****

**中数通信息有限公司内部参考资料**

**嵌入式数据库研究报告**

编制单位：中数通信息有限公司

编制日期：2012年10月

**声明**

本文档所有权和解释权归中数通信息有限公司所有，未经书面许可，不得复制或向第三方公开。

This document is the property of China DataCom Corporation Limited and can be neither reproduced nor disclosed to a third party without a written authorization.

**修订记录**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **日期** | **AMD** | **修订者** | **说明** |
| 0.1 | 2012-10-08 | A | 梁夏旭 | 建立空白文档 |
| 0.2 | 2012-10-19 | A | 梁夏旭 | 初稿 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

（A-添加，M-修改，D-删除）

**目录**

第一章 嵌入式数据库概况 6

嵌入式数据库综述 6

常用的嵌入式数据库的介绍 6

Progress 6

SQLite 7

Empress 7

eXtremeDB 8

Firebird 9

mSQL 10

BerkeleyDB 10

OpenBASE Lite 11

第二章 轻量级的SQLite 12

SQLite的简介 12

SQLite的特性 12

SQLite的下载 13

SQLite安装 14

SQLite初级SQL 15

建立数据库 17

建表 17

插入数据 21

查询 24

删除数据 25

删表 26

SQLite关键字 27

SQLite高级SQL（事务性与锁机制） 28

事务 28

冲突解决 30

SQLite锁 32

SQLite的编程接口 34

C/C++接口 34

第三章 SQLite性能分析与应用建议 51

SQLite文件模式与内存模式性能测试 51

插入操作时间性能测试（绑定查询） 51

查询操作时间性能测试（绑定查询） 53

删除操作时间性能测试（绑定查询） 55

递归操作时间性能测试（绑定查询） 57

SQLite文件模式性能曲线（稳定性测试） 58

测试表格 59

测试方法 59

数据记录 59

结论 65

SQLite内存模式性能曲线（稳定性测试） 68

测试表格 68

测试方法 69

数据记录 69

结论 73

关于SQLite的应用建议 76

SQLite的优势 76

SQLite的劣势 76

应用建议 77

图表索引 78

# 第一章 嵌入式数据库概况

## 嵌入式数据库综述

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，避免了与客户端服务端（CS）配置相关的开销，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）。

可靠性要求对于嵌入式数据库是必须的，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。

## 常用的嵌入式数据库的介绍

### Progress

<http://www.progress.com/en/products/allproducts.html>

Progress软件公司2000年全面发售在Linux操作系统上运行的数据库及其部署产品。

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

### 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（同时受操作系统磁盘文件大小限制），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，大部分应用比目前常见的客户端服务端的数据库快。

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。

### Empress

Empress是一款商业数据库，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C语言API，称为MR（micro relation）， 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。Empress对关键功能用MR语句实现，对变动较大功能用SQL语句实现。

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. 可以7×24小时连续工作，无需任何额外操作免维护。

3. 内核级C语言API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。

### 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。其结果是每个交易耗时一微秒甚至更少，接近极限速度，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。

### Firebird

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。

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SQL99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）。

### mSQL

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。它短小精悍，用它开发的应用系统特别受互联网用户青睐。mSQL小型，性能不太好，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。

mSQL的技术特点：

1、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。

2、mSQL缺乏ANSI SQL的大多数特征，它仅实现一个最少的API，没有事务和参考完整性。

3、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，是JDBC、ODBC、Perl和PHP API。

### BerkeleyDB

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，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。

### 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。

# 第二章 轻量级的SQLite

## SQLite的简介

官网：<http://www.sqlite.org/>

## SQLite的特性

（1）独立self-contained

SQLite仅需调用外部少量的C函数，包括：memset()、memcpy()、memcmp()、strcmp()、malloc()、free()及realloc()

SQLite包含amalgamation作为独立的C函数库，包括"sqlite3.c"和"sqlite3.h"。

编译时链接共享库"libsqlite3.so"：-lsqlite3

（2）无服务器serverless

（3）零配置zero-configuration

（4）事务性transactional

ACID特性：原子的Atomic、连续的Consistent、独立的Isolated、持久的Durable 。

（5）开源的open-source

## SQLite的下载

下载地址：<http://www.sqlite.org/download.html>

SQLite组件

（1）amalgamation（SQLite可用的C源码源文件）

（2）SQLite说明文档

（3）Command Line Shell for SQLite（SQLite命令行组件：Unix下的sqlite3及Windows下的sqlite3.exe）

（4）SQLite Analysis Program（SQLite分析组件）

安装包命令规则如下：

（1）通用源码包：sqlite-product-version.zip

（2）仅限于类Unix平台源码：sqlite-product-version.tar.gz

（3）预编译二进制包，是平台相关的：sqlite-product-os-cpu-version.zip

（4）非官方的预览版源码：sqlite-product-date.zip

下载的安装包

说明文档：sqlite-doc-3071300.zip

Unix平台源码包（包含amalgamation）：

（1）自动配置安装包：sqlite-autoconf-3071300.tar.gz

或预编译包：sqlite-shell-linux-x86-3071300.zip及sqlite-analyzer-linux-x86-3071300.zip

Windows平台安装包：

（1）命令行shell：sqlite-shell-win32-x86-3071300.zip

（2）动态链接库：sqlite-dll-win32-x86-3071300.zip

（3）分析程序：sqlite-analyzer-win32-x86-3071300.zip

## SQLite安装

Linux平台源码包安装

（1）解压

（2）阅读README，注意如下四点内容

1）源码包内容：amalgamation（单文件）、shell.c文件用于生成sqlite3 shell、sqlite3.h和sqlite3ext.h头文件用于链接程序和SQLite扩展。

2）autoconf/automake安装结构

3）涉及的功能：

--enable-readline # shell工具 [default=yes]

--enable-threadsafe # 建立线程安全库 [default=yes]

--enable-dynamic-extensions # 支持加载扩展 [default=yes]

4）修改编译参数，比如：

$ CFLAGS="-Os" ./configure

$ CFLAGS="-Os -DSQLITE\_OMIT\_TRIGGERS" ./configure

（3）阅读INSTALL文件

安装步骤：

$ ./configure

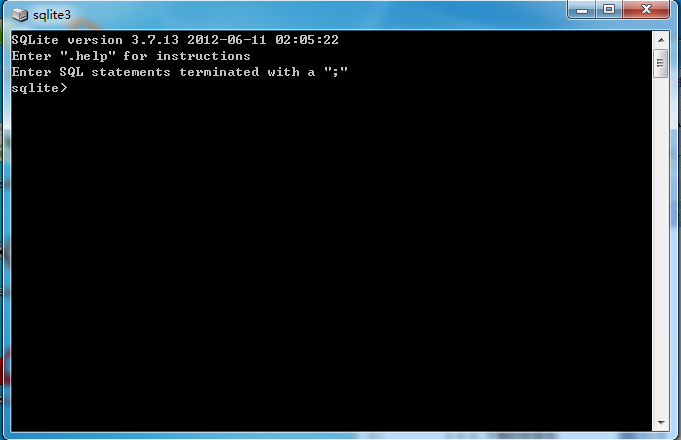
$ make

$ make install

安装后就可以执行sqlite3命令

Windows平台安装

（1）将sqlite-shell-win32-x86-3071300.zip解压出来的sqlite3.exe就是命令行shell，双击或在命令行窗口运行即可进入sqlite命令行界面。



图表 1 SQLite CLP

## SQLite初级SQL

SQLite shell命令行涉及自身命令和SQL命令，自身命令以“.”开头，SQL命令以“;”结束。

一般数据采用固定的静态数据类型，而SQLite采用的是动态数据类型，会根据存入值自动判断。（即使数据项声明为A类型，如果我们输入的数据类型是B，SQLite也可以进行自动转换。）

