# SQLite的基本使用

梁夏旭

[liangxx@cndatacom.com](mailto:liangxx@cndatacom.com)

2012-08

# 目录

[1. SQLite的简介 1](#_Toc335743971)

[1.1. SQLite的特性 1](#_Toc335743972)

[1.1.1. 独立self-contained 1](#_Toc335743973)

[1.1.2. 无服务器serverless 1](#_Toc335743974)

[1.1.3. 零配置zero-configuration 1](#_Toc335743975)

[1.1.4. 交互式transactional 1](#_Toc335743976)

[1.1.5. 开源的open-source 1](#_Toc335743977)

[1.2. SQLite的下载 1](#_Toc335743978)

[1.2.1. SQLite组件 1](#_Toc335743979)

[1.2.2. 安装包命令规则 2](#_Toc335743980)

[1.2.3. 下载的安装包 2](#_Toc335743981)

[2. SQLite安装 2](#_Toc335743982)

[2.1. Linux平台源码包安装 2](#_Toc335743983)

[2.2. Windows平台安装 4](#_Toc335743984)

[3. SQLite的SQL使用 5](#_Toc335743985)

[3.1. 建立数据库 6](#_Toc335743986)

[3.2. 建表 6](#_Toc335743987)

[3.2.1. 语法 6](#_Toc335743988)

[3.2.2. 操作实例 9](#_Toc335743989)

[3.3. 插入数据 10](#_Toc335743990)

[3.3.1. 语法 10](#_Toc335743991)

[3.3.2. 操作实例 10](#_Toc335743992)

[3.4. 查询 12](#_Toc335743993)

[3.4.1. 语法 12](#_Toc335743994)

[3.4.2. 操作实例 13](#_Toc335743995)

[3.5. 删除数据 13](#_Toc335743996)

[3.5.1. 语法 13](#_Toc335743997)

[3.5.2. 操作实例 13](#_Toc335743998)

[3.6. 删表 14](#_Toc335743999)

[3.6.1. 语法 14](#_Toc335744000)

[3.6.2. 操作实例 14](#_Toc335744001)

[3.7. SQLite关键字 14](#_Toc335744002)

[4. SQLite的编程接口 15](#_Toc335744003)

[4.1. C/C++接口 15](#_Toc335744004)

[4.1.1. 两个重要的对象（并非面向对象编程中的术语对象） 15](#_Toc335744005)

[4.1.2. C/C++接口程序（程序族） 16](#_Toc335744006)

[4.1.3. 接口使用方法 20](#_Toc335744007)

[4.1.4. 接口封装 21](#_Toc335744008)

[4.1.5. \*Sqlite3\_exec封装接口 22](#_Toc335744009)

[4.1.6. 绑定参数与重用Prepared Statements 23](#_Toc335744010)

[4.1.7. 扩展的SQLite 24](#_Toc335744011)

[4.1.8. 例子 25](#_Toc335744012)

[4.2. TCL接口（略） 29](#_Toc335744013)

