux系统及其变种系统、Windows操作系统以及多种嵌入式实时操作系统之下。Berkeley DB被链接到应用程序中，终端用户一般根本感觉不到有一个数据库系统存在。

4. Berkeley DB是可伸缩（Scalable）的，这一点表现在很多方面。Database library本身是很精简的（少于300KB的文本空间），但它能够管理规模高达256TB的数据库。它支持高并发度，成千上万个用户可同时操纵同一个数据库。Berkeley DB能以足够小的空间占用量运行于有严格约束的嵌入式系统。

Berkeley DB在嵌入式应用中比关系数据库和面向对象数据库要好，有以下两点原因：

（1）因为数据库程序库同应用程序在相同的地址空间中运行，所以数据库操作不需要进程间的通讯。在一台机器的不同进程间或在网络中不同机器间进行进程通讯所花费的开销，要远远大于函数调用的开销；

（2）因为Berkeley DB对所有操作都使用一组API接口，因此不需要对某种查询语言进行解析，也不用生成执行计划，大大提高了运行效。

### 2.8 国产嵌入式数据库OpenBASE Lite

OpenBASE Lite是东软集团股份有限公司开发的嵌入式数据库产品。它是一个典型的轻量级数据库，定制的数据库引擎大小在250KB到600KB之间伸缩，可支持多种桌面操作系统、主流嵌入式系统平台及不同的处理器。作为一款功能全面的关系型数据库系统，OpenBASE Lite支持标准的SQL语法、ACID事务特性、备份/恢复等功能，提供了标准化开发接口JDBC、ODBC，能够在嵌入式环境下沿用关系数据库的经验继续来进行应用的开发。OpenBASE Lite提供了内存数据库运行模式，提供高速的数据访问与更新能力。

产品特色：

1.完善的数据管理功能

OpenBASE Lite嵌入式数据库具有完善的数据管理功能，提供了对SQL92标准子集的支持；提供对标准数据类型以及BLOB/CLOB类型的支持；支持数据库完整性控制；具有完整的数据管理能力，可以处理GB级的数据量；并提供对空间数据的管理能力。

2.广泛的平台通用性

OpenBASE Lite嵌入式数据库可运行于Windows 2000/2003/XP/Vista/Win 7、Windows Mobile 5&6、Windows CE、Linux、Embedded Linux、VxWorks、Symbian、Android等多种操作系统平台。

3.微小的核心内核

OpenBASE Lite嵌入式数据库具有微内核特性，可根据需求定制和裁剪，内核大小在250KB~600KB之间伸缩。

4.真正的零管理

在OpenBASE Lite嵌入式数据库的使用过程中无需对数据库进行配置，在移动终端应用中实现了“零管理”。

5.出色的处理性能

OpenBASE Lite嵌入式数据库可以作为内存数据库进行使用，实现了高速的数据访问与更新，单条数据处理时间不超过15us；在并发处理性能上，提供库级锁与表级锁并发访问控制，提高了进程与多线程对数据库并发访问处理性能。

6.充分的安全保障

OpenBASE Lite嵌入式数据库支持用户身份认证以及数据库对象的自主访问控制，可以有效防止用户数据的非法访问；支持128-bit AES存储加密，以保证数据库文件的安全性。

7.快速的故障恢复

提供了日志和故障恢复机制有效地保障了事务的ACID特性，另外还提供了数据库的联机热备与主从复制功能，使用户可以简单快速地提高应用系统的可靠性。

8.标准的访问接口

OpenBASE Lite嵌入式数据库为嵌入式应用的开发提供了C API接口以及JDBC/ODBC标准访问接口。

9.丰富的实用工具

OpenBASE Lite提供了丰富的实用工具进行数据库管理，包括数据库图形管理工具、建库工具、查询工具、备份恢复工具、导入导出工具及数据库加解密工具等。

## 3、SQLite常规SQL操作时间性能测试

### 3.1插入操作时间性能测试（优化）

1、环境

172.18.8.36

SQLite V3.7.13

2、方法

使用bind()、reset()优化方式，循环插入1000000条记录。

（1）磁盘文件模式，测试20次

（2）内存驻留模式，测试20次

3、参数

插入表格与对应数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试表格** | | | | | |
| **字段** | ID(int) | ParentID(int) | NAME(text) | GOAL(double) | NOTE(text) |
| **1** | 1 | 2 | Myname | 2.3 | Hello, nice to see you! |
| **2** | 2 | 3 | Myname | 2.3 | Hello, nice to see you! |
| **3** | 3 | 4 | Myname | 2.3 | Hello, nice to see you! |
| **……** | …… | …… | …… | …… | …… |
| **1000000** | 1000000 | 0 | Myname | 2.3 | Hello, nice to see you! |

insert into inserttest values(?, ?, ?, ?);

