

第10章 SQLite数据库



目 录

10.1 sqlite数据库概述

10.2 sqlite安装

10.3 sqlite的常用命令

10.4 sqlite的数据类型

10.5 sqlite的API函数

嵌入式数据库系统是指支持移动计算或某种特定计算模式的数据库管理系统，它通常与操作系统和具体应用集成在一起，运行在智能型嵌入式设备或移动设备上。由于嵌入式数据库系统总是与移动计算相结合，所以通常情况下嵌入式数据库也被称为嵌入式移动数据库。

- ▶ 在嵌入式数据库领域，各大数据库厂商竞争也日趋激烈，Oracle、IBM、Sybase、InterSystems、日立、Firebird等均在这一领域有所行动。如Oracle并购全球最大的内存数据库厂商TimesTen之后，又收购了全球下载用户最多的嵌入式数据库厂商Sleepycat及其Berkeley DB产品，进一步完善了嵌入式软件的产品线。从Oracle自身来说，Oracle提供的不仅是一个嵌入式数据库产品，更重要的是从底层提供的一种端到端的数据管理架构，并大力支持重点行业领域的关键合作伙伴在此架构上开发的相关应用和服务。

嵌入式数据库的特点：

- ▶ 嵌入性：是嵌入式数据库的基本特性。嵌入式数据库不仅可以嵌入到其他的软件当中，也可以嵌入到硬件设备当中。
- ▶ 例如嵌入式数据库Empress就是使数据库以组件的形式存在，并发布给客户，客户只需要像调用自己定义的函数那样调用相应的函数就可以创建表、插入删除数据等常规的数据库操作。客户在自己的产品发布时，可以将Empress数据库编译到自己的产品内，变成自己产品的一部分，最终用户是感受不到数据库的存在的，也不用特意去维护数据库。

- ▶ 移植性：由于嵌入式系统的应用领域非常广泛，所采用的嵌入式操作系统和软硬件环境也各不相同，为了能适应各种差异性，嵌入式数据库必须具有一定的可移植性。
- ▶ 安全性：许多应用领域中的嵌入式设备是系统中数据管理和处理的关键设备，因此嵌入式设备上的数据库系统对存取权限的控制比较严格。同时，某些数据的个人隐私性很高。

- ▶ 实时性：实时性和嵌入性是分不开的。只有具有了嵌入性的数据库才能够第一时间得到系统的资源，对系统的请求在第一时间做出响应。但是，并不是具有嵌入性就一定具有实时性。要想嵌入式数据库具有很好的实时性，必须做很多额外的工作。
- ▶ 比如：Empress实时数据库将嵌入性和高速的数据引擎、定时功能以及防断片处理等措施整合在一起来保证最基本的实时性。当然，不同的场合实时性要求比较高时，除了软件的实时性外，硬件的实时性也是必须的，具体情况需要有具体和切实的解决方案，不能一概而论。

- ▶ **可靠性：**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。

- ▶ 主动性：传统的数据库系统是被动的，因为仅当用户或者应用程序控制使用它时，它才会现存信息做出相应的处理和响应。而主动数据库系统能够主动监视当前信息，推断当前未来状态的出现。

嵌入式数据库的分类方法很多，根据其嵌入的对象不同分为：

- ▶ 面向软件嵌入数据库。它将数据库作为组件嵌入到其他的软件系统中。一般用在对数据库的安全性、稳定性和速度要求比较高的系统中。这种结构资源消耗低，最终用户不用维护数据库，甚至感受不到数据的存在。
- ▶ 面向设备嵌入数据库。它将关系型数据库嵌入到设备当中去，作为设备数据处理的核心组件。这种场合要求数据库有很高的实时性和稳定性，一般运行在实时性非常高的操作系统当中。为了达到这些要求有的厂商采用关系型的数据结构，有的采用非关系型的数据结构。有时候甚至直接和硬件打交道。当然，这种结构在实时性要求不高的移动场合更能够胜任。
- ▶ 内存数据库。数据库直接在内存内运行，数据处理更加高速，不过安全性等方面需要额外的手段来保障。