SQLite支持五种存储类型：

（1）NULL：空值。

（2）INTEGER：带符号的整型，具体取决有存入数字的范围大小。

（3）REAL：浮点数字，存储为8-byte IEEE浮点数。

（4）TEXT：字符串文本。

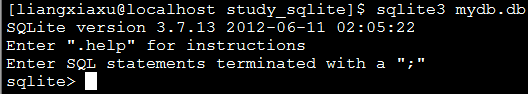
（5）BLOB：二进制对象。

实际上，sqlite3接受如下的数据类型。

|  |  |
| --- | --- |
| smallint | 16位元的整数。 |
| interger | 32位元的整数。 |
| decimal(p,s) | p精确值和s大小的十进位整数，精确值p是指全部有几个数（digits）大小值，s是指小数点后有几位数。如果没有特别指定，则系统会设为p=5; s=0。 |
| float | 32位元的实数。 |
| double | 64位元的实数。 |
| char(n) | n长度的字串，n不能超过254。 |
| varchar(n) | 长度不固定且其最大长度为n的字串，n不能超过4000。 |
| graphic(n) | 和char(n)一样，不过其单位是两个字元double-bytes，n不能超过127。这个形态是为了支援两个字元长度的字体，例如中文字。 |
| vargraphic(n) | 可变长度且其最大长度为n的双字元字串，n不能超过2000。 |
| date | 包含了年份、月份、日期。 |
| time | 包含了小时、分钟、秒。 |
| timestamp | 包含了年、月、日、时、分、秒、千分之一秒。 |
| datetime | 包含日期时间格式，必须写成'2010-08-05'不能写为'2010-8-5'，否则在读取时会产生错误。 |

### 建立数据库

$ sqlite3 mydb.db



图表 2 SQLite CLP

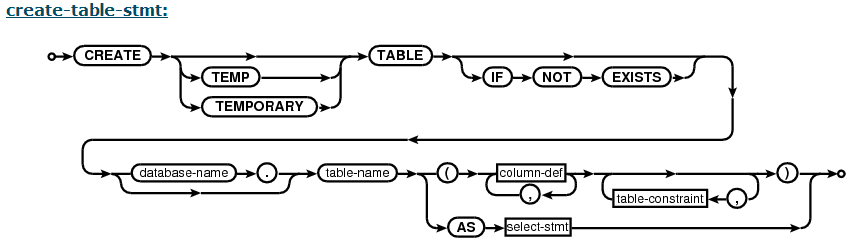
新建数据库mydb.db，若未对该数据进行建表操作（建立非临时表），退出该数据库后，mydb.db也将被删除。

### 建表

#### 语法

（1）关于表名

表名sqlite\_\*保留作内部使用，用户无法创建名为sqlite\_\*的数据表。



图表 3 Create-table-stmt

（2）temp字段

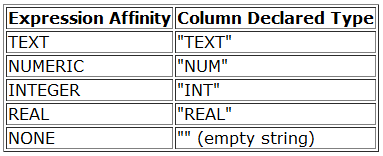
create与table之间加上temp或temporary表明数据表建立在数据库temp中，若加上关联的数据库即表明数据表建立在该关联数据库上，同时加上temp和关联数据库名是错误的语法，若没有加上任何数据库名，即将数据表建立在main数据库上。

（3）if not exists字段

table后的if not exists用于判别是否存在同名数据表存在，若存在同名数据表则该建表命令无效，并且不会返回错误信息，若没有添加if not exists，建表名与已存在表名冲突会导致报错。

（4）create table ……as select ……

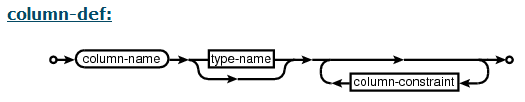
将根据select的结果建表，行列一致，命名一致，列数据项的数据类型将采用类似表述：



图表 4 Expression affinity

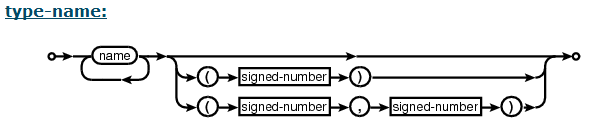
create table ……as select ……新建的表没有设置主键、约束关系，默认值为空。初始值为select返回的值，排序按照select结果。

（5）定义列（数据项）



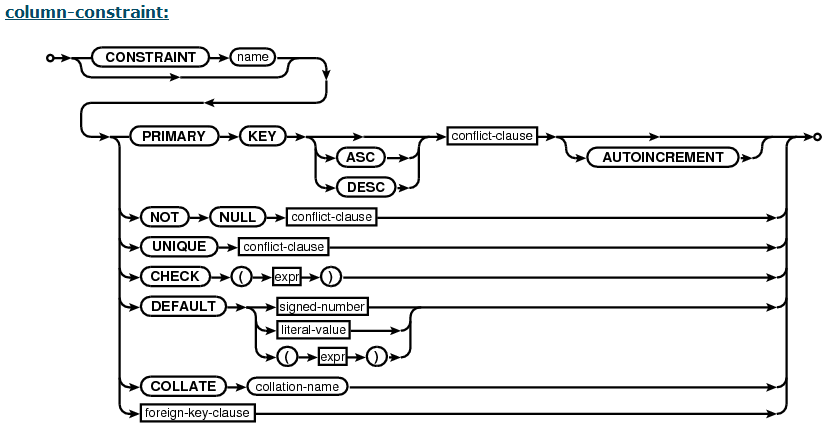
图表 5 Column-def

数据项名+类型名+约束条件



图表 6 Type-name

（6）6种数据约束



图表 7 Column-constraint

1、primary约束

SQLite允许主键为空值，这一点需要用户自己注意，不要给主键设置空值。

2、unique约束

与主键primary约束类似，但是一个表允许多个unique值。这个约束保证了该行数据的该列值在该表是唯一的。

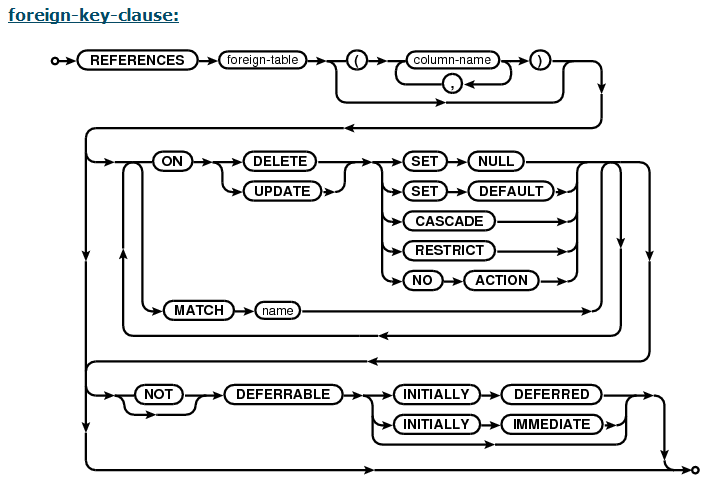
3、integer primary key

4、check约束

用于限定某数据项的值的范围。

5、not null约束

6、foreign key



图表 8 Foreign-key-clause

（7）ROWIDs与INTEGER PRIMARY KEY

#### 操作实例

统一配置：

sqlite> .mode column

.mode column

sqlite> .headers on

.headers on

sqlite> .width 12 12 12 12 12 12

|  |  |  |
| --- | --- | --- |
| ID | NAME | GOAL |
|  |  |  |

图表 9 courses（空）

sqlite> **create table** courses(ID integer primary key, NAME text not null unique, GOAL integer not null);

sqlite> .tables

courses

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | GENDER | AGE | COURSE |
|  |  |  |  |  |