4、测试代码

4.1文件模式

/home/liangxx/learning/src/study\_sqlite/time\_test/insert\_test/insert\_test.cpp

**//---------------------------------------------------------------------**

**gettimeofday(&begin, NULL);**

**for (i = 1; i <= TESTTIME; i ++)**

**{**

**//bind**

**sqlite3\_bind\_int(stmt, 1, i);**

**sqlite3\_bind\_text(stmt, 2, insertname, strlen(insertname), SQLITE\_TRANSIENT);**

**sqlite3\_bind\_double(stmt, 3, 2.3);**

**sqlite3\_bind\_text(stmt, 4, insertnote, strlen(insertnote), SQLITE\_TRANSIENT);**

**while (sqlite3\_step(stmt) == SQLITE\_ROW);**

**//reset**

**sqlite3\_reset(stmt);**

**}**

**gettimeofday(&end, NULL);**

**//---------------------------------------------------------------------**

4.2内存模式

/home/liangxx/learning/src/study\_sqlite/time\_test/insert\_test/insert\_test\_mem.cpp

**//---------------------------------------------------------------------**

**gettimeofday(&begin, NULL);**

**for (i = 1; i <= TESTTIME; i ++)**

**{**

**//bind**

**sqlite3\_bind\_int(stmt, 1, i);**

**sqlite3\_bind\_text(stmt, 2, insertname, strlen(insertname), SQLITE\_TRANSIENT);**

**sqlite3\_bind\_double(stmt, 3, 2.3);**

**sqlite3\_bind\_text(stmt, 4, insertnote, strlen(insertnote), SQLITE\_TRANSIENT);**

**while (sqlite3\_step(stmt) == SQLITE\_ROW);**

**//reset**

**sqlite3\_reset(stmt);**

**}**

**gettimeofday(&end, NULL);**

**sqlite3\_finalize(stmt);**

**//---------------------------------------------------------------------**

5、测试结果

4.1文件模式（单位s）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3568 | 5098 | 4673 | 5823 | 5598 |
| 5950 | 3258 | 3330 | 5700 | 3498 |
| 3767 | 3158 | 4316 | 5612 | 5030 |
| 3352 | 5983 | 3723 | 3565 | 3783 |

平均：4439.25s

4.2内存模式（单位us）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7717460 | 7762232 | 7761384 | 6943695 | 7036881 |
| 9870924 | 8312949 | 7444938 | 8152426 | 7333887 |
| 7448746 | 8908814 | 8224034 | 7764799 | 7142657 |
| 7135006 | 7386375 | 7755169 | 7319889 | 6731809 |

平均值：7707703.7us

6、结论

文件模式：插入单条记录4439.25us

内存模式：插入单条记录7.71us

内存模式的插入速度是文件模式的576倍。

### 3.2查询操作时间性能测试（优化）

1、环境

172.18.8.36

SQLite V3.7.13

2、方法

使用bind()、reset()优化方式循环查询1000条记录，各测试20次。

（1）磁盘文件模式

（2）内存驻留模式

3、参数

查询：1000条数据

查询表格与对应数据

查询条件：ID%1000==0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试表格** | | | | | |
| **字段** | ID(int) | ParentID(int) | NAME(text) | GOAL(double) | NOTE(text) |
| **1** | 1 | 2 | Myname | 2.3 | Hello, nice to see you! |
| **2** | 2 | 3 | Myname | 2.3 | Hello, nice to see you! |
| **3** | 3 | 4 | Myname | 2.3 | Hello, nice to see you! |
| **……** | …… | …… | …… | …… | …… |
| **1000000** | 1000000 | 0 | Myname | 2.3 | Hello, nice to see you! |

select NOTE from inserttest where ID = ?;