也可以根据其应用的不同分为：
普通嵌入式数据库、
嵌入式移动数据库、
小型架构数据库等。



嵌入式数据库的应用

1. 医疗领域

北美和欧洲的一些著名的厂商利用嵌入式数据库开发过完整的电子病历系统，同时将数据库嵌入到医疗器械当中。如，血液分析装置、乳癌的检测装置、医学图像装置等。这样医疗系统的各个环节可以无缝地和各种医疗设备进行数据交流，并轻松地处理这些设备送过来的数据信息，在必要的时候共享给有权限查看的用户。

2. 军事设备和系统

一些著名的军事机构和全球著名的武器生产商将嵌入式数据库运用到他们的系统控制装置、战士武器、军舰装置、火箭和导弹装置中。这些场合用的数据库有很多的安全设定和特化设定，基本上严格按照每个客户的技术标准的要求来特化引擎级构件。具体的应用级的构件由客户自己完成。

3.地理信息系统

地理信息包括的范围很广，在国外地理信息系统已经发展了很多年，国内这几年也逐渐加大对地理信息系统方面的投入。嵌入式数据库在地理信息系统方面的应用非常广泛。如，空间数据分析系统、卫星天气数据、龙卷风和飓风监控及预测、大气研究监测装置、天气数据监测、相关卫星气象和海洋数据的采集装置、导航系统等等。几乎涉及到地理信息的方方面面。

4.工业控制

工业控制的一个基本方式是一个反馈的闭环或半闭环的控制方式。随着工业控制技术的发展，简单的数据采集方式和反馈方式基本上很难满足要求。采用嵌入式数据库即能够进行高速的数据采集，也能够快速的反馈。正因为如此，在一些核电站监控装置、化学工厂系统监控装置、电话制造系统监控装置、汽车引擎监控装置及工业级机器人中有广泛应用。

5.网络通讯

随着互联网的发展，网络越来越普及，网络设备的处理能力越来越强、各种要求也越来越高，运用嵌入式数据库也成了必然趋势。我们现在日常见到的很多网络设备和系统都已经使用了嵌入式数据库。嵌入式数据库在一些企业内部互联网装置、网络传输的分布式管理装置、语音邮件追踪系统、VoIP交换机、路由器、基站控制器等系统中都有应用。

6.空间探索

一些全球著名的机构将嵌入式数据库用在一些著名的空间探索装置中，如大家熟知的一些太阳系内行星的探测器等。

7.消费类电子

目前在中国消费类电子比较火热，它包含的范围也非常广。如：个人消费相关的PND、移动电话、PDA、SmartPhone、数码产品等；信息家电和智能办公相关的机顶盒、家用多媒体盒、互联网电视接收装置、打印机、一体机等；还有汽车电子等。在欧美和日本不仅在这些方面已经有不少的成功应用和技术积累，还正在和亚太的一些著名厂商积极展开新的合作和研发，目前已经取得实质性的成果。