图表 10 Students（空）

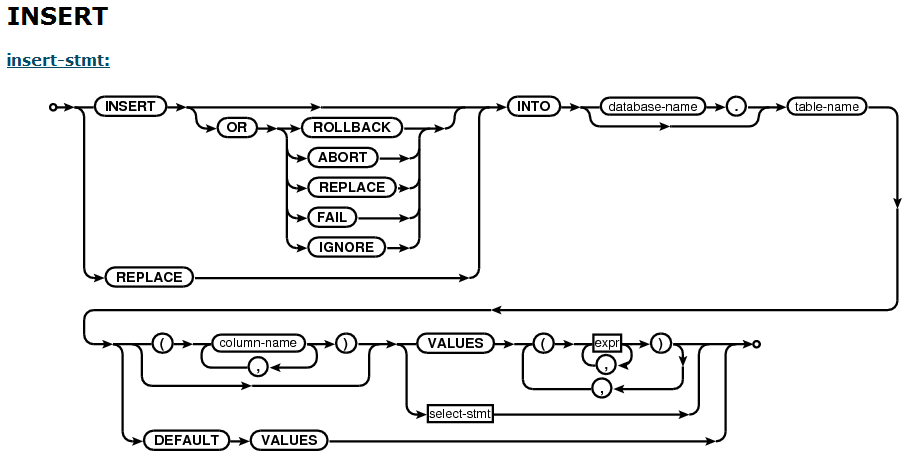
sqlite> **create table** students(ID integer primary key not null, NAME text not null, GENDER text not null, AGE integer not null, COURSE integer, foreign key(COURSE) references courses(ID));

sqlite> .table

courses students

### 插入数据

#### 语法



图表 11 Insert-stmt

#### 操作实例

|  |  |  |
| --- | --- | --- |
| ID | NAME | GOAL |
| 1 | 政治 | 5 |
| 2 | 经济 | 3 |
| 3 | 文学 | 4 |
| 4 | 逻辑 | 5 |
| 5 | 美容 | 6 |
| 6 | 武术 | 2 |

图表 12 courses

sqlite> **insert into** courses **values**(1, '政治', 5);

sqlite> **insert into** courses **values**(2, '经济', 3);

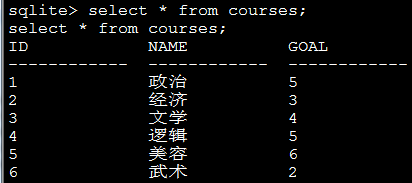
sqlite> **insert into** courses **values**(3, '文学', 4);

sqlite> **insert into** courses **values**(4, '逻辑', 5);

sqlite> **insert into** courses **values**(5, '美容', 6);

sqlite> **insert into** courses **values**(6, '武术', 2);

sqlite> select \* from courses;



图表 13 查询courses

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | NAME | GENDER | AGE | COURSE |
| 1 | 关羽 | Male | 11 | 6 |
| 2 | 郭美美 | Female | 22 | 5 |
| 3 | 奥巴马 | Male | 66 | 1 |
| 4 | Conan | Male | 44 | 4 |
| 5 | 方舟子 | Male | 88 | 3 |

图表 14 students

sqlite> **insert into** students **values**(1, '关羽', 'Male', 11, 6);

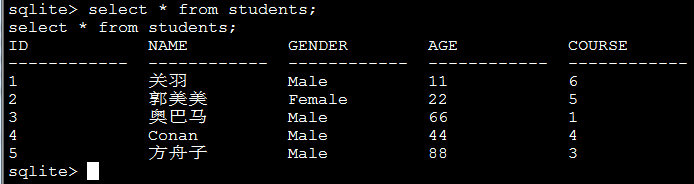
sqlite> **insert into** students **values**(2, '郭美美', 'Female', 22, 5);

sqlite> **insert into** students **values**(3, '奥巴马', 'Male', 66, 1);

sqlite> **insert into** students **values**(4, 'Conan', 'Male', 44, 4);

sqlite> **insert into** students **values**(5, '方舟子', 'Male', 88, 3);

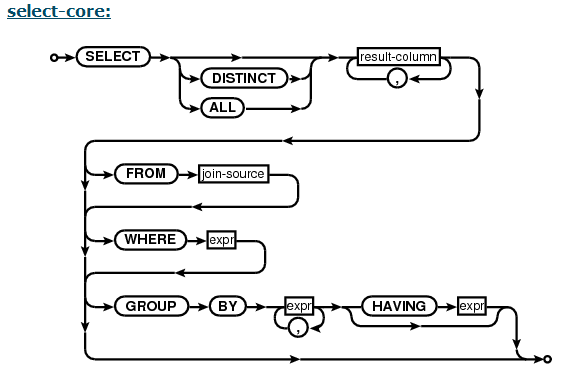
sqlite> select \* from students;



图表 15 查询students

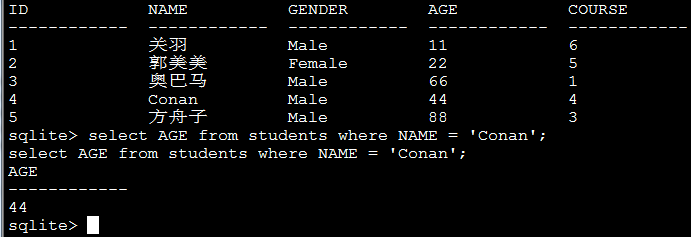
### 查询

#### 语法



图表 16 Select-core

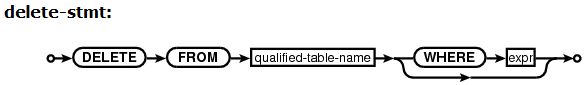
#### 操作实例



图表 17 查询

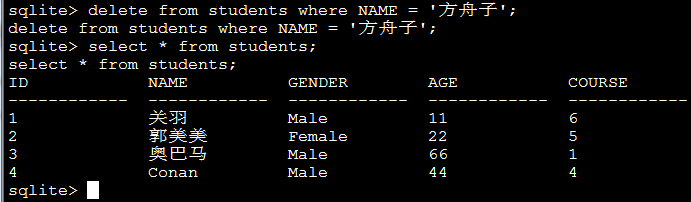
### 删除数据

#### 语法



图表 18 Delete-stmt

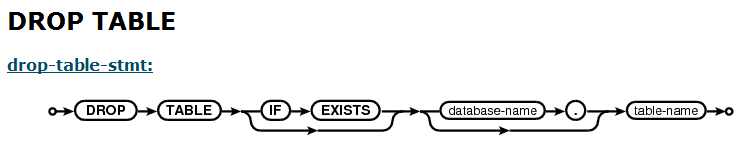
#### 操作实例



图表 19 Delete一行

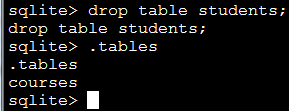
### 删表

#### 语法



图表 20 Drop-table-stmt

#### 操作实例



图表 21 Drop

### SQLite关键字

ABORT、ACTION、ADD、AFTER、ALL、ALTER、ANALYZE、AND、AS、ASC、ATTACH、AUTOINCREMENT、

BEFORE、BEGIN、BETWEEN、BY、

CASCADE、CASE、CAST、CHECK、COLLATE、COLUMN、COMMIT、CONFLICT、CONSTRAINT、CREATE、CROSS、CURRENT\_DATE、CURRENT\_TIME、CURRENT\_TIMESTAMP

DATABASE、DEFAULT、DEFERRABLE、DEFERRED、DELETE、DESC、DETACH、DISTINCT、DROP

EACH、ELSE、END、ESCAPE、EXCEPT、EXCLUSIVE、EXISTS、EXPLAIN、

FAIL、FOR、FOREIGN、FROM、FULL、

GLOB、GROUP

HAVING

IF、IGNORE、IMMEDIATE、IN、INDEX、INDEXED、INITIALLY、INNER、INSERT、INSTEAD、INTERSECT、INTO、IS、ISNULL、

JOIN

KEY

LEFT、LIKE、LIMIT

MATCH

NATURAL、NO、NOT、NOTNULL、NULL、

OF、OFFSET、ON、OR、ORDER、OUTER、

PLAN、PRAGMA、PRIMARY、

QUERY、

RAISE、REFERENCES、REGEXP、REINDEX、RELEASE、RENAME、REPLACE、RESTRICT、RIGHT、ROLLBACK、ROW、

SAVEPOINT、SELECT、SET、

TABLE、TEMP、TEMPORARY、THEN、TO、TRANSACTION、TRIGGER、

UNION、UNIQUE、UPDATE、USING、

VACUUM、VALUES、VIEW、VIRTUAL、

WHEN、WHERE

## SQLite高级SQL（事务性与锁机制）

### 事务

SQLite事务涉及到的CLP命令：begin、commit、rollback

Begin——开始一项事务

Commit——确认一项事务

Rollback——回滚本事务

一项事务以begin和commit作为分隔，默认情况下，SQLite单条SQL语句自成事务（自动提交），如图所示。



图表 22单条SQL语句自成事务

由begin和commit之间的一组操作组成一项事务，事务性保证了一项操作或一组操作，要么100%完成，要么全部没有执行。

示例：

begin；

delete from test where id = 5;

rollback;

delete from test where goal = 88;

commit;