4、测试代码

4.1文件模式

/home/liangxx/learning/src/study\_sqlite/time\_test/query\_test/query\_test.cpp

**//---------------------------------------------------------------------**

**gettimeofday(&begin, NULL);**

**for (i = 1; i <= queryline; i ++)**

**{**

**if (i % 1000 == 0)**

**{**

**sqlite3\_bind\_int(stmt, 1, i);**

**while (sqlite3\_step(stmt) == SQLITE\_ROW)**

**{**

**// id = sqlite3\_column\_int(stmt, 0);**

**// name = (unsigned char \*)sqlite3\_column\_text(stmt, 1);**

**// goal = sqlite3\_column\_double(stmt, 2);**

**note = (unsigned char \*)sqlite3\_column\_text(stmt, 0);**

**}**

**sqlite3\_reset(stmt);**

**}**

**else {}**

**}**

**gettimeofday(&end, NULL);**

**waste = (end.tv\_sec - begin.tv\_sec) \* 1000000 + (end.tv\_usec - begin.tv\_usec);**

**printf("the time is :%d us\n", waste);**

**//---------------------------------------------------------------------**

4.2内存模式

/home/liangxx/learning/src/study\_sqlite/time\_test/query\_test/query\_test\_mem.cpp

**//---------------------------------------------------------------------**

**sqlite3\_prepare(db, sql\_query, strlen(sql\_query), &stmt, 0);**

**register int count = 21;**

**while (-- count)**

**{**

**gettimeofday(&begin, NULL);**

**for (i = 1; i <= TESTTIME; i ++)**

**{**

**if (i % 1000 == 0) {**

**//bind**

**sqlite3\_bind\_int(stmt, 1, i);**

**while (sqlite3\_step(stmt) == SQLITE\_ROW)**

**{**

**note = (unsigned char \*)sqlite3\_column\_text(stmt, 0);**

**}**

**//reset**

**sqlite3\_reset(stmt);**

**}**

**}**

**gettimeofday(&end, NULL);**

**waste = (end.tv\_sec - begin.tv\_sec) \* 1000000 + (end.tv\_usec - begin.tv\_usec);**

**printf("The Waste Of Time Is:%2d ==== %d us\n", (20 - count), waste);**

**}**

5、测试结果

5.1文件模式（单位us）

48100 | 47604 | 30044 | 34647 | 35185 |

35820 | 41492 | 45622 | 41425 | 37621 |

41992 | 34869 | 47555 | 38538 | 54993 |

36499 | 46013 | 35592 | 46333 | 38144 |

平均值：40904.4us

5.2内存模式（单位us）

6655 | 6490 | 6401 | 6294 | 6341 |

6330 | 6284 | 6327 | 6040 | 6177 |

6564 | 6461 | 6380 | 6435 | 6351 |

6376 | 6172 | 6407 | 6386 | 6462 |

平均值：6366.65us

6、结论

文件模式：查询单条记录40.9044us

内存模式：查询单条记录6.36665us

内存模式速度是文件模式的6.42倍。

### 3.3删除操作时间性能测试（优化）

1、环境

172.18.8.36

SQLite V3.7.13

2、方法

使用bind()、reset()优化方式循环删除1000000条记录。

（1）磁盘文件模式：测试1次

（2）内存驻留模式：测试1次

3、参数

删除表格与对应数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试表格** | | | | | |
| **字段** | ID(int) | ParentID(int) | NAME(text) | GOAL(double) | NOTE(text) |
| **1** | 1 | 2 | Myname | 2.3 | Hello, nice to see you! |
| **2** | 2 | 3 | Myname | 2.3 | Hello, nice to see you! |
| **3** | 3 | 4 | Myname | 2.3 | Hello, nice to see you! |
| **……** | …… | …… | …… | …… | …… |
| **1000000** | 1000000 | 0 | Myname | 2.3 | Hello, nice to see you! |

delete from inserttest where ID = ?;

4、测试代码

4.1文件模式

/home/liangxx/learning/src/study\_sqlite/time\_test/delete\_test/delete\_test.cpp

**//---------------------------------------------------------------------**

**gettimeofday(&begin, NULL);**

**for (i = 1; i <= insertline; i ++)**

**{**

**if (i % 20000 == 0)**

**++ d;**

**sqlite3\_bind\_int(stmt, 1, i);**

**sqlite3\_step(stmt);**

**sqlite3\_reset(stmt);**

**}**

**gettimeofday(&end, NULL);**

**waste\_delete = (end.tv\_sec - begin.tv\_sec) \* 1000 + (end.tv\_usec - begin.tv\_usec) / 1000;**

**//---------------------------------------------------------------------**

4.2内存模式

/home/liangxx/learning/src/study\_sqlite/time\_test/delete\_test/delete\_test\_mem.cpp

