[**SQLite3命令操作大全**](http://blog.csdn.net/jiguanghoverli/article/details/8170165)

SQLite库包含一个名字叫做sqlite3的命令行,它可以让用户手工输入并执行面向SQLite数据库的SQL命令。本文档提供一个样使用sqlite3的简要说明.

# 一.qlite3一些常用Sql语句操作

**创建表**: create table 表名(元素名 类型,…);

**删除表**: drop table 表名;

**插入数据**: insert into 表名 values(, , ,) ;

**创建索引**: create [unique] index 索引名on 表名(col….);

**删除索引**： drop index 索引名(索引是不可更改的，想更改必须删除重新建)  
**删除数据**: delete from 表名;

**更新数据**: update 表名 set 字段=’修改后的内容’ where 条件;

**增加一个列**: Alter table 表名 add column 字段 数据类型;

**选择查询:** select 字段(以”,”隔开) from 表名 where 条件;

日期和时间**: Select datetime('now')**

日期**: select** date('now');   
时间**: select** time('now');

**总数：select count(\*) from table1;  
求和：select sum(field1) from table1;  
平均：select avg(field1) from table1;  
最大：select max(field1) from table1;  
最小：select min(field1) from table1;**

**排序：select 字段 from table1 order by 字段(desc或asc) ;(降序或升序)**

**分组：select 字段 from table1 group by 字段,字段… ;**

**限制输出:select 字段 from table1 limit x offset y;**

**=** **select 字段 from table1 limit y , x;**

**(备注:**跳过y行，取x行数据**)**

(操作仍待完善)…

## SQLite支持哪些数据类型些？

NULL 值为NULL  
INTEGER 值为带符号的整型，根据类别用1，2，3，4，6，8字节存储  
REAL 值为浮点型，8字节存储  
TEXT 值为text字符串，使用数据库编码(UTF-8, UTF-16BE or UTF-16-LE)存储  
BLOB 值为二进制数据，具体看实际输入

**但实际上，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是指小数点後有几位数。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。

如果将声明表的一列设置为 INTEGER PRIMARY KEY，则具有：

1．每当你在该列上插入一NULL值时， NULL自动被转换为一个比该列中最大值大1的一个整数；

2．如果表是空的， 将会是1；

## 算术函数

abs(X)返回给定数字表达式的绝对值。

max(X,Y[,...])返回表达式的最大值。

min(X,Y[,...])返回表达式的最小值。

random(\*)返回随机数。

round(X[,Y])返回数字表达式并四舍五入为指定的长度或精度。

## 字符处理函数

length(X)返回给定字符串表达式的字符个数。

lower(X)将大写字符数据转换为小写字符数据后返回字符表达式。

upper(X)返回将小写字符数据转换为大写的字符表达式。

substr(X,Y,Z)返回表达式的一部分。

randstr()

quote(A)

like(A,B)

确定给定的字符串是否与指定的模式匹配。

glob(A,B)

## 条件判断函数

coalesce(X,Y[,...])

ifnull(X,Y)

nullif(X,Y)

## 集合函数

avg(X)返回组中值的平均值。

count(X)返回组中项目的数量。

max(X)返回组中值的最大值。

min(X)返回组中值的最小值。

sum(X)返回表达式中所有值的和。

## 其他函数

typeof(X)返回数据的类型。

last\_insert\_rowid()返回最后插入的数据的 ID 。

sqlite\_version(\*)返回 SQLite 的版本。

change\_count()返回受上一语句影响的行数。

last\_statement\_change\_count()

# 二.有关事务的操作

(成批操作的时候，启动事务，比不启动事务快n倍)

开始事物处理

BEGIN TRANSACTION;

…………..

进行对数据库操作

…………..