在众多的数据库中，如何选择适用于嵌入式系统的数据库呢？嵌入式系统开发环境决定了其对数据库需求的要素：

- (1) 体积较小
- (2) 功能齐备
- (3) 代码开源
- (4) 性能可靠

10.1 Part One

sqlite数据库概述

10.1.1 基于Linux平台的嵌入式数据库概述

基于Linux平台的数据库非常多，大型的商用数据库有Oracle、Sybase、Informix、Informix、IBM DB2等；

中小型的更是不胜枚举，最常见的几种是：PostgreSQL、MySQL、mSQL、Berkeley DB和SQLite数据库。

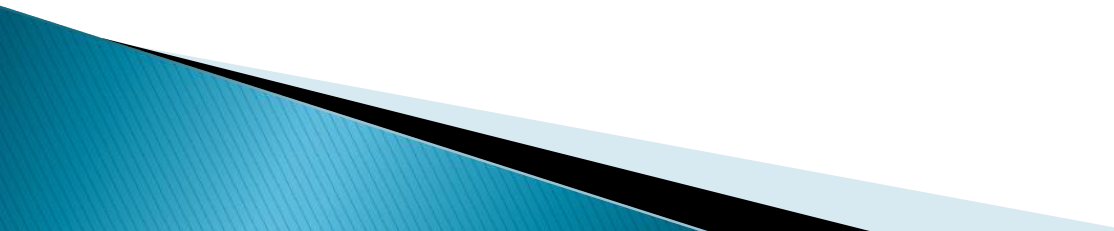
MySQL是多用户、多进程的**SQL**数据库系统。**MySQL**既能够作为一个单独的应用程序应用在客户端服务器网络环境中，也能够作为一个库而嵌入到其他的软件中。使用 **C**和 **C++**编写，并使用了多种编译器进行测试，保证了源代码的可移植性。**MySQL**为多种编程语言提供了**API**。支持多线程，充分利用 **CPU** 资源。**MySQL** 软件采用了双授权政策，它分为社区版和商业版，由于其体积小、速度快、总体拥有成本低，尤其是开放源码这一特点，一般中小型网站的开发常选择 **MySQL** 作为网站数据库。

开源数据库性能对比表

产品名称	速度	稳定性	数据库容量	SQL支持	Win32 平台下最小体积	数据操纵
SQLite	最快	好	2TB	大部分SQL-92	374KB	SQL
Berkeley DB	快	好	256TB	不支持	840KB	仅应用程序接口
Firebird 嵌入式服务器版	快	好	64TB	完全SQL-92与大部分SQL-99	3.68M	SQL
SQL CE	慢	好	4GB	完全SQL-92	3MB	SQL

10.1.2 SQLite的特点

SQLite是一个开源的、内嵌式的关系型数据库。它是**D. Richard Hipp**采用**C**语言开发出来的完全独立的，不具有外部依赖性的嵌入式数据库引擎。**SQLite**第一个版本发布于**2000年5月**，在便携性、易用性、紧凑性、有效性和可靠性方面有突出的表现。



Sqlite主要特点有：

- 支持**ACID**事务。
- 零配置，即无需安装和管理配置。
- 储存在单一磁盘文件中的一个完整的数据库。
- 数据文件可在不同字节顺序的机器间自由共享。
- 支持数据库大小至**2TB**。
- 程序体积小，全部**C**语言代码约**3**万行（核心软件，包括库和工具），**250KB**大小。
- 相对于目前其他嵌入式数据库具有更快捷的数据操作。
- 支持事务功能和并发处理。
- 程序完全独立，不具有外部依赖性。
- 支持多种硬件平台，如**arm/ Linux**、**SPARC/Solaris**等。

10.1.3 sqlite的体系结构

SQLite组件

SQLite 由以下几个组件组成：SQL 编译器、内核、后端以及附件，SQLite 通过利用虚拟机和虚拟数据库引擎（VDBE），使调试、修改和扩展 SQLite 的内核变得更加方便。所有 SQL 语句都被编译成易读的、可以在 SQLite 虚拟机中执行的程序集。SQLite 通过将SQL语句翻译为**字节码**之后在**虚拟机**上运行该字节码的方式来工作