//---------------------------------------------------------------------

sqlite3\_prepare(db, delet, strlen(delet), &stmt, 0);

**gettimeofday(&begin, NULL);**

**for (i = 1; i <= TESTTIME; i ++)**

**{**

**//bind**

**sqlite3\_bind\_int(stmt, 1, i);**

**while (sqlite3\_step(stmt) == SQLITE\_ROW);**

**//reset**

**sqlite3\_reset(stmt);**

**}**

**gettimeofday(&end, NULL);**

**waste = (end.tv\_sec - begin.tv\_sec) \* 1000000 + (end.tv\_usec - begin.tv\_usec);**

**printf("The Waste Of Delete Time Is: %ld us\n", waste);**

}

5、测试结果

5.1文件模式（单位ms）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5024727 | 6458030 | 4399107 | 7297804 | 4470264 |
| 5224969 | 4648675 | 5161535 | 6398206 | 7293629 |
| 7609287 | 7832012 | 5517289 | 7421823 | 6539289 |
| 6753921 | 5729162 | 4872191 | 6652191 | 5419283 |

平均值：6036169.70

5.2内存模式（单位us）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 9209465 | 7987816 | 10269509 | 7364937 | 8293791 |
| 6834946 | 8479504 | 7715321 | 8915214 | 8904720 |
| 8847275 | 7048427 | 8602543 | 7650822 | 8779752 |
| 8613658 | 7966454 | 7515766 | 8167565 | 8286321 |

平均值：8272690.3us

6、结论

文件模式：删除单条记录6.04ms

内存模式：删除单条记录8.27us

内存模式的删除速度是文件模式的729倍。

### 3.4递归操作时间性能测试（优化）

1、环境

172.18.8.36

SQLite V3.7.13

2、方法

使用bind()、reset()优化方式递归1000000条记录。

（1）磁盘文件模式：测试20次

（2）内存驻留模式：测试20次

3、参数

表格与对应数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **测试表格** | | | | | |
| **字段** | ID(int) | ParentID(int) | NAME(text) | GOAL(double) | NOTE(text) |
| **1** | 1 | 2 | Myname | 2.3 | Hello, nice to see you! |
| **2** | 2 | 3 | Myname | 2.3 | Hello, nice to see you! |
| **3** | 3 | 4 | Myname | 2.3 | Hello, nice to see you! |
| **……** | …… | …… | …… | …… | …… |
| **1000000** | 1000000 | 0 | Myname | 2.3 | Hello, nice to see you! |

select \* from inserttest where ID = ?;

4、测试代码

4.1文件模式

/home/liangxx/learning/src/study\_sqlite/time\_test/iter\_test/iter\_test.cpp

**//---------------------------------------------------------------------**

**gettimeofday(&begin, NULL);**

**for (i = 1; i <= TESTTIME; i ++)**

**{**

**sqlite3\_bind\_int(stmt, 1, i);**

**while (sqlite3\_step(stmt) == SQLITE\_ROW)**

**{**

**id = sqlite3\_column\_int(stmt, 0);**

**name = (unsigned char \*)sqlite3\_column\_text(stmt, 1);**

**goal = sqlite3\_column\_double(stmt, 2);**

**note = (unsigned char \*)sqlite3\_column\_text(stmt, 3);**

**}**

**sqlite3\_reset(stmt);**

**}**

**gettimeofday(&end, NULL);**

**waste = (end.tv\_sec - begin.tv\_sec) \* 1000000 + (end.tv\_usec - begin.tv\_usec);**

**printf("The iter\_test time is :%d us\n", waste);**

**//---------------------------------------------------------------------**

4.2内存模式

/home/liangxx/learning/src/study\_sqlite/time\_test/iter\_test/iter\_test\_mem.cpp