事物提交

COMMIT;

具体事例如下:

假设有一个 t1 表，其中有 "a", "b", "c" 三列， 如果要删除列 c ，以下过程描述如何做:

BEGIN TRANSACTION;  
CREATE TEMPORARY TABLE t1\_backup(a,b);  
INSERT INTO t1\_backup SELECT a,b FROM t1;  
DROP TABLE t1;  
CREATE TABLE t1(a,b);  
INSERT INTO t1 SELECT a,b FROM t1\_backup;  
DROP TABLE t1\_backup;  
COMMIT;

# 三.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程序将提示你输入SQL。敲入SQL语句（以分号“；”结束），敲回车键之后，SQL语句就会执行。  
    例如，创建一个包含一个表"tb11"名字为"ex1"的SQLite数据库，你可以这样做：  
数据库、表的建立，记录的添加、查询、修改和删除  
F:\>sqlite3 database.db  
sqlite> create table admin(username text,age integer);  
sqlite> insert into admin values('kuang',25);  
sqlite> select \* from admin;  
sqlite> update admin set username='kk',age=24 where username='kuang' and age=25;  
sqlite> delete from admin where username='kk';  
注：每条sql语句后必须以";"号结尾！  
$sqlite3 ex1  
SQLite version 3.3.17  
Enter ".help" for instructions  
sqlite> create table tbl1(one varchar(10), two smallint);  
sqlite> insert into tbl1 values('hello!', 10);  
sqlite> insert into tbl1 values('goodbye', 20);  
sqlite> select \* from tbl1;  
hello!|10  
goodbye|20  
sqlite>  
        你可以通过敲你所用系统的文件结束符（通常是Ctrl + D）或者中断字符（通常是Ctrl + C）。来终止sqlite3程序。确定你在每个SQL语句结束敲入分号！sqlite3程序通过查找分号来决定一个SQL语句的结束。如果你省略分 号，sqlite3将给你一个连续的命令提示符并等你给当前的SQL命令添加更多的文字。这个特点让你输入多行的多个SQL语句，例如：  
sqlite> create table tbl2(  
   ...> f1 varchar(30) primary key,  
   ...> f2 text,  
   ...> f3 real  
   ...> );  
sqlite>

# 四.题外话：查询SQLITE\_MASTER表

    SQLite数据库的框架被保存在一个名叫"sqlite\_master"的特殊的表中。你可以像查询其它表一样通过执行“SELECT”查询这个特殊的表。例如：  
$ sqlite3 ex1  
SQlite vresion 3.3.10  
Enter ".help" for instructions  
sqlite> select \* from sqlite\_master;  
     type = table  
     name = tbl1  
tbl\_name = tbl1  
rootpage = 3  
      sql = create table tbl1(one varchar(10), two smallint)  
sqlite>  
    但你不能在sqlite\_master表中执行诸如DROP TABLE, UPDATE, INSERT 或者DELETE命令。sqlite\_master表在你创建、删除和索引数据库时自动更新这个表。你不能手工更改sqlite\_master表。  
    TEMPORARY表的结构没有存储在"sqlite\_master"表中，由于TEMPORARY表对应用是不可见的，而不是应用程序创建这个表。 TEMPORARY表结构被存储在另外一个名叫"sqlite\_temp\_master"的特定的表中。"sqlite\_temp\_master"表是临 时表自身。

# 五.sqlite3的特殊命令

   大多数候，sqlite3读入输入行，并把它们传递到SQLite库中去运行。但是如果输入行以一个点（“.”）开始，那么这行将被sqlite3程序自 己截取并解释。这些“点命令”通常被用来改变查询输出的格式，或者执行鞭个预封包（预定义prepackaged）的查询语句。  
你可以在任何时候输入“.help”，列出可用的点命令。例如  
sqlite> .help

## .bail ON|OFF

Stop after hitting an error.  Default OFF(遇到错误时不再继续, 默认为OFF)

## .databases

 List names and files of attached databases(列出附加到数据库的数据库和文件)

## .dump ?TABLE? ...

Dump the database in an SQL text format(保存表到SQL格式的文件中, 没有指表名, 则保存所有. 如果要保存到磁盘上需要结合 .output 命令.)