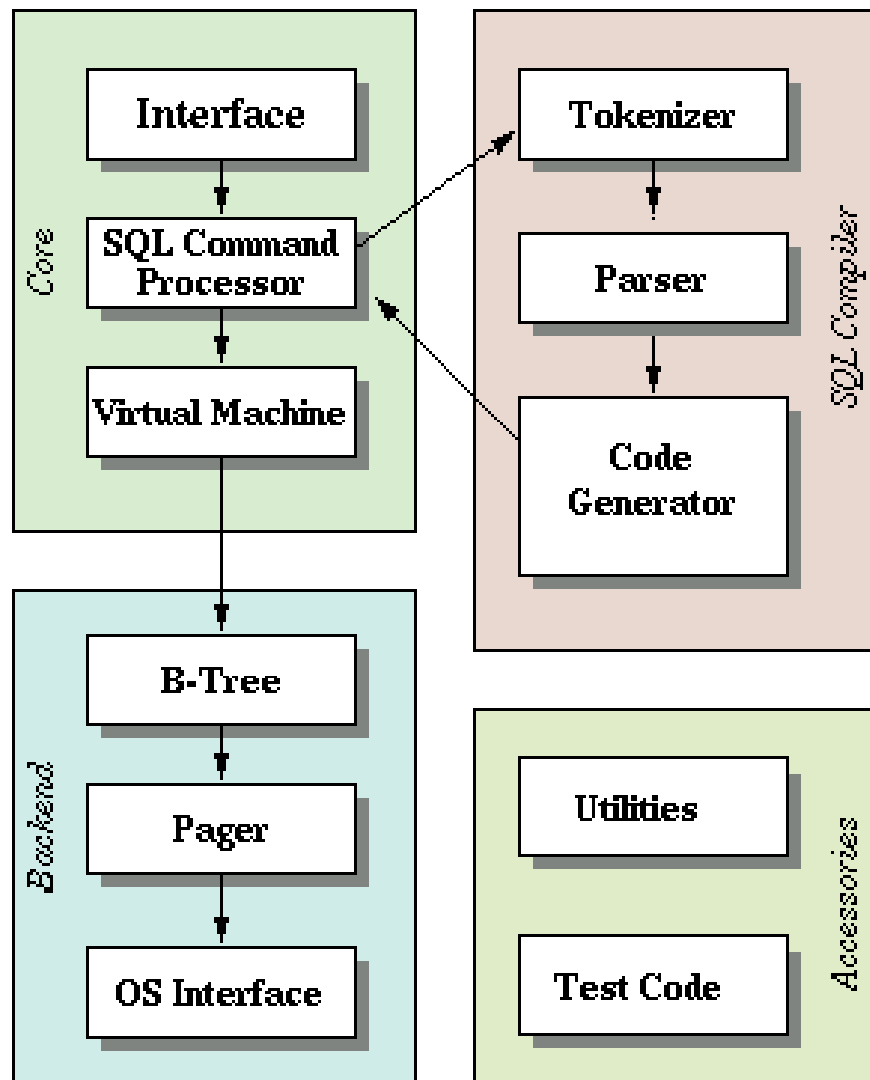


图10-1 sqlite的体系结构

10.2

Part Two

sqlite安装

SQLite网站(www.sqlite.org)同时提供**SQLite**的已编译版本和源程序。编译版本可同时适用于**Windows**和**Linux**。

有几种形式的二进制包供选择，以适应**SQLite**的不同使用方式。包括：静态链接的命令行程程序(**CLP**)、**SQLite**动态链接库(**DLL**)和**Tcl**扩展。

SQLite源代码以两种形式提供，以适应不同的平台：

一种为了在**Windows**下编译，
另一种为了在**POSIX**平台(如**Linux**、**BSD**、**Solaris**)下编译

安装SQLite

- (1) 复制sqlite-autoconf-3080701.tar.gz到/opt目录下。
- (2) 解压sqlite-autoconf-3080701.tar.gz。
`#tar xvzf sqlite-autoconf-3080701.tar.gz`
- (3) 新建一个安装目录/opt/sqlite_x86。
`#mkdir /opt/sqlite_x86`
- (4) 进入解压目录/opt/sqlite-autoconf-3080701配置SQLite，执行configure命令生成Makefile文件。
`# cd /opt/sqlite-autoconf-3080701`
`# ./configure --prefix=/opt/sqlite_x86(或/usr/local/sqlite)`
- (5) 执行make安装SQLite。
`# make`
- (6) 执行make install将SQLite安装在/opt/sqlite_x86路径下
`# make install`
- (7) 安装完成后，进入/opt/sqlite_x86目录查看安装文件
`# cd /opt/sqlite_x86`
`# ls` (有: bin include lib share 目录)
`# cd bin`
`# ls` (有访问数据库的工具: sqlite3)
- (8) 安装完成后，可删除安装目录下的临时文件
`#rm -rf /opt/sqlite-autoconf-3080701`