图表 23Begin-rollback-commit

### 冲突解决

冲突解决涉及的CLP命令：replace、ignore、fail、abort、rollback。

|  |  |
| --- | --- |
| **解决方案** | **说明** |
| **Replace** | 违反的记录被删除，以新记录代替之 |
| **Ignore** | 违反的记录保持原貌，其它记录继续执行 |
| **Fail** | 终止命令，违反之前执行的操作得到保存 |
| **Abort** | 终止命令，恢复违反之前执行的修改 |
| **Rollback** | 终止命令和事务，回滚整个事务 |

图表 24冲突解决CLP命令



图表 25冲突解决CLP命令的性质排序

冲突解决语法

1.语句级（可覆盖对象级的冲突解决手段）

insert/update/create or [resolution] table/index [tbl\_name/idx\_name] ……

关键词or后接五种冲突解决手段之一

2.对象级（定义表格时）

create table/index [tbl\_name/idx\_name] ([field\_name] [format] [constraint] on conflict [resolution]);

constraint：unique、not null……

示例：

create temp table cast (name text unique on conflict rollback);

insert into cast values (‘Jerry’);

insert into cast values (‘Bean’);

insert into cast values (‘Android 4.1’);

begin;

insert into cast values (‘Jerry’);

:uniqueness constraint failed

commit;

:cannot commit – no transaction is active

begin;

insert or replace into cast values (‘Jerry’);

commit;

### SQLite锁

SQLite提出事务的五种状态：unlocked、shared、reserved、pending、exclusive

|  |  |
| --- | --- |
| **锁** | **说明** |
| **Unlocked** | 未加锁：  未对数据库进行访问（读写）之前的状态 |
| **Shared** | 共享锁：  对数据库进行读操作，一个数据库允许多个共享锁共存 |
| **Reserved** | 预留锁：  对数据库进行写（缓存）操作，一个数据库只允许一个预留锁 |
| **Pending** | 未决锁：  等待其它共享锁关闭 |
| **Exclusive** | 排它锁：  将缓存中的操作提交到数据库 |

图表 26五种状态



图表 27五种状态的切换状态机

对SQLite的多连接操作容易引发死锁。死锁的原因是拥有共享锁的连接与拥有未决锁的连接都不想放弃断开连接，如下例子所示。



图表 28死锁示例

SQLite提供预防死锁的三种事务类型：deferred、immediate、exclusive。

|  |  |
| --- | --- |
| **事务类型** | **说明** |
| **Deferred** | 直到必须使用时才获取锁 |
| **Immediate** | 在begin执行时试图获取预留锁 |
| **Exclusive** | 试图获取排它锁 |

图表 29预防死锁的三种事务类型

SQLite根据不同的事务类型，以不同的锁状态启动事务（而不是以未加锁状态启动），事务类型在begin命令中指定：

begin [deferred | immediate | exclusive] transaction;

为预防死锁，操作SQLite必须遵守的基本准则是多人对同一数据库进行操作时，必须共同遵守如下准则：

使用

begin immediate | exclusive transaction;

而不是

begin deferred transaction;

或单纯的

begin;

## SQLite的编程接口

### C/C++接口

<http://www.sqlite.org/c3ref/intro.html>

SQLite拥有超过185个明确的API

#### 两个重要的对象（并非面向对象编程中的术语对象）

<http://www.sqlite.org/cintro.html>

（1）database connection

数据库连接对象sqlite3，即数据库名。

（2）prepared statement

预声明对象sqlite3\_stmt，简单理解就是SQL操作语句。

#### C/C++接口程序（分步查询）

数据库连接对象sqlite3和预声明对象sqlite3\_stmt由下面少量的C/C++接口程序（程序族）组合实现：

（1）sqlite3\_open()

（2）sqlite3\_prepare()

（3）sqlite3\_step()

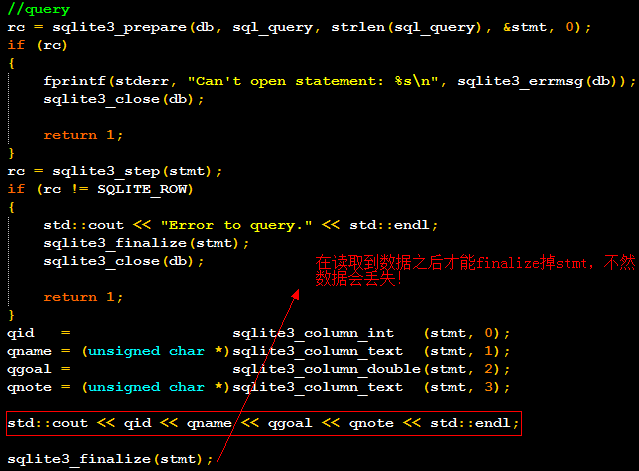
（4）sqlite3\_column()

（5）sqlite3\_finalize()

（6）sqlite3\_close()

以上六个接口是概念接口，实际调用时的函数名也许跟上面有出入。

**注意：sqlite3\_open()要与sqlite3\_close()配对；sqlite3\_prepare()要与sqlite3\_finalize()配对（否则可能会导致事务回滚）。**

****

图表 30 接口匹配

**sqlite3\_open()**

作为database connection对象的构造函数，连接一个数据库文件，并返回一个database connection对象。包括sqlite3\_open、sqlite3\_open16、sqlite3\_open\_v2。

int sqlite3\_open(

const char \*filename, /\* Database filename (UTF-8) \*/

sqlite3 \*\*ppDb /\* OUT: SQLite db handle \*/

);

参数1：数据库名

参数2：返回数据库句柄

成功返回 SQLITE\_OK

**sqlite3\_prepare()**

将SQL文本（即字符串格式）转换为prepared statement对象（byte-code格式），并返回一个指向该对象的指针。前提是存在一个由sqlite3\_open()产生的指向database connection对象的指针和SQL文本。该接口并不执行SQL文本上的命令，而只是作转换，为执行做好准备。对于不支持sqlite3\_prepare()的新应用，采用sqlite3\_prepare\_v2()作替代。包括sqlite3\_prepare、sqlite3\_prepare\_v2、sqlite3\_prepare16、sqlite3\_prepare16\_v2。

int sqlite3\_prepare\_v2(

sqlite3 \*db, /\* Database handle \*/

const char \*zSql, /\* SQL statement, UTF-8 encoded \*/

int nByte, /\* Maximum length of zSql in bytes. \*/

sqlite3\_stmt \*\*ppStmt, /\* OUT: Statement handle \*/

const char \*\*pzTail /\* OUT: Pointer to unused portion of zSql \*/

);

参数1：数据库句柄

参数2：SQL命令字符串

参数3：读入的SQL命令字符串的最长字节数（当字符串长于nByte时读取字符串的nByte字节，当字符串小于等于nByte时读取到字符串的结束）

参数4：返回命令句柄

参数5：指向SQL字符串未可用的剩余字符串的首字节，一般设置为0

成功返回 SQLITE\_OK

**sqlite3\_step()**

执行SQL命令，得到第一行可用结果，若存在多行结果，则多次调用sqlite3\_step()，对于不返回结果的SQL命令，则只执行一次sqlite3\_step()。

int sqlite3\_step(sqlite3\_stmt\*);

参数1：SQL命令句柄

返回结果：

SQLITE\_BUSY：无法响应。

SQLITE\_DONE：执行完成，不可再继续调用sqlite3\_step()，直到执行sqlite3\_reset()后才能继续调用sqlite3\_step()，主要针对insert、delete等操作。