## .echo ON|OFF

Turn command echo on or off(打开/关闭 命令行回显)

## .exit

Exit this program(退出该命令行)

## .explain ON|OFF

Turn output mode suitable for EXPLAIN on or off.( 以合适的方式显示表头, 不带参数则为开启)

## .header(s) ON|OFF

Turn display of headers on or off(是否显示表头, 和 .explain 差别不是很大)

## .help

Show this message(显示帮助信息)

## .import FILE TABLE

Import data from FILE into TABLE(从文件中导入表)

## .indices TABLE

Show names of all indices on TABLE(显示索引)

## .load FILE ?ENTRY?

Load an extension library(加载一个扩展库)

## .mode MODE ?TABLE?

Set output mode where MODE is one of:

(设置输出模式, 模式可以是以下几种):  
                          csv       Comma-separated values( 以逗号分隔的值)  
                          column    Left-aligned columns.   (See .width)( 表头左对齐(参见 .width))  
                          html      HTML <table> code(显示 HTML 代码)  
                          insert    SQL insert statements for TABLE( SQL插入语句)  
                          line      One value per line( 一行一个值)  
                          list      Values delimited by separator string(值用 string 分隔)  
                          tabs      Tab-separated values(以 tab 分隔的值)  
                          tcl       TCL list elements(TCL 列表元素)

## .nullvalue STRING

Print STRING in place of NULL values(以 STRING 代替 NULL 值的输出)

## .output FILENAME

Send output to FILENAME(输出到文件, 而不是显示在屏幕上)

## .output stdout

Send output to the screen(输出到屏幕上)

## .prompt MAIN CONTINUE

Replace the standard prompts(替换默认的命令提示信息, 默认就是 sqlite>)

## .quit

Exit this program(退出命令行)

## .read FILENAME

Execute SQL in FILENAME(执行 FILENAME 中的 SQL语句)

## .schema ?TABLE?

Show the CREATE statements(显示 CREATE 语句)

## .separator STRING

Change separator used by output mode and .import

## .show

Show the current values for various settings(显示各种设置)

## .tables ?PATTERN?

List names of tables matching a LIKE pattern(查看数据库的表列表)

## .timeout MS

Try opening locked tables for MS milliseconds(在 MS 时间内尝试打开被锁定的表)

## .width NUM NUM ...

Set column widths for "column" mode(设置 column 模式中的列的宽度)

## .timer ON|OFF

(显示CPU时间)

## .vfsname ?AUX?

(显示 VFS 栈信息)