10.3 Part Three

sqlite的常用命令

启动**sqlite3**程序，仅仅需要敲入带有**SQLite**数据库名字的"**sqlite3**"命令即可。如果文件不存在，则创建一个新的数据库文件。然后 **sqlite3**程序将提示输入**SQL**，输入**SQL**语句（以分号“**;**”结束）并回车之后，**SQL**语句就会执行。

例如，创建一个名字为“**test**”的**SQLite**数据库，如下：

```
sqlite3 test.db
```

```
SQLite version 3.7.14
```

```
Enter ".help" for instructions
```

```
sqlite>
```

```
sqlite> .tables
```

```
sqlite> .exit
```

```
$ ls
```

```
test.db
```

.tables列出**test.db**中所有表。

.exit命令退出**sqlite3**

一、利用SQL语句操作SQLite数据库

- ▶ 下面为创建学生表及插入数据的SQL语句。

```
sqlite> create table student(id,name,sex,age);  
sqlite> insert into student values(1,'Jack','M',20);  
sqlite> insert into student values(2,'Tom','M',21);  
sqlite> insert into student values(3,'Mary','M',19);  
sqlite> select *from student;
```

一、利用SQL语句操作SQLite数据库

创建学生表:

id	name	sex	age
1	Jack	M	20
2	Tom	M	21
3	Mary	M	19

- ▶ 显示结果如下：

```
1|Jack|M|20  
2|Tom|M|21  
3|Mary|M|19
```

- ▶ 以下为更新表的操作，将学生表中age=20的项更新为age=24.

```
sqlite> update student set age=24 where age=20;  
sqlite> select *from student;
```

- ▶ 以上操作的结果打印如下：

```
1|Jack|M|24  
2|Tom|M|21  
3|Mary|M|19
```

- ▶ 执行完对数据库的操作后，使用.quit命令退出数据库

```
sqlite> .quit  
[root@localhost bin]#
```


10.4

Part Four

sqlite的数据类型

SQLite采用的是弱类型的数据。**SQLite 3.0**及以后版本支持更多的数据类型。每个数据值本身的数据类型可以是下列五种类型对象之一：

- (1) **NULL**，空值；
- (2) **INTEGER**，整型，根据大小使用1、2、3、4、6、8个字节来存储；
- (3) **REAL**，浮点型，用来存储8个字节的**IEEE**浮点；
- (4) **TEXT**，文本字符串，使用**UTF-8**、**UTF-16**、**UTF-32**等保存数据；
- (5) **BLOB**（**Binary Large Objects**），二进制类型，按照二进制存储，不做任何改变。

10.5 Part Five

sqlite的API函数

从功能的角度来区分，SQLite的API可分为两类：**核心API**和**扩充API**。

核心API由所有完成基本数据库操作的函数构成，主要包括连接数据库、执行SQL和遍历结果集等。它还包括一些功能函数，用来完成字符串格式化、操作控制、调试和错误处理等任务。

扩充API提供不同的方法来扩展SQLite，它向用户提供创建自定义的SQL扩展，并与SQLite本身的SQL相集成等功能。

10.5.1 核心C API函数

核心C API主要与执行SQL命令有关。核心C API大约有十个。它们分别是：

sqlite3_open()

sqlite3_prepare()

sqlite3_step()

sqlite3_column()

sqlite3_finalize()

sqlite3_close()

sqlite3_exec()

sqlite3_get_table()

sqlite3_reset()

sqlite3_bind()

有两种方法执行SQL语句：预编译查询(Prepared Query)和封装查询(有回调)。预编译查询由三个阶段构成：准备(preparation)、执行(execution)和定案(finalization)。而封装查询只是对预编译查询的三个过程进行了封装，最终也会转化为预编译查询来执行。