SQLITE\_ROW：执行完一次得到一行结果，需要继续调用sqlite3\_step()以得到下一行结果，主要针对query等操作。

SQLITE\_ERROR：执行错误。

SQLITE\_MISUSE：执行不恰当，可能是句柄已经被注销、之前已经报过错或同名数据库被两个线程同时调用。

**sqlite3\_column()**

读取sqlite3\_step()每一次得到的一行结果的每一列内容，sqlite3\_column()只是一个概念，它包含如下接口家族（针对不同的数据类型设计）：

sqlite3\_column\_blob()、sqlite3\_column\_bytes()、sqlite3\_column\_bytes16()、sqlite3\_column\_count()、sqlite3\_column\_double()、sqlite3\_column\_int()、sqlite3\_column\_int64()、sqlite3\_column\_text()、sqlite3\_column\_text16()、sqlite3\_column\_type()、sqlite3\_column\_value()

const void \*sqlite3\_column\_blob(sqlite3\_stmt\*, int iCol);

int sqlite3\_column\_bytes(sqlite3\_stmt\*, int iCol);

int sqlite3\_column\_bytes16(sqlite3\_stmt\*, int iCol);

double sqlite3\_column\_double(sqlite3\_stmt\*, int iCol);

int sqlite3\_column\_int(sqlite3\_stmt\*, int iCol);

sqlite3\_int64 sqlite3\_column\_int64(sqlite3\_stmt\*, int iCol);

const unsigned char \*sqlite3\_column\_text(sqlite3\_stmt\*, int iCol);

const void \*sqlite3\_column\_text16(sqlite3\_stmt\*, int iCol);

int sqlite3\_column\_type(sqlite3\_stmt\*, int iCol);

sqlite3\_value \*sqlite3\_column\_value(sqlite3\_stmt\*, int iCol);

这些函数都是从结果中读取每一列的数据。

参数1：SQL命令句柄。

参数2：列索引。结果列的最左边为序号0，以此类推。列数由sqlite3\_column\_count()返回得到。

若SQL语句执行结果无效，或列索引超出范围，则本函数结果无定义。

函数sqlite3\_column\_type()返回结果列初始数据的数据类型代码，数据类型代码有：SQLITE\_INTEGER、SQLITE\_FLOAT、SQLITE\_TEXT、SQLITE\_BLOB和SQLITE\_NULL。若发生类型转换，sqlite3\_column\_type()的结果无效。

函数sqlite3\_column\_bytes()用于返回结果字节数，若结果为BLOB或UTF-8字符串类型，sqlite3\_column\_bytes()返回BLOB或UTF-8字符串字节数。

若结果为UTF-16字符串类型，sqlite3\_column\_bytes()将字符串转换为UTF-8，然后返回字节数。

若结果为数值类型，sqlite3\_column\_bytes()使用sqlite3\_snprintf()将它转换为UTF-8字符串类型，然后返回字节数。

若结果为空，sqlite3\_column\_bytes()返回0。

若结果为BLOB或UTF-16字符串了性，sqlite3\_column\_bytes16()返回字节数。

若结果为UTF-8字符串类型，sqlite3\_column\_bytes16()将它转换为UTF-16然后返回字节数。

若结果为数值类型，sqlite3\_column\_bytes16()使用sqlite3\_snprintf()将它转换为UTF-16字符串类型，然后返回字节数。

若结果为空，sqlite3\_column\_bytes16()返回0。

以上返回长度不包括字符串0终止符。

\*由sqlite3\_column\_text()和sqlite3\_column\_text16()返回的字符串，即使是空字符串，也都是以0终止符结束的。sqlite3\_column\_blob()读取一个zero-length BLOB返回得到的是空指针。

记住：

sqlite3\_column\_text() followed by sqlite3\_column\_bytes()

sqlite3\_column\_blob() followed by sqlite3\_column\_bytes()

sqlite3\_column\_text16() followed by sqlite3\_column\_bytes16()

**sqlite3\_finalize()**

为防止内存泄露，调用该接口释放sqlite3\_prepare()产生的对象。

int sqlite3\_finalize(sqlite3\_stmt \*pStmt);

成功返回SQLITE\_OK

**sqlite3\_close()**

用于关闭数据库连接，所有的对象释放sqlite3\_finalize()都要先于该操作。

int sqlite3\_close(sqlite3 \*);

成功返回SQLITE\_OK

若未关闭prepared statements或BLOB handles，返回SQLITE\_BUSY

#### 分步查询接口使用方法

（1）一般调用sqlite3\_open()连接一个数据库，一个连接同时调用多个数据库可以使用attach功能。

（2）使用sqlite3\_prepare()创建prepared statement对象。

（3）调用sqlite3\_step()执行prepared statement一次或多次。

（4）查询时，在两个sqlite3\_step()调用之间调用sqlite3\_column()解压结果。

（5）使用sqlite3\_finalize()销毁prepared statement对象。

（6）调用sqlite3\_close()关闭数据库连接。



图表 31 接口调用过程

#### 封装查询

为方便使用，sqlite3\_exec或sqlite3\_get\_table封装了prepared statement。

（1）sqlite3\_exec()接口集合了接口使用的步骤（2）~（5）。callback函数用于传入sqlite3\_exec()去处理结果集的每一行。

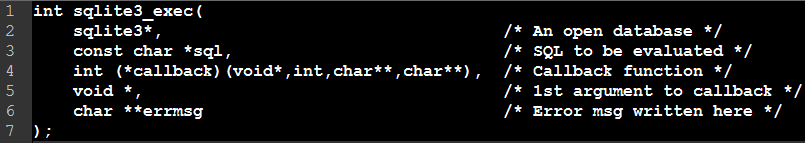
（2）sqlite3\_get\_table()是另一个集合了步骤（2）~（5）的封装接口，唯一的不同是它将结果存储在堆内存中而不是交给callback函数处理。



图表 32 封装接口调用过程

#### \*Sqlite3\_exec封装接口

sqlite3\_exec：One-Step Query Execution Interface



图表 33 Sqlite3\_exec接口参数

sqlite3\_exec 封装了sqlite3\_prepare\_v2()、sqlite3\_step()和sqlite3\_finalize()，只需少量代码就可读入多个SQL语句。

sqlite3\_exec接口参数简介：

（1）**sqlite3\***

数据库指针，指向一个开放的数据库。

（2）**const char \*sql**

SQL语句指针，指向零句或多句通过“;”隔开的SQL语句字符串。若sqlite3\_exec()的第二个参数为空指针，或指向一个空字符串或指向一个只有空格以及/或标点符，而没有任何可以被执行的SQL命令，那么数据库将不会有变动。

（3）**int (\*callback)(void\*,int,char\*\*,char\*\*)**

callback函数指针，若非空，它将调用SQL语句执行的结果的行信息，则将结果信息传入callback函数，callback函数让用户自己定义，一般是打印功能的函数。若为空，结果行信息将不被调用，直接忽略。

int：callback第2个参数是结果的列数。

char\*\*：callback第3个参数是一个指向字符串的指针数组，该字符串犹如从sqlite3\_column\_text()获取得到，每一列对应一个字符串。若结果行为空，对应的字符串指针为空指针。

char\*\*：callback第4个参数是一个指向字符串的指针数组，字符串从sqlite3\_column\_name()获取，指向结果列的列名。

如果callback返回non-zero，sqlite3\_exec()返回SQLITE\_ABORT。

（4）**void \***

传送到callback函数的第一个参数。

（5）**char \*\*errmsg**

若SQL语句执行错误，若该参数非空，所有错误信息将被写入由sqlite3\_malloc()获取的内存空间，并返回到该参数。当错误信息不再需要时，为防止内存泄露，应用程序应该调用sqlite3\_free()释放该参数的空间。若该参数非空，且没有错误发生，sqlite3\_exec()在返回前将该指针参数置空。

#### 绑定查询参数

SQLite允许同样的prepared statement被执行多次，这需要使用下面两个程序接口：

sqlite3\_reset()

sqlite3\_bind()

在prepared statement被sqlite3\_step()执行一次或多次之后，为了可以被执行多一轮，可以调用sqlite3\_reset()将它重置，在存在的prepared statement中使用sqlite3\_reset()而不是重新创建新的prepared statement可以避免不必要的对sqlite3\_prepare()的调用。在许多SQL命令中，执行sqlite3\_prepare()的时间等于甚至超过执行sqlite3\_step()所需的时间，所以重置可以帮助提升性能。

通常我们并不会多次执行相同的SQL命令，但是执行相似的命令却很常见。比如插入命令，我们需要多次插入，每一次的值不同，为适应这种应用，SQLite允许SQL statements包含参数，参数在执行前被绑定。这些参数可以被更改，同样的prepared statement带不同的参数可以被继续执行而不必新建prepared statement。

在SQLite中，无论包含一个字符串文字是否有效，我们都可以使用下列形式的参数：

?