## .restore ?DB? FILE

(从文件中还原数据到表,  默认表为 main)  
sqlite>

# 六.改变输出格式(.mode)

    sqlite3程序可以以八种不同的格式显示一个查询的结果："csv", "列", "html", "插入", "行", "制表"和"tcl"。你可以用".mode"点命令在这些输出格式之间切换。  
    默认的输出格式是“列表”。在列表模式下，每条查询结果记录被写在一行中并且每列之间以一个字符串分割符隔开。默认的分隔符是一个管道符号（“|”）。列表符号在当你输出查询结果到另外一个符加处理的程序（如AWK）中去是尤为有用。  
sqlite> .mode list  
sqlite> select \* from tbl1;  
hello|10  
goodbye|20  
sqlite>  
    你可以用“.separator”点命令来改变分界符。例如，为了把分割符改为一个逗号和一个空格，你可以这样做：  
sqlite> .separator ", "  
sqlite> select \* from tbl1;  
hello, 10  
goodbye, 20  
sqlite>  
    在“line"模式下，每一个位于条记录中的列在它自己那行显示。每行由列名、一个等号和列数据组成。下一条记录以一个空行隔开。这是一个行模式输出的例子：  
sqlite> .mode line  
sqlite> select \* from tbl1;  
one = hello  
two = 10  
one = goodbye  
two = 20  
sqlite>  
    在列模式下，每条记录在一个单独的行中以数据列对齐的方式显示。列如：  
sqlite> .mode column  
sqlite> select \* from tbl1;  
one          two         
----------   ----------  
hello        10          
goodbye      20          
sqlite>  
        在默认的情况下，每列至少10个字符宽。太宽(超过设定的宽度)的数据将被截取。你可以用“.width”命令来调整列宽。如下所示：  
sqlite> .width 12 6 (第一列12,第二列6)  
sqlite> select \* from tbl1;  
one            two     
------------   ------  
hello          10      
goodbye        20      
sqlite>  
     上面例子中".width"命令设置第一列宽为12第二列宽为6。其它的列宽不变。你可以指定与你查询结果需要的列数一样多的“.width”参数。  
    如果你指定一列宽为0，那么这个列宽将自动以下面三个数字中的最大值做为列宽：10、表头宽度和最宽的数据列的宽度。这可以让列自动调整宽度。每列的默认设置为自动调整的0值。  
    出现在输出开头两行的列标示可以用".header"点命令关闭。在上面的例子中，列标示是打开的。可以用下面的方法关闭列标示：  
sqlite> .header off  
sqlite> select \* from tbl1;  
hello          10      
goodbye        20      
sqlite>  
    另外一个有用的输出模式是"insert"。在插入模式下，被子格式化为看起来像SQL INSERT语句的样式。你可以用插入模式来产生文件（便于）以后用于不同数据库的输入。  
    当指定插入模式时，你必须给定一个特定参数就是要插入的表名。例如：  
sqlite> .mode insert new\_table  
sqlite> select \* from tbl1;  
INSERT INTO 'new\_table' VALUES('hello',10);  
INSERT INTO 'new\_table' VALUES('goodbye',20);  
sqlite>  
    最新的输出格式是“html”。在这种模式下，sqlite3把查询的结果写做XHTML表。开始的<TABLE>和结束 的</TABLE>（标记）没有写出，但有<TR>、<TH>和<TD>等分界符。html输出对 CGI来说是相当有用地。  
把结果写到文件中

# 七.查询数据库结构

     sqlite3程序提供几个有用的用于查询数据库结构的快捷命令。这些不是不可以用别的方式来实现。这些命令仅仅是一个快捷方式而已。  
    例如，为了查看数据库的表列表，你可以敲入“.tables”。  
sqlite> .tables  
tbl1  
tbl2  
sqlite>  
    “.tables”命令相似于设置列表模式然后执行接下来的查询：  
SELECT name FROM sqlite\_master WHERE type IN ('table','view') AND name NOT LIKE 'sqlite\_%'UNION ALL SELECT name FROM sqlite\_temp\_master WHERE type IN ('table','view') ORDER BY 1sqlite> .schema  
create table tbl1(one varchar(10), two smallint)  
CREATE TABLE tbl2 (  
  f1 varchar(30) primary key,  
   f2 text,  
   f3 real  
)  
sqlite> .schema tbl2  
CREATE TABLE tbl2 (  
   f1 varchar(30) primary key,  
   f2 text,  
   f3 real  
)  
sqlite>