预处理查询是SQLite执行所有SQL命令的方式，主要包括以下三个步聚：

(1) 准备

分词器、分析器和代码生成器把SQL语句编译成虚拟机字节码，编译器会创建一个Sql语句句柄(sqlite3_stmt)，它包括字节码以及其它执行命令和遍历结果集所需的全部资源。相应的C API为sqlite3_prepare()，位于prepare.c文件中，有多种类似的形式，如sqlite3_prepare()、sqlite3_prepare16()、sqlite3_prepare_v2()等。这个工作就是把Sql语句编译成字节码

完整的API语法是：

```
int sqlite3_prepare_v2(  
    sqlite3 *db,          /* db为sqlite3的句柄*/  
    const char *zSql,     /* zSql为要执行的SQL语句 */  
    int nByte,           /* nByte为要执行语句在zSql中的最大长度,  
                          -1表示自动计算*/  
    sqlite3_stmt **ppStmt, /* ppStmt为预编译后的句柄 */  
    const char **pzTail    /* pzTail预编译后剩下的字符串（未预  
编译成功或者多余的）的指针，一般传入0或者NULL即可*/  
)
```


(2) 执行

虚拟机执行字节码，执行过程是一个步进(**stepwise**)的过程，每一步由**sqlite3_step()**启动，并由虚拟机执行一段字节码。

(3) 定案

虚拟机关闭语句，释放资源。相应的C API为**sqlite3_finalize()**，它导致虚拟机结束程序运行并关闭语句句柄。如果事务是由人工控制开始的，它必须由人工控制进行提交或回卷，否则**sqlite3_finalize()**会返回一个错误。当**sqlite3_finalize()**执行成功，所有与语句对象关联的资源都将被释放。在自动提交模式下，还会释放关联的数据库锁。

sqlite3_open()或sqlite3_open16()函数

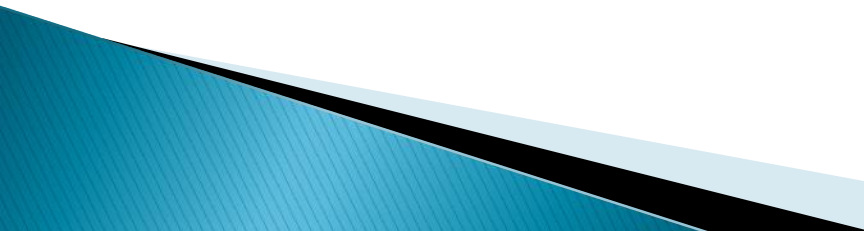
打开数据库用sqlite3_open()或sqlite3_open16()函数，它们的声明如下：

```
int sqlite3_open(  
    const char *filename,  
    sqlite3 **ppDb  
);
```

```
/* 数据库文件名 (UTF-8) */  
/* 输出数据库句柄 */
```

```
int sqlite3_open16(  
    const void *filename,  
    sqlite3 **ppDb  
);
```

```
/* 数据库文件名 (UTF-16) */  
/* 输出数据库句柄 */
```



sqlite3_close()函数

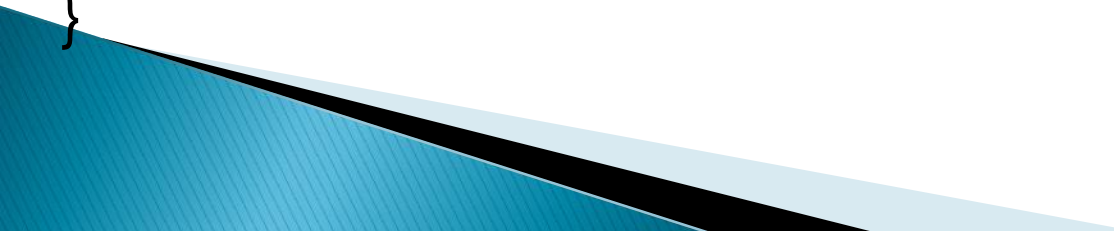
关闭数据库用sqlite3_close()函数，声明如下：

```
int sqlite3_close(sqlite3*);
```