## SQLite的简介

官网：<http://www.sqlite.org/>

### SQLite的特性

#### 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

#### 无服务器serverless

#### 零配置zero-configuration

#### 交互式transactional

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

#### 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

命名模板如下：

（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

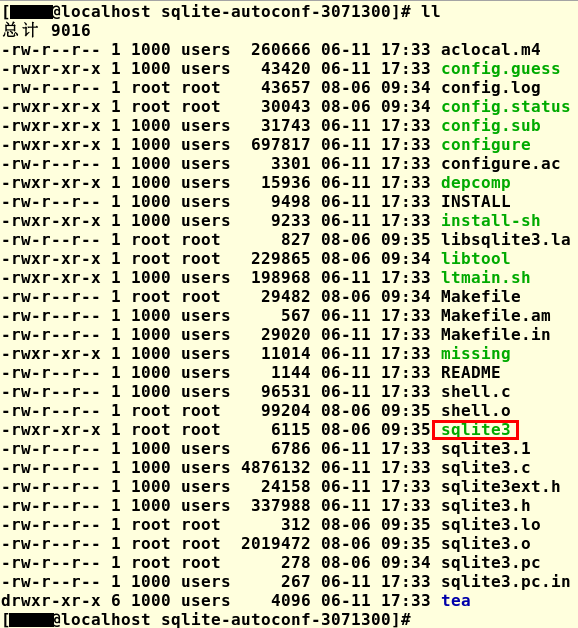
安装步骤：

$ ./configure

$ make

$ make install

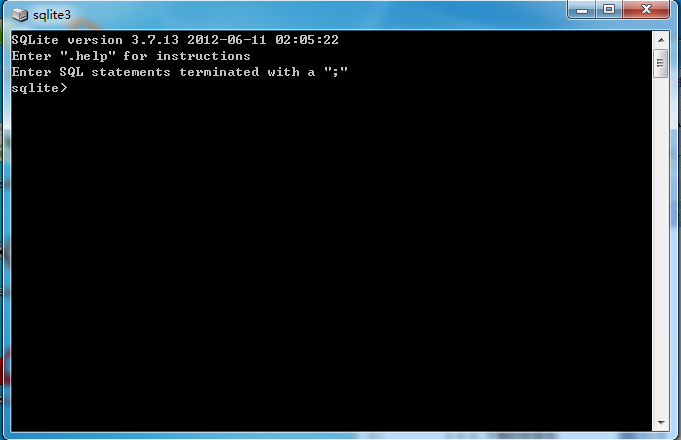
安装后就可以执行sqlite3命令



图表 0‑1生成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 0‑2 Windows shell

## SQLite的SQL使用

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，但是对于INTEGER primary是例外。）

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也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。

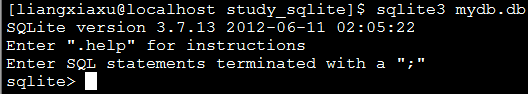
time：包含了小时、分钟、秒。

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 0‑3建立数据库

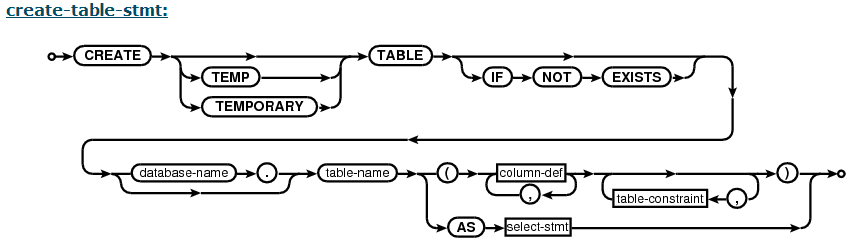
新建数据库mydb.db，若未对该数据进行建表操作（非临时表），再退出该数据库后，mydb.db也将被删除。

### 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 0‑4 Create-table-stmt

（2）temp字段

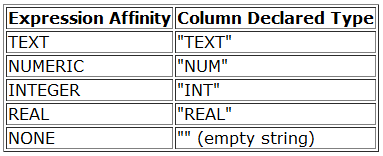
create与table之间加上temp或temporary表明数据表建立在**数据库temp**中，若加上关联的数据库即表明数据表建立在该关联数据库上，同时加上temp和关联数据库名是错误的语法，若没有加上任何数据库名，即将数据表建立在main数据库上。

（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 ……

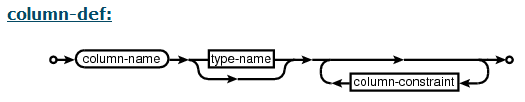
将根据select的结果建表，行列一致，命名一致，列数据项的数据类型将采用类似表述：



图表 0‑5 Expression affinity

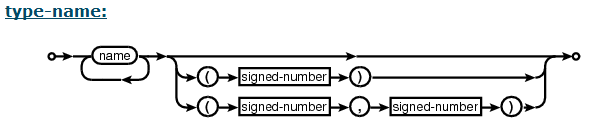
create table ……as select ……新建的表没有设置主键、约束关系，默认值为空。初始值为select返回的值，排序按照select结果。