# 八.事实上

你可以查看sqlite3的源代码（可以在源文件树的src/shell.c中），你可找到上面的具体的查询。  
“.indices” 命令作用类似的方式是列出特定表的所有的索引。“.indics”命令须一个参数即所要索引表的表名。最后，但不是至少，是“.schema”命令。不带 任何参数，“.schema”命令显示原始的用于创建当前数据库的CREATE TABLE和CREATE INDEX语句。如果你给".schema"命令一个表名，它显示原始的创建该表和它所有索引的CREATE语句。我们可以：  
".schema"命令可以用设置列表然后执行以下查询来实现：  
SELECT sql FROM (SELECT \* FROM sqlite\_master UNION ALL SELECT \* FROM sqlite\_temp\_master)WHERE type!='meta'ORDER BY tbl\_name, type DESC, name  
或者，如果你给".schema"命令一个参数，由于你只想得到一个表的结构，查询可以是这样：  
SELECT sql FROM (SELECT \* FROM sqlite\_master UNION ALL SELECT \* FROM sqlite\_temp\_master)WHERE type!='meta' AND sql NOT NULL AND name NOT LIKE 'sqlite\_%'ORDER BY substr(type,2,1), name  
你可以担供一个参数给.schema命令。如果这橛，查询可以是这样的：  
SELECT sql FROM (SELECT \* FROM sqlite\_master UNION ALL SELECT \* FROM sqlite\_temp\_master)WHERE tbl\_name LIKE '%s' AND type!='meta' AND sql NOT NULL AND name NOT LIKE 'sqlite\_%'ORDER BY substr(type,2,1), name  
在查询中“%S“为你的参数所取代。这使你可以询数据库结构的某个子集。  
sqlite> .schema %abc%  
与这些一起，“.table”命令也接受一个模式作为他的参数。如果你给“.table”一个参数，“%”将被前后扩展并且一个LIKE子句被附加到查询上。这让你可以列出只与特定模式相匹配的的表。  
“.databases” 命令显示所有当前连接打开的数据库的一个列表。将允许一次到少两个。第一个是“main”，最初打开的那个数据库。第二个是"temp"，用于临时表的数 据库。对于用ATTACH语句附加的数据也许有附加数据库列表。输出的第一列与之相联的数据库名，第二列是外部文件名。  
sqlite> .databases

# 九.将整个数据库转换为ASCII文本文件

".dump"命令成一个单一的ASCII文本文件。这个文件可以被用作管道传递给sqlite3命令来转换回数据库。  
一个最好的制作一个数据库档案拷贝的命令是：  
$ echo '.dump' | sqlite3 ex1 | gzip -c >ex1.dump.gz  
它产生一个一个名为ex1.dump.gz的文件，它包含了你以后或在其它机器上重构数据库的所有的信息。要重构数据库，只须敲入：  
$ zcat ex1.dump.gz | sqlite3 ex2  
这个文本格式是纯粹的SQL语句所以你可以用.dump命令来导出一个SQLite数据库到另一个常用的SQL数据库引擎。比如：  
$ createdb ex2  
$ sqlite3 ex1 .dump | psql ex2

# 十.Sqlite3 表结构和数据的导出及导入

**数据库结构**导出和导入:

“.output” 默认情况下，sqlte3把结送到标准输出。你可以用“.output”命令改变它。只须把输出文件名做为.output命令的输出参数然后所有后续查询结果将被写到那个文件中。

用“.output stdout”再一次改为标准输出。例如:

**全部导出**   
$sqlite3 data.db   
sqlite >.output dd   
sqlite >.dump

sqlite > .exit

$cat dd //以下是数据库的结构

PRAGMA foreign\_keys=OFF;

BEGIN TRANSACTION;

CREATE TABLE long(h,m);

INSERT INTO "long" VALUES('hhhhhh',2);

INSERT INTO "long" VALUES('hello',2);

INSERT INTO "long" VALUES('mall',5);

INSERT INTO "long" VALUES('suinvzi',8);

INSERT INTO "long" VALUES('meimei',5);

CREATE TABLE bak(m varchar(10), n int);

INSERT INTO "bak" VALUES('hhhhhh',2);

INSERT INTO "bak" VALUES('hell',2);

INSERT INTO "bak" VALUES('mall',5);

INSERT INTO "bak" VALUES('suinvzi',8);

INSERT INTO "bak" VALUES('meimei',5);

COMMIT;

**导出表结构**

sqlite >.output dd

sqlite >.schema

sqlite > .exit

$cat dd //以下是表的结构

CREATE TABLE bak(m varchar(10), n int);

CREATE TABLE long(h,m);