?NNN

:AAA

$AAA

@AAA

其中，NNN代表一个整型值，AAA代表一个标示符。

一个参数初始时有个空值。在第一次调用sqlite3\_step()之前或调用sqlite3\_reset()即可之后，应用程序可以调用sqlite3\_bind()接口去捆绑参数和值。每次调用sqlite3\_bind()都覆盖相同参数的先前绑定。

一个应用程序允许提前准备多个SQL命令，没有任何数量的限制。

下面是对sqlite3\_reset()和sqlite3\_bind()的接口介绍。

int sqlite3\_bind\_blob(sqlite3\_stmt\*, int, const void\*, int n, void(\*)(void\*));

int sqlite3\_bind\_double(sqlite3\_stmt\*, int, double);

int sqlite3\_bind\_int(sqlite3\_stmt\*, int, int);

int sqlite3\_bind\_int64(sqlite3\_stmt\*, int, sqlite3\_int64);

int sqlite3\_bind\_null(sqlite3\_stmt\*, int);

int sqlite3\_bind\_text(sqlite3\_stmt\*, int, const char\*, int n, void(\*)(void\*));

int sqlite3\_bind\_text16(sqlite3\_stmt\*, int, const void\*, int, void(\*)(void\*));

int sqlite3\_bind\_value(sqlite3\_stmt\*, int, const sqlite3\_value\*);

int sqlite3\_bind\_zeroblob(sqlite3\_stmt\*, int, int n);

以上是绑定接口函数，针对不同类型进行参数匹配。

int sqlite3\_reset(sqlite3\_stmt \*pStmt);

#### 扩展的SQLite

SQLite包含用于扩展功能的程序接口：

sqlite3\_create\_collation()

sqlite3\_create\_function()

sqlite3\_create\_module()

The sqlite3\_create\_collation()接口用于为排序文本创建新的排序序列。

The sqlite3\_create\_module()接口用于注册新的虚拟表。

The sqlite3\_create\_function()接口创建新的SQL函数，无论是scalar还是aggregate。新函数典型地使用下面的附加接口：

sqlite3\_aggregate\_context()

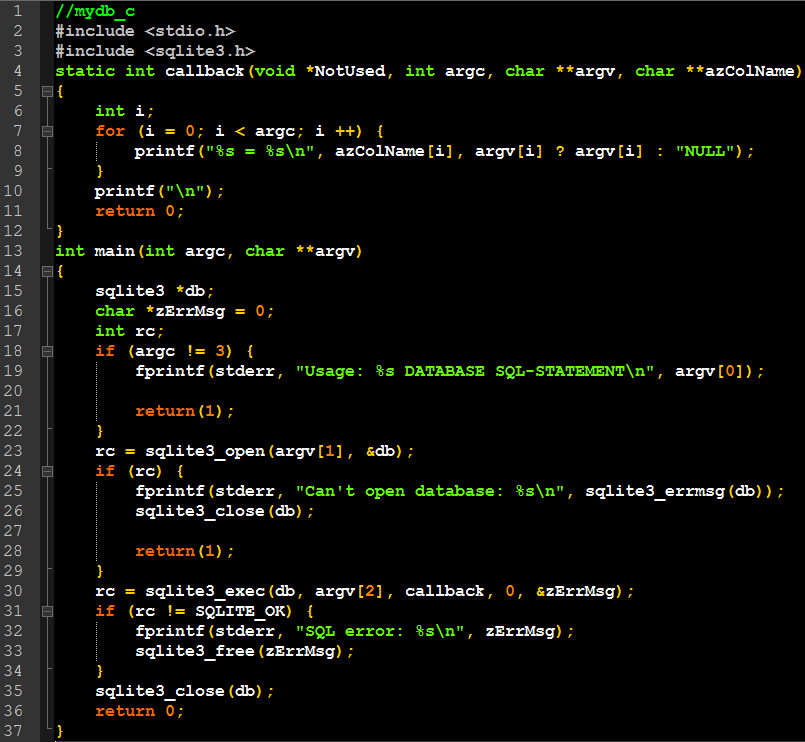
sqlite3\_result()

sqlite3\_user\_data()

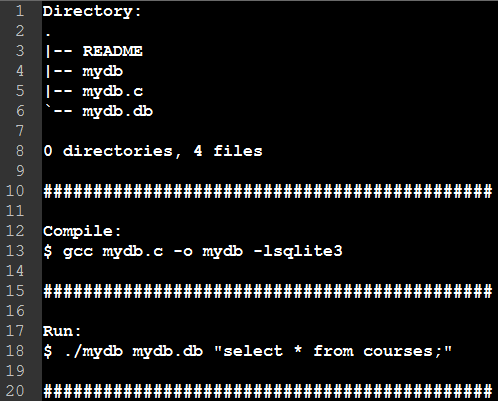
sqlite3\_value()