为了sqlite3_close()能够成功执行，所有与连接所关联的且已编译的查询必须被定案。如果仍然有查询没有定案，sqlite3_close()将返回SQLITE_BUSY和错误信息。

Ex1:

```
int main(int argc, char **argv)
{
    int rc, i, ncols;
    sqlite3 *db;    //sqlite3结构体定义了数据库句柄
    sqlite3_stmt *stmt;// 该结构表示一个Sql语句预编译后句柄
    char *sql;
    const char *tail;
    rc = sqlite3_open_v2("test.db", &db);
    if(rc) {
        fprintf(stderr, "Can't open database: %s\n ", sqlite3_errmsg(db));
        sqlite3_close(db); .
        exit(1);
    }
```



```
sql ="select * from episodes; ";
rc = sqlite3_prepare_v2(db, sql, -1, &stmt, &tail);
if(rc != SQLITE_OK) {
    fprintf(stderr, "SQL error: %s\n", sqlite3_errmsg(db));
    sqlite3_close(db);
    exit(1);
}
rc = sqlite3_step(stmt);
ncols = sqlite3_column_count(stmt);
while(rc == SQLITE_ROW) {
    for(i=0; i < ncols; i++) {
        fprintf(stderr, "%s ", sqlite3_column_text(stmt, i));
    }
    fprintf(stderr, "\n");
    rc = sqlite3_step(stmt);
}
sqlite3_finalize(stmt);
sqlite3_close(db);
return 0;
}
```

程序的编译运行过程如下：

```
[root@JLUZH sqlite]# gcc -o ex1 ex1.c -lsqlite3
[root@JLUZH sqlite]# ./ex1
ID      name   age    num
1       LiMing  20     362302198901010214
2       LiSi    21     362302199005010254
3       WangWu  20     362302198905050954
[root@JLUZH sqlite]#
```

如果需要指定路径：

```
gcc ex1.c -o ex1 -lsqlite3 -L/usr/local/sqlite3/lib -I/usr/local/sqlite3/include
```

sqlite3_exec () 函数（封装查询）

大部分sql操作都可以通过sqlite3_exec来完成，它的API形式如下：

```
int sqlite3_exec(  
    sqlite3*,                               /* 数据库句柄 */  
    const char *sql,                         /* 要执行的SQL语句*/  
    int (*callback)(void*,int,char**,char**), /* callback回调函数 */  
    void *,                                  /* void *回调函数的第一个参数 */  
    char **errmsg                             /* errmsg错误信息，如果没有SQL问题则值为NULL */  
);
```

sqlite3_get_table () 函数

sqlite3_get_table的说明如下：

```
int sqlite3_get_table(  
    sqlite3 *db,          /* db是sqlite3的句柄*/  
    const char *zSql,     /* zSql是要执行的sql语句 */  
    char ***pazResult,    /* pazResult是执行查询操作的返回结果集 */  
    int *pnRow,           /* pnRow是记录的行数*/  
    int *pnColumn,        /* pnColumn是记录的字段个数 */  
    char **pzErrMsg       /* pzErrMsg是错误信息 */  
);
```

由于sqlite3_get_table是sqlite3_exec的包装，因此返回的结果和sqlite3_exec类似。对数据库进行查询操作时，可以通过该函数来获取结果集。该函数的入口参数为查询的SQL语句，出口参数有二维数据指针，指示查询结果的内容，还有结果集的行数和列数。