**//---------------------------------------------------------------------**

**sqlite3\_prepare(db, sql\_iter, strlen(sql\_iter), &stmt, 0);**

**gettimeofday(&begin, NULL);**

**for (i = 1; i <= TESTTIME; i ++)**

**{**

**//bind**

**sqlite3\_bind\_int(stmt, 1, i);**

**sqlite3\_bind\_text(stmt, 2, insertname, strlen(insertname), SQLITE\_TRANSIENT);**

**sqlite3\_bind\_double(stmt, 3, 2.3);**

**sqlite3\_bind\_text(stmt, 4, insertnote, strlen(insertnote), SQLITE\_TRANSIENT);**

**while (sqlite3\_step(stmt) == SQLITE\_ROW)**

**{**

**id = sqlite3\_column\_int(stmt, 0);**

**name = (unsigned char \*)sqlite3\_column\_text(stmt, 1);**

**goal = sqlite3\_column\_double(stmt, 2);**

**note = (unsigned char \*)sqlite3\_column\_text(stmt, 3);**

**}**

**//reset**

**sqlite3\_reset(stmt);**

**}**

**gettimeofday(&end, NULL);**

**waste = (end.tv\_sec - begin.tv\_sec) \* 1000000 + (end.tv\_usec - begin.tv\_usec);**

**printf("The iter\_test\_mem time is :%d us\n", waste);**

**//---------------------------------------------------------------------**

5、测试结果

5.1文件模式（单位us）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 40923878 | 47129810 | 37323136 | 38193470 | 42917329 |
| 39237846 | 42123189 | 39743290 | 39164898 | 43770312 |
| 44238724 | 36217378 | 41873202 | 40172386 | 39234703 |
| 43237923 | 38293013 | 47484739 | 41837429 | 39812379 |

平均值：41146451.7us

5.2内存模式（单位us）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4939718 | 5672393 | 4132345 | 4983762 | 4823689 |
| 5073892 | 4322389 | 4345675 | 5623868 | 5623979 |
| 4427181 | 4565232 | 5204198 | 6672513 | 6728265 |
| 4576239 | 6634424 | 5612890 | 4234622 | 5354132 |

平均值：5177570.3us

6、结论

文件模式：递归1000000记录41.15s

内存模式：递归1000000记录5.18s

内存模式的递归速度是文件模式的7.94倍。

### 3.5\*内存数据库、临时数据库与文件数据库性能对比

/home/liangxx/learning/src/study\_sqlite/mode\_configure/templib/

1、插入性能对比

内存数据库

**100：**1349 us、1348 us、1169 us、1517 us、1330 us

**1000：**4944 us、5364 us、6715 us、10098 us、8101 us

**10000：**55739 us、54711 us、54557 us、53846 us、53271 us

**100000：**497602 us、393154 us、429767 us、448017 us、425517 us

临时数据库

**100：**7646 us、4924 us、3126 us、3997 us、3277 us

**1000：**21981 us、30012 us、30870 us、30016 us、26871 us、28155 us、34483 us、36659 us

**10000：**227592 us、156978 us、178348 us、137131 us、161119 us、182308 us、442197 us

**100000：**1445486 us、1610571 us、2159278 us、1325063 us、1460177 us、1970883 us、1950561 us

文件数据库

**100：**317115 us、271684 us、277371 us、260661 us、322307 us

**1000：**2805561 us、3775442 us、2833744 us、2779821 us、2910210 us

**10000：**29670339 us、31132331 us、28644974 us、30834055 us、31973082 us

**100000：**325791831 us、321693665 us、338502059 us、351378342 us、673694482 us、951382463 us、980610979 us、924399383 us

## 4、mSQL常规SQL操作时间性能测试

### 4.1插入操作

本地UNIX SOCKET MODE：

**（1）测试代码**

// msql\_api\_cpp

#include <stdio.h>

#include <stdlib.h>

#include <memory.h>

#include <sys/time.h>

#include "msql.h"

// test times

const int TESTTIME = 1000000;

// primary key display width

const int PKWIDTH = 8;

// time

struct timeval begin;

struct timeval end;

long int waste;

int sql\_proc(char \* sql, int i)

{

int offset = 0;

char buf[PKWIDTH];

memset(sql, 0, 1024);

offset += sprintf(sql + offset, "%s", "insert into inserttest values(");

offset += sprintf(sql + offset, "%s", gcvt((double)i, PKWIDTH, buf));

offset += sprintf(sql + offset, "%s", ", 'David', 'Smith', 'Development')");

//printf("%s\n", sql);

return offset;

}

int main()

{

// var

int fd;

int rc;

register int i;

m\_result\* pr;

m\_row row;

char sql[1024];

// connect the local server running mSQL server

rc = msqlConnect(NULL);// Local Unix Socket Mode

// rc = msqlConnect(“172.18.8.36”);// Local TCP/IP Socket Mode

if (rc == -1)