#### 接口使用例子

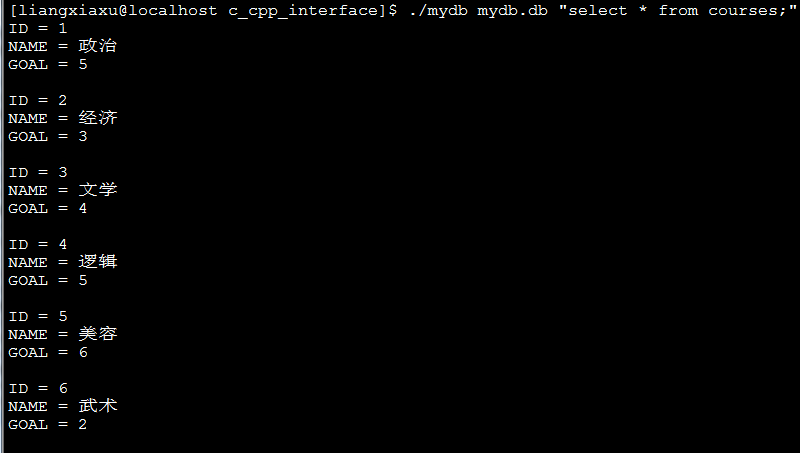
##### 封装查询



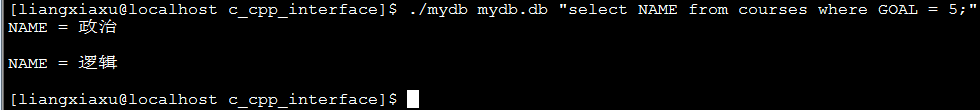
图表 34 封装查询例子



图表 35 编译与使用示例

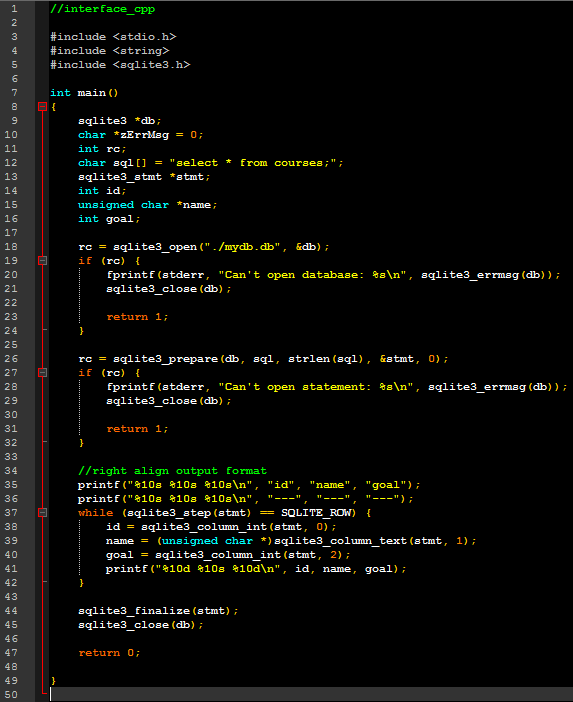


图表 36 C/C++接口效果1

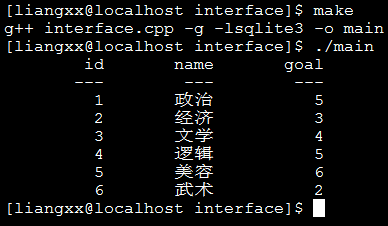


图表 37 C/C++接口效果2

##### 分步查询

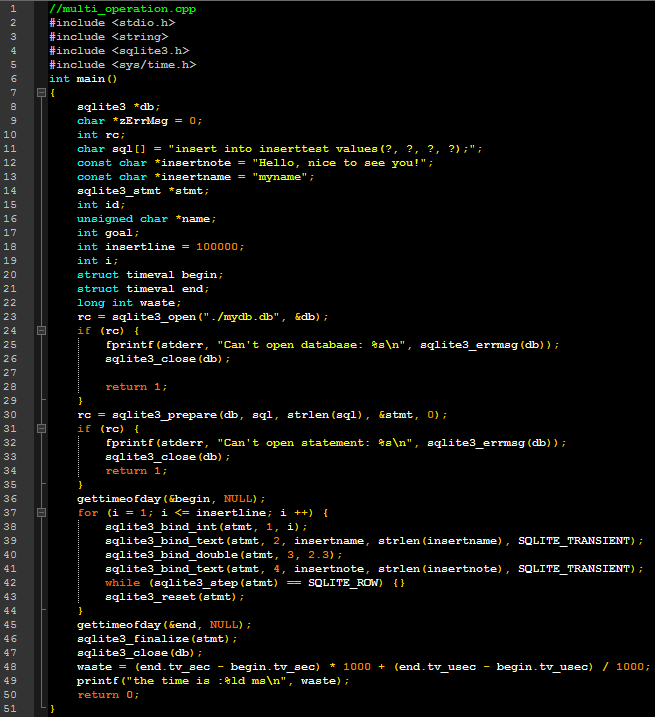


图表 38 分步查询



图表 39 效果

##### 绑定查询



图表 40 绑定查询

# 第三章 SQLite性能分析与应用建议

## SQLite文件模式与内存模式性能测试

环境

172.18.8.36

SQLite V3.7.13

pragma synchronous = 2;//未经优化

### 插入操作时间性能测试（绑定查询）

#### 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

#### 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(?, ?, ?, ?);

#### 测试结果

##### 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

##### 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

#### 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。**

### 查询操作时间性能测试（绑定查询）

#### 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

#### 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 = ?;

#### 测试结果

##### 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

##### 内存模式（单位us）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6655 | 6490 | 6294 | 6401 | 6341 |
| 6330 | 6284 | 6327 | 6040 | 6177 |
| 6564 | 6461 | 6380 | 6435 | 6351 |
| 6376 | 6172 | 6407 | 6386 | 6462 |

平均值：6366.65us

#### 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

**内存模式的查询速度是文件模式的6.42倍。**

### 删除操作时间性能测试（绑定查询）

#### 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20次

（2）内存驻留模式：测试20次

#### 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 = ?;

#### 测试结果

##### 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

##### 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

#### 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。**

### 递归操作时间性能测试（绑定查询）

#### 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

#### 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 = ?;

#### 测试结果

##### 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

##### 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

#### 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。**

## SQLite文件模式性能曲线（稳定性测试）

### 测试表格

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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! |

### 测试方法

以文件模式分别测试插入、查询、删除与递归在记录量为10万、20万……100万的时间性能，得到大致的时间性能曲线。

### 数据记录

#### 插入

（1）//36

Inserted 100000 : 700 s

Inserted 200000 : 1025 s

Inserted 300000 : 1344 s

Inserted 400000 : 1658 s

Inserted 500000 : 1970 s

Inserted 600000 : 2286 s

Inserted 700000 : 2603 s

Inserted 800000 : 2918 s

Inserted 900000 : 3228 s

Inserted 1000000 : 3538 s

（2）//36

Inserted 100000 : 322 s

Inserted 200000 : 650 s

Inserted 300000 : 1002 s

Inserted 400000 : 1340 s

Inserted 500000 : 1683 s

Inserted 600000 : 2036 s

Inserted 700000 : 2431 s

Inserted 800000 : 2757 s

Inserted 900000 : 3174 s

Inserted 1000000 : 3586 s

（3）//36

Inserted 100000 : 472 s

Inserted 200000 : 890 s

Inserted 300000 : 1407 s

Inserted 400000 : 1887 s

Inserted 500000 : 2360 s

Inserted 600000 : 2685 s

Inserted 700000 : 3009 s

Inserted 800000 : 3330 s

Inserted 900000 : 3652 s

Inserted 1000000 : 3973 s

#### 删除

（1）//36

Deleted 100000 :418810 ms

Deleted 200000 :847425 ms

Deleted 300000 :1272729 ms

Deleted 400000 :1693850 ms

Deleted 500000 :2126292 ms

Deleted 600000 :2557329 ms

Deleted 700000 :3005906 ms

Deleted 800000 :3449551 ms

Deleted 900000 :3917235 ms

Deleted 1000000 :4573497 ms

（2）//36

Deleted 100000 :414154 ms

Deleted 200000 :813430 ms

Deleted 300000 :1214220 ms

Deleted 400000 :1621969 ms

Deleted 500000 :2032979 ms

Deleted 600000 :2437076 ms

Deleted 700000 :2768456 ms

Deleted 800000 :3100208 ms

Deleted 900000 :3430651 ms

Deleted 1000000 :3760807 ms

（3）//36

Deleted 100000 :370830 ms

Deleted 200000 :710955 ms

Deleted 300000 :1055890 ms

Deleted 400000 :1398603 ms

Deleted 500000 :1728580 ms

Deleted 600000 :2064468 ms

Deleted 700000 :2397483 ms

Deleted 800000 :2734016 ms

Deleted 900000 :3070721 ms

Deleted 1000000 :3421397 ms

#### 递归

（1）

1855304 us|3351625 us|4746351 us|6542699 us|7969278 us|

9447883 us|10862539 us|12691431 us|14430473 us|16444249 us

（2）

1154951 us|2528247 us|4028121 us|5555688 us|6978395 us|

8440328 us|9831851 us|11353220 us|12877061 us|14092965 us

（3）

Itered 100000 :1649104 us

Itered 200000 :3209989 us

Itered 300000 :4732152 us

Itered 400000 :6288031 us

Itered 500000 :7837180 us

Itered 600000 :9362591 us

Itered 700000 :10874118 us

Itered 800000 :12397127 us

Itered 900000 :13876492 us

Itered 1000000 :15441910 us

（4）//36

Itered 100000 :1615425 us

Itered 200000 :3007158 us

Itered 300000 :4409856 us

Itered 400000 :5697908 us

Itered 500000 :7092365 us

Itered 600000 :8322465 us

Itered 700000 :9608453 us

Itered 800000 :10892382 us

Itered 900000 :12155803 us

Itered 1000000 :13503717 us

（5）//36

Itered 100000 :2175002 us

Itered 200000 :3983139 us

Itered 300000 :5601850 us

Itered 400000 :7176726 us

Itered 500000 :8894231 us

Itered 600000 :10830991 us

Itered 700000 :13088850 us

Itered 800000 :15462005 us

Itered 900000 :17614787 us

Itered 1000000 :19689549 us

（6）//36

Itered 100000 :2023259 us

Itered 200000 :4056199 us

Itered 300000 :6162354 us

Itered 400000 :8114893 us

Itered 500000 :10172845 us

Itered 600000 :12185307 us

Itered 700000 :14365771 us

Itered 800000 :16308041 us

Itered 900000 :18221800 us

Itered 1000000 :20223517 us

（7）

Itered 100000 :2732668 us

Itered 200000 :5496716 us

Itered 300000 :8186975 us

Itered 400000 :11340429 us

Itered 500000 :13654048 us

Itered 600000 :15875435 us

Itered 700000 :17858533 us

Itered 800000 :20024660 us

Itered 900000 :21879395 us

Itered 1000000 :24106384 us

### 结论

#### 插入



图表 41文件插入测试

#### 删除



图表 42文件删除测试

#### 递归



## SQLite内存模式性能曲线（稳定性测试）

### 测试表格

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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! |
| **……** | …… | …… | …… | …… | …… |
| **10000000** | 10000000 | 0 | Myname | 2.3 | Hello, nice to see you! |