SQLite实例分析

```
/**Ex2.c***/  
#include<stdio.h>  
#include<sqlite3.h>  
int main()  
{  
    sqlite3 *db=NULL;  
    int rc;  
    char *ErrorMsg;  
    int row;  
    int col;  
    char **Result;  
    int i=0;  
    int j=0;  
    rc =sqlite3_open("test.db",&db);  
    if(rc)
```

```
{fprintf(stderr,"cant,t open:%s\n",sqlite3_errmsg(db));
  sqlite3_close(db);
  return 1;      }
else
{printf("open successly!\n");}
char *sql="create table people(ID integer primary key,name varchar(10),age
integer,num varchar(18))";
sqlite3_exec(db,sql,0,0,&Errormsg);
sql="insert into people values(1,'LiMing',20,'362302198901010214')";
sqlite3_exec(db,sql,0,0,&Errormsg);
sql="insert into people values(2,'LiSi',21,'362302199005010254')";
sqlite3_exec(db,sql,0,0,&Errormsg);
sql="insert into people values(3,'WangWu',20,'362302198905050954')";
sqlite3_exec(db,sql,0,0,&Errormsg);
sql="select * from people";
```

```
sqlite3_get_table(db,sql,&Result,&row,&col,&ErrorMsg);  
printf("row=%d column=%d\n\n",row,col);  
for(i=0;i<row+1;i++)  
{  
    for(j=0;j<col;j++)  
        {printf("%s\t",Result[j+i*col]);}  
        printf("\n");  
}  
sqlite3_free(ErrorMsg);  
sqlite3_free_table(Result);  
sqlite3_close(db);  
return 0;  
}
```

使用回调函数

回调函数是一个比较复杂的函数。原型如下：

```
int callback(void *params,int column_size,
char **column_value,char **column_name)
```

参数说明如下：

params是sqlite3_exec传入的第四个参数；

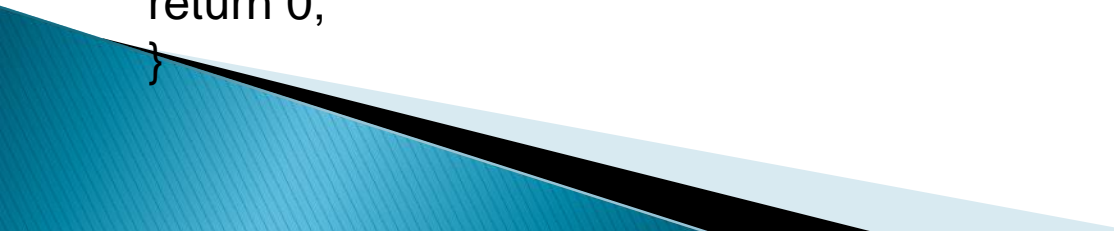
column_size是结果字段的个数；

column_value是返回记录的一位字符数组指针；


column_name是结果字段的名称；

Ex3:

```
int callback(void *, int, char **, char **);
int main(void)
{
    sqlite3 *db;
    char *err_msg = 0;
    int rc = sqlite3_open("test.db", &db);
    if (rc != SQLITE_OK) {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return 1; }
    char *sql = "SELECT * FROM Cars";
    rc = sqlite3_exec(db, sql, callback, 0, &err_msg);
    if (rc != SQLITE_OK ) {
        fprintf(stderr, "Failed to select data\n");
        fprintf(stderr, "SQL error: %s\n", err_msg);
        sqlite3_free(err_msg);
        sqlite3_close(db);
        return 1; }
    sqlite3_close(db);
    return 0;
}
```



```
int callback(void *NotUsed, int argc, char **argv, char **azColName) {  
    NotUsed = 0;  
    for (int i = 0; i < argc; i++) {  
        printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");  
    }  
    printf("\n");  
    return 0;  
}
```



10.5.2 扩充C API 函数

SQLite的扩充API用来支持用户定义的函数、聚合和排序法。用户定义函数是一个SQL函数，它对应于用C语言或其它语言实现的函数的句柄。使用C API时，这些句柄用C/C++实现。用户定义的扩展必须注册到一个由连接到连接的基础之上，存储在程序内存中。也就是说，它们不是存储在数据库中，而是存储在用户的程序中。