（5）定义列（数据项）



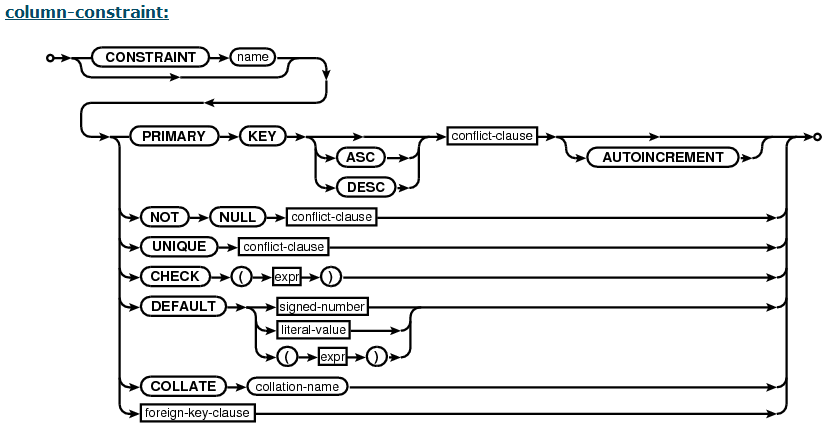
图表 0‑6 Column-def

数据项名+类型名+约束条件



图表 0‑7 Type-name

（6）6种数据约束



图表 0‑8 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。

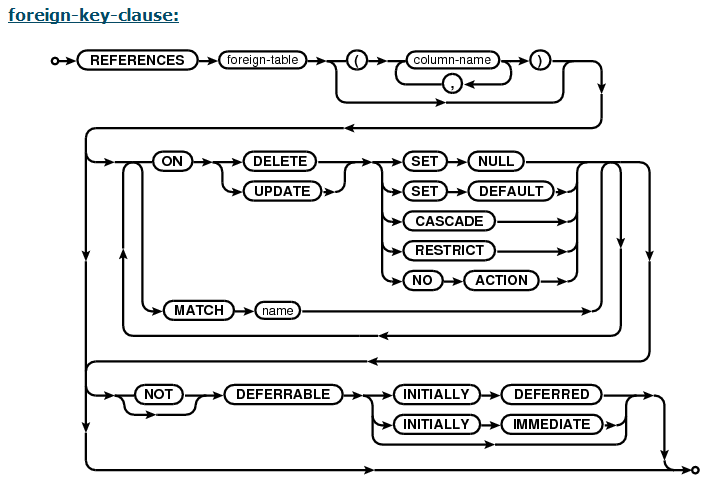
3、integer primary key

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 0‑9 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 0‑10 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 0‑11 Students（空）

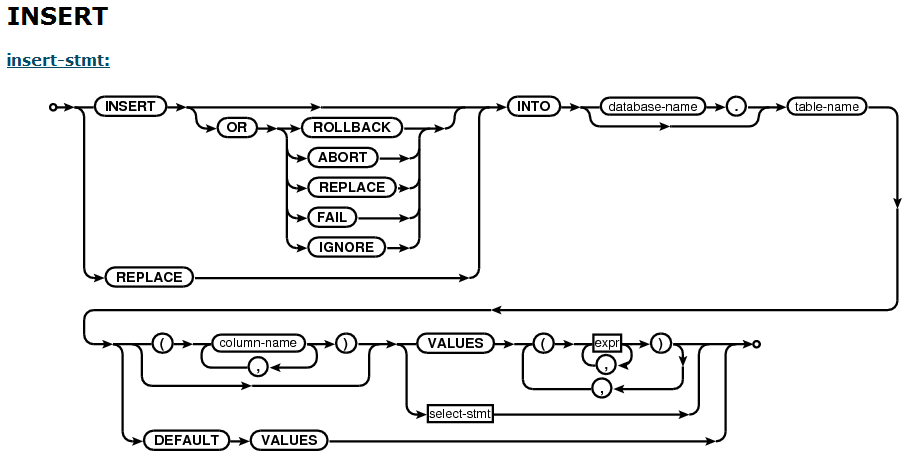
sqlite> **create table** students(ID integer primary key not null, NAME text not null, GENDER text not null, AGE integer not null, COURSE integer, foreign key(COURSE) references courses(ID));