### 测试方法

以内存模式分别测试插入、删除与递归在记录量为100万、200万……1000万的时间性能各三次，得到大致的时间性能稳定性曲线。

### 数据记录

#### 插入

1）//36

Inserted 1000000 : 7976431 us

Inserted 2000000 : 14892516 us

Inserted 3000000 : 21950989 us

Inserted 4000000 : 30006010 us

Inserted 5000000 : 38148008 us

Inserted 6000000 : 45215063 us

Inserted 7000000 : 52271200 us

Inserted 8000000 : 60465057 us

Inserted 9000000 : 68207345 us

Inserted 10000000 : 75586576 us

2）//36

Inserted 1000000 : 8234258 us

Inserted 2000000 : 17246392 us

Inserted 3000000 : 23846802 us

Inserted 4000000 : 33478773 us

Inserted 5000000 : 42746689 us

Inserted 6000000 : 51330580 us

Inserted 7000000 : 60381793 us

Inserted 8000000 : 67518484 us

Inserted 9000000 : 74653747 us

Inserted 10000000 : 82009016 us

3）//36

Inserted 1000000 : 8179029 us

Inserted 2000000 : 15272920 us

Inserted 3000000 : 23749032 us

Inserted 4000000 : 32925969 us

Inserted 5000000 : 39785839 us

Inserted 6000000 : 47353577 us

Inserted 7000000 : 54503782 us

Inserted 8000000 : 62411587 us

Inserted 9000000 : 69884166 us

Inserted 10000000 : 76843287 us

#### 删除

1）

Deleted 1000000: 8798165 us

Deleted 2000000: 16945737 us

Deleted 3000000: 24660682 us

Deleted 4000000: 32897718 us

Deleted 5000000: 41569590 us

Deleted 6000000: 50103427 us

Deleted 7000000: 58166290 us

Deleted 8000000: 65728632 us

Deleted 9000000: 73315073 us

Deleted 10000000: 81670755 us

2）

Deleted 1000000: 8032858 us

Deleted 2000000: 16439323 us

Deleted 3000000: 24152502 us

Deleted 4000000: 32582603 us

Deleted 5000000: 40492229 us

Deleted 6000000: 48556505 us

Deleted 7000000: 56383190 us

Deleted 8000000: 64001876 us

Deleted 9000000: 72752782 us

Deleted 10000000: 81769339 us

3）

Deleted 1000000: 9116650 us

Deleted 2000000: 16681822 us

Deleted 3000000: 26716232 us

Deleted 4000000: 34700394 us

Deleted 5000000: 43002109 us

Deleted 6000000: 52047968 us

Deleted 7000000: 60693536 us

Deleted 8000000: 69412588 us

Deleted 9000000: 76816128 us

Deleted 10000000: 84416313 us

#### 递归

1）

Itered 1000000 : 5268770 us

Itered 2000000 : 10139808 us

Itered 3000000 : 14391026 us

Itered 4000000 : 18750444 us

Itered 5000000 : 23847901 us

Itered 6000000 : 28419808 us

Itered 7000000 : 32682493 us

Itered 8000000 : 36840145 us

Itered 9000000 : 41233320 us

Itered 10000000 : 45567138 us

2）

Itered 1000000 : 4866050 us

Itered 2000000 : 9449031 us

Itered 3000000 : 14397720 us

Itered 4000000 : 18926265 us

Itered 5000000 : 24445841 us

Itered 6000000 : 28986159 us

Itered 7000000 : 34491299 us

Itered 8000000 : 38880243 us

Itered 9000000 : 43486521 us

Itered 10000000 : 48231763 us

3）

Itered 1000000 : 4532485 us

Itered 2000000 : 10021045 us

Itered 3000000 : 14929680 us

Itered 4000000 : 20247541 us

Itered 5000000 : 25046072 us

Itered 6000000 : 30146118 us

Itered 7000000 : 35776499 us

Itered 8000000 : 40383882 us

Itered 9000000 : 45231450 us

Itered 10000000 : 50419806 us

### 结论

#### 插入



图表 43内存插入测试

#### 删除



图表 44内存删除测试

#### 递归



图表 45内存递归测试

## 关于SQLite的应用建议

### SQLite的优势

（1）单一的磁盘文件，非常适合系统间的移植操作。

（2）C接口功能丰富，操作简单，并且库文件非常小，对C函数库的依赖少，独立性好。

（3）针对小数据量的操作性能稳定。

### SQLite的劣势

（1）不适合大数据量操作

对于磁盘文件操作，百万级别数据量的操作花费为几千秒，根据硬件环境的繁忙程度性能波动大。

对于内存驻留操作，千万级以上数据量的波动较大。

（2）

### 应用建议

（1）针对实时性要求低、小数据量数据操作，采用磁盘文件方式，可以方便数据的备份移植。

（2）内存驻留模式的插入删除性能是文件模式的几百倍，查询递归性能为文件模式的几倍，针对百万级以上数据量，采用内存驻留方式操作、磁盘方式备份相结合方式，可以获得较好的效果。

# 图表索引

[图表 1 SQLite CLP 15](#_Toc339467382)

[图表 2 SQLite CLP 17](#_Toc339467383)

[图表 3 Create-table-stmt 17](#_Toc339467384)

[图表 4 Expression affinity 18](#_Toc339467385)

[图表 5 Column-def 18](#_Toc339467386)

[图表 6 Type-name 19](#_Toc339467387)

[图表 7 Column-constraint 19](#_Toc339467388)

[图表 8 Foreign-key-clause 20](#_Toc339467389)

[图表 9 courses（空） 21](#_Toc339467390)

[图表 10 Students（空） 21](#_Toc339467391)

[图表 11 Insert-stmt 22](#_Toc339467392)

[图表 12 courses 22](#_Toc339467393)

[图表 13 查询courses 23](#_Toc339467394)

[图表 14 students 23](#_Toc339467395)

[图表 15 查询students 24](#_Toc339467396)

[图表 16 Select-core 25](#_Toc339467397)

[图表 17 查询 25](#_Toc339467398)

[图表 18 Delete-stmt 26](#_Toc339467399)

[图表 19 Delete一行 26](#_Toc339467400)

[图表 20 Drop-table-stmt 26](#_Toc339467401)

[图表 21 Drop 27](#_Toc339467402)

[图表 22单条SQL语句自成事务 29](#_Toc339467403)

[图表 23Begin-rollback-commit 30](#_Toc339467404)

[图表 24冲突解决CLP命令 31](#_Toc339467405)

[图表 25冲突解决CLP命令的性质排序 31](#_Toc339467406)

[图表 26五种状态 32](#_Toc339467407)

[图表 27五种状态的切换状态机 33](#_Toc339467408)

[图表 28死锁示例 33](#_Toc339467409)

[图表 29预防死锁的三种事务类型 34](#_Toc339467410)

[图表 30 接口匹配 36](#_Toc339467411)

[图表 31 接口调用过程 41](#_Toc339467412)

[图表 32 封装接口调用过程 42](#_Toc339467413)

[图表 33 Sqlite3\_exec接口参数 42](#_Toc339467414)

[图表 34 封装查询例子 46](#_Toc339467415)

[图表 35 编译与使用示例 47](#_Toc339467416)

[图表 36 C/C++接口效果1 47](#_Toc339467417)

[图表 37 C/C++接口效果2 48](#_Toc339467418)

[图表 38 分步查询 49](#_Toc339467419)

[图表 39 效果 49](#_Toc339467420)

[图表 40 绑定查询 50](#_Toc339467421)

[图表 41文件插入测试 66](#_Toc339467422)

[图表 42文件删除测试 67](#_Toc339467423)

[图表 43内存插入测试 74](#_Toc339467424)

[图表 44内存删除测试 75](#_Toc339467425)

[图表 45内存递归测试 76](#_Toc339467426)