{

printf("msqlConnect: %s\n", msqlErrMsg);

return -1;

}

fd = rc;

// choose a db

rc = msqlSelectDB(fd, "test");

if (rc == -1)

{

printf("msqlSelectDB: %s\n", msqlErrMsg);

return -1;

}

// create table

rc = msqlQuery(fd, "create table inserttest(emp\_id int not null, first\_name char(15) not null, last\_name char(15) not null, dept char(20))");

if (rc == -1)

{

printf("msqlQuery(create table): %s\n", msqlErrMsg);

return -1;

}

**// insert**

**gettimeofday(&begin, NULL);**

**for (i = 0; i < TESTTIME; i ++)**

**{**

**sql\_proc(sql, i);**

**rc = msqlQuery(fd, sql);**

**if (rc == -1)**

**{**

**printf("msqlQuery(insert into): %s\n", msqlErrMsg);**

**return -1;**

**}**

**if (i % 10000 == 0)**

**printf("insert->%10d\n", i);**

**}**

**gettimeofday(&end, NULL);**

**waste = (end.tv\_sec - begin.tv\_sec) \* 1000000 + (end.tv\_usec - begin.tv\_usec);**

**printf("Insert %ld us\n", waste);**

// close db

msqlClose(fd);

return 0;

}

// vim: set tabstop=4 shiftwidth=4 expandtab:

**（2）测试结果**

Unix Socket: 43932551 us

TCP/IP Socket: 50498947 us

## 5、SQLite与mSQL的对比

### 5.1对SQL的支持

**SQLite不支持ANSI SQL92的部分特性**

（1）右外连接与全外连接

支持左外连接LEFT OUTER JOIN，但是还不支持右外连接RIGHT OUTER JOIN和全外连接FULL OUTER JOIN。

（2）大部分的修改表操作

只有RENAME TABLE和ADD COLUMN这两种ALTER TABLE命令被支持。其它ALTER TABLE操作比如DROP COLUMN、ALTER COLUMN、ADD CONSTRAINT未被支持。

对于COUNT（DISTINCT column），SQLite在执行如下语句的时候会报错：

SELECT COUNT(DISTINCT watchid) FROM [watch\_item] WHERE watch\_item.watchid = 1;

其原因是SQLite的所有内置函数都不支持DISTINCT限定，所以如果要统计不重复的记录数的时候会出现一些麻烦。比较可行的做法是先建立一个不重复的记录表的视图，然后再对该视图进行计数。

（3）触发器

支持FOR EACH ROW触发器，不支持FOR EACH STATEMENT触发器。

（4）视图的创建与改写

SQLite在创建多表视图的时候有一个BUG，问题如下：

CREATE VIEW watch\_single AS SELECT DISTINCT watch\_item.[watchid],watch\_item.[itemid] FROM watch\_item;

上面这条SQL语句执行后会显示成功，但是实际上除了

SELECT COUNT(\*) FROM [watch\_single ] WHERE watch\_ single.watchid = 1;

能执行之外是无法执行其他任何语句的。其原因在于建立视图的时候指定了字段所在的表名，而SQLite并不能正确地识别它。所以上面的创建语句要改为：

CREATE VIEW watch\_single AS SELECT DISTINCT [watchid],[itemid] FROM watch\_item;

但是随之而来的问题是如果是多表的视图，且表间有重名字段的时候就会出现问题。

SQLite中的视图是只读的，对视图不能执行DELETE、INSERT或UPDATE命令。但是可以创建一个触发器试图DELETE、INSERT或UPDATE一个视图。

（5）授权与撤销

因为SQLite读取和写入一个普通磁盘文件，唯一的访问许可对于底层操作系统的一般文件有效。GRANT和REVOKE命令（通常见于CS RDBMSes） 不被支持，它们对于嵌入式数据库引擎无意义。

**mSQL语言提供ANSI SQL子集**

mSQL允许程序和用户对表格结构的数据进行操作。它不支持一些关联型操作，比如视图和嵌套查询。

### 5.2支持的语言接口

（1）SQLite语言接口包括较为完整的C编程接口，具有TCL绑定版本。

（2）mSQL的具有基础的C编程接口：查询相关函数（基本查询操作）、模式相关函数、时间日期相关函数、其它杂类函数。但是相对于SQLite C API功能有限，缺乏primary key、主键自增功能，需要自己编程实现，操作性能不佳（本地插入操作较好，但是删除及递归速度太慢）