sqlite> .table

courses students

### 插入数据

#### 语法



图表 0‑12 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 0‑13 courses

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

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

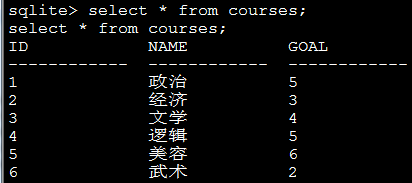
sqlite> **insert into** courses **values**(3, '文学', 4);

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 0‑14 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 0‑15 students

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

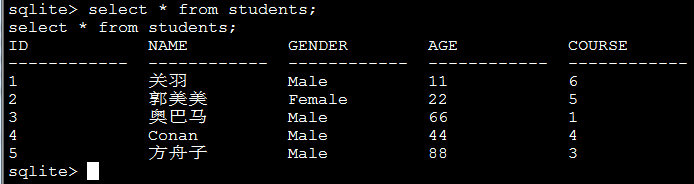
sqlite> **insert into** students **values**(2, '郭美美', 'Female', 22, 5);

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);

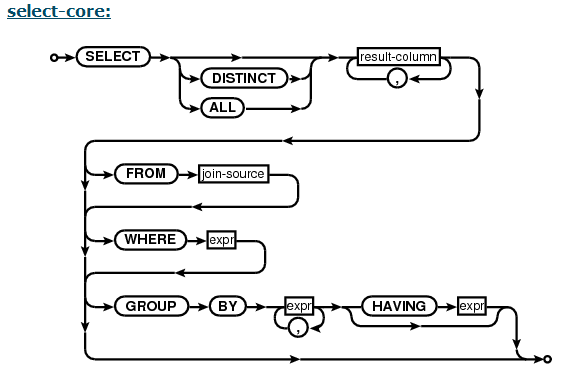
sqlite> select \* from students;



图表 0‑16 查询students

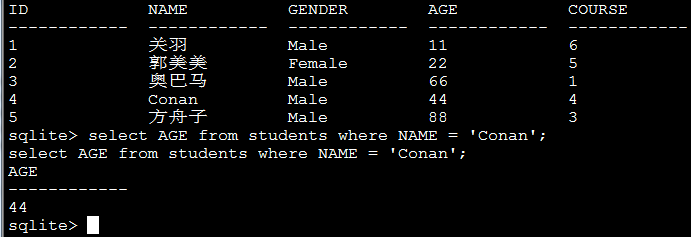
### 查询

#### 语法



图表 0‑17 Select-core

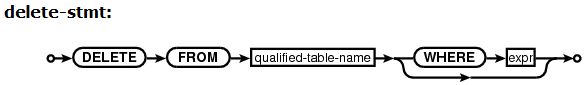
#### 操作实例



图表 0‑18 查询

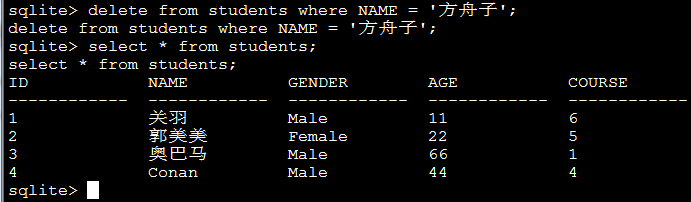
### 删除数据

#### 语法



图表 0‑19 Delete-stmt

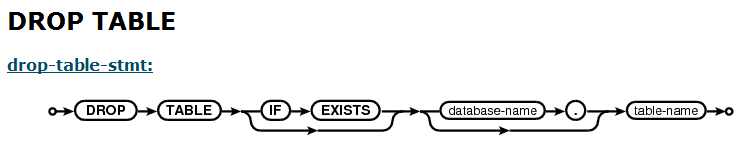
#### 操作实例



图表 0‑20 Delete一行

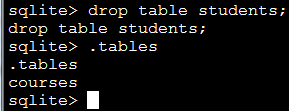
### 删表

#### 语法



图表 0‑21 Drop-table-stmt

#### 操作实例



图表 0‑22 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（程序族）

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()

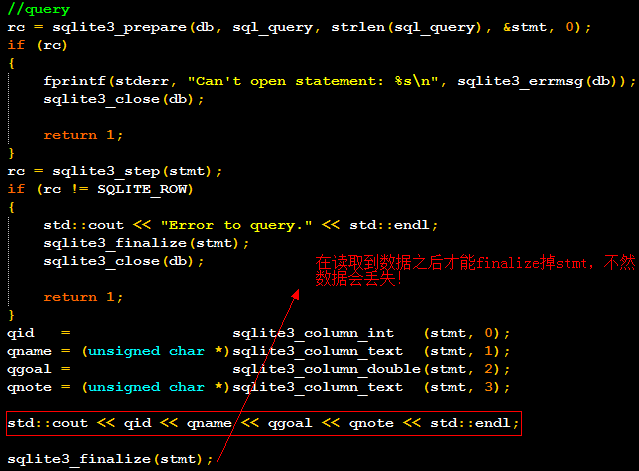
（4）sqlite3\_column()

（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）。**

****

**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

#### 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 0‑23 接口调用过程

#### 接口封装

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。

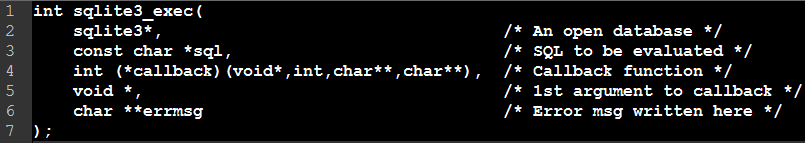
（2）sqlite3\_get\_table()是另一个集合了步骤（2）~（5）的封装接口，唯一的不同是它将结果存储在堆内存中而不是交给callback函数处理。



图表 0‑24 封装接口调用过程

#### \*Sqlite3\_exec封装接口

sqlite3\_exec：One-Step Query Execution Interface



图表 0‑25 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。

#### 绑定参数与重用Prepared Statements

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()

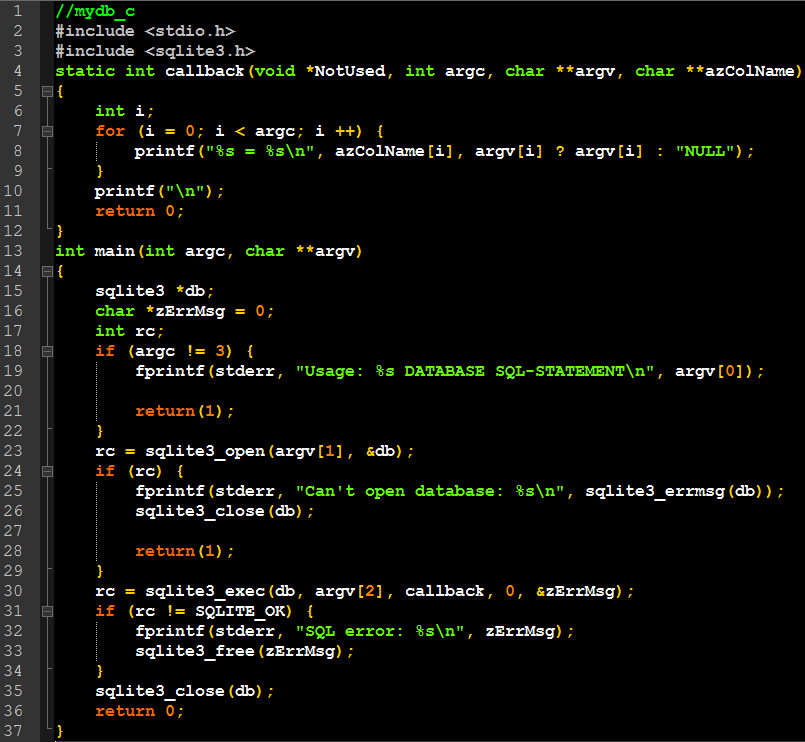
sqlite3\_result()

sqlite3\_user\_data()

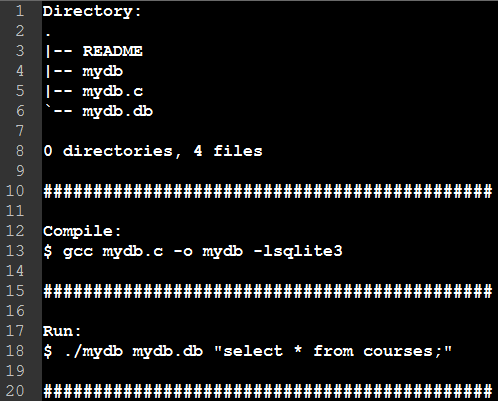
sqlite3\_value()

#### 例子

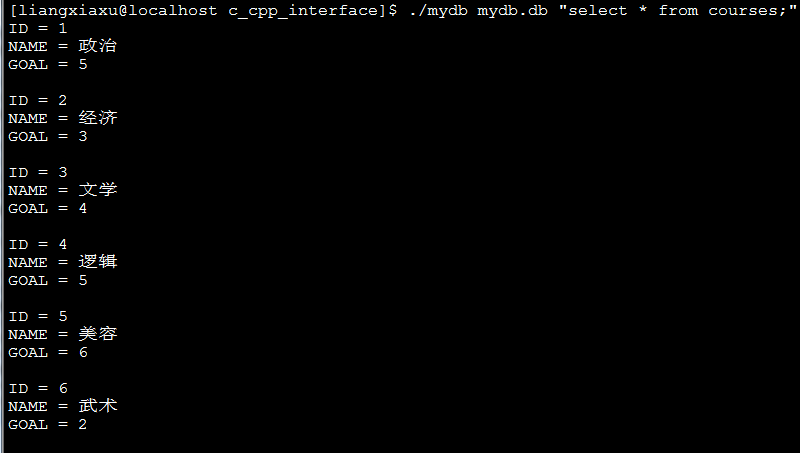
**sqlite3\_exec**



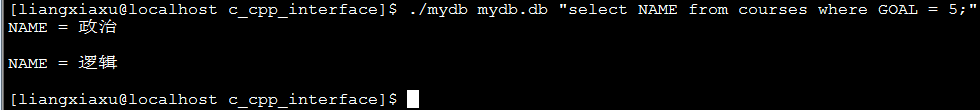
图表 0‑26 一个例子



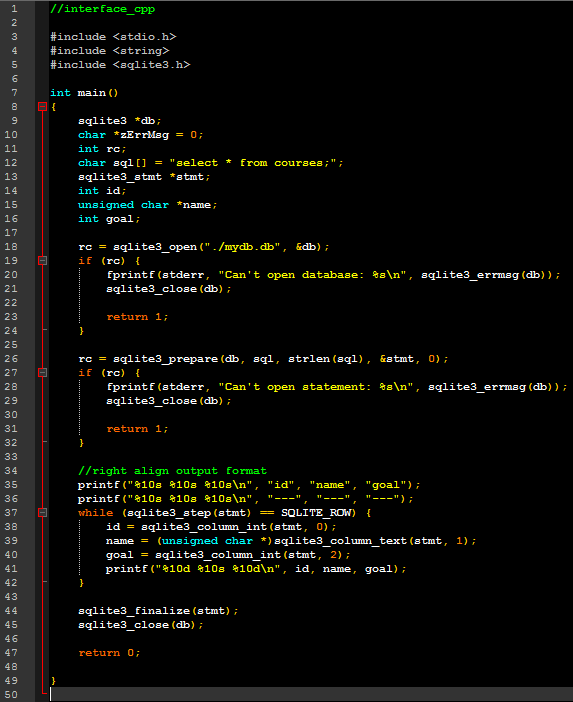
图表 0‑27 编译与使用示例



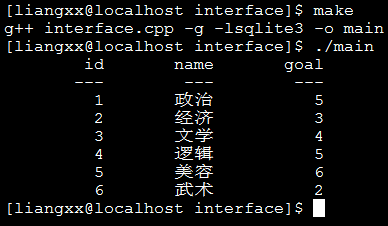
图表 0‑28 C/C++接口效果1



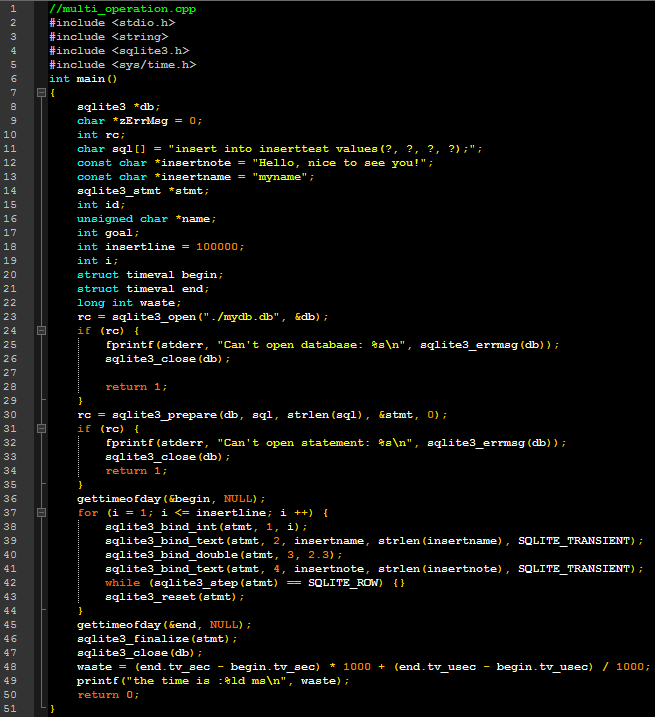
图表 0‑29 C/C++接口效果2



图表 0‑30 例子2分步接口



图表 0‑31 效果



图表 0‑32 例子3绑定与重置接口实现多次操作

### TCL接口（略）

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

# 图表索引

图表 0‑1生成sqlite3命令 4

图表 0‑2 Windows shell 5

图表 0‑3建立数据库 6

图表 0‑4 Create-table-stmt 7

图表 0‑5 Expression affinity 7

图表 0‑6 Column-def 7

图表 0‑7 Type-name 8

图表 0‑8 Column-constraint 8

图表 0‑9 Foreign-key-clause 9

图表 0‑10 courses（空） 9

图表 0‑11 Students（空） 10

图表 0‑12 Insert-stmt 10

图表 0‑13 courses 11

图表 0‑14 查询courses 11

图表 0‑15 students 11

图表 0‑16 查询students 12

图表 0‑17 Select-core 12

图表 0‑18 查询 13

图表 0‑19 Delete-stmt 13

图表 0‑20 Delete一行 13

图表 0‑21 Drop-table-stmt 14

图表 0‑22 Drop 14

图表 0‑23 接口调用过程 21

图表 0‑24 封装接口调用过程 22

图表 0‑25 Sqlite3\_exec接口参数 22

图表 0‑26 一个例子 25

图表 0‑27 编译与使用示例 26

图表 0‑28 C/C++接口效果1 26

图表 0‑29 C/C++接口效果2 26

图表 0‑30 例子2分步接口 27

图表 0‑31 效果 28

图表 0‑32 例子3绑定与重置接口实现多次操作 28