**全部导入 : (**用.read命令,执行 FILENAME 中的 SQL语句**)**  
sqlite3 mydb.db   
sqlite >.read dd

sqlite >.tables

bak long //导入的是空表bak,long

**数据的**导出和导入:

**导出:**例如：

$sqlite3 data.db  
sqlite> .mode list  
sqlite> .separator |  
sqlite> .output test\_file\_1   
sqlite> select \* from long;  
sqlite> .exit  
$ cat test\_file\_1   
hhhhhh|2

hello|2

mall|5

suinvzi|8

meimei|5  
$

**备注:**每次导出若为同一个文本,则正在导出的内容覆盖上次导出的内容

**导入:** 例如：**(**用.import 导入的文本 被导入的表 命令, 把文件中的数据导入表中**)**

$sqlite3 mydb.db

sqlite> .show

echo: off

explain: off

headers: off

mode: list

nullvalue: ""

output: stdout

separator: "|"

stats: off

width:  
sqlite> .import test\_file\_1 long  
sqlite> select \* from long;  
hhhhhh|2

hello|2

mall|5

suinvzi|8

meimei|5  
导入结束

**备注:**文本的内容所对应的数据分布与所要导入的表一致,元素间的分割符要与”.show”中separator: "|"的字符一致如果不一致可直接修改, 比如: sqlite>.separator "," 将分隔符转为逗号,不然导入会出错.

# 十一.其它的点命令

".explain" 命令可以被用来设置输出格式为“column” 并设置列宽为EXPLAIN命令看起来比较合理的宽度。EXPLAIN命令是SQLite特有的SQL扩展，它是对调试有用。如果任何常规的SQL被 EXPLAIN执行，那么SQL命令被分解并分析但并不执行。取而代之的是，虚拟机指令序列将被用于执行SQL命令并返回一个类似的查询结果。如：  
sqlite> .explain  
sqlite> explain delete from tbl1 where two<20;  
addr   opcode         p1      p2      p3            
----   ------------   -----   -----   -------------------------------------     
0      ListOpen       0       0                    
1      Open           0       1       tbl1          
2      Next           0       9                    
3      Field          0       1                    
4      Integer        20      0                    
5      Ge             0       2                    
6      Key            0       0                    
7      ListWrite      0       0                    
8      Goto           0       2                    
9      Noop           0       0                    
10     ListRewind     0       0                    
11     ListRead       0       14                   
12     Delete         0       0                    
13     Goto           0       11                   
14     ListClose      0       0  
“.timeout”命令设置sqlite3等待一个试图存储文件锁定请除直到错误返回的总时间。默认的超时值是0因此如果任何需要的数据库表或序列列被锁定时它将立即返回一个错误。  
最后，我们提到“.exit”命令它将导致sqlite3退出。

# 十二.在命令和脚本中使用sqlite3

一 个在脚本命令中使用sqlite3的方式是用“echo”或“cat”来产生一个命令序列在一个文件中，然后当从一个产生的命令行中重定向输入时调用 sqlite3。它有用并且适应许多环境。但作为一附加的便利是，sqlite3允许一个单一的SQL语句在命令行中作为数据库名后的第二个参数输入。当 sqlite3程序带着两个参数启动时，第二个参数被传递给SQLite库处理，查询以列表模式打印到标准输出，然后程序退出。这个机制被设计用于让 sqlite3容易于用于连接诸如"AWK"的程序。例如：  
$ sqlite3 ex1 'select \* from tbl1' |  
> awk '{printf "<tr><td>%s<td>%s\n",$1,$2 }'  
<tr><td>hello<td>10  
<tr><td>goodbye<td>20  
$

# 十三.结束命令行命令

SQLite 命令通常以一个分号结束。在一个命令行中你也可以用“GO”单词（大小写敏感）或者一个“/”斜线在它所在好行结束一个命令。这常被SQL Server和Oracle使用。这些将不在sqlite3\_exec()中有用，因为命令行在传递它们到函数之前把这些翻译为分号。

# 十四.参数化的SQL语句

sqlite可以在shell/dos command底下直接执行命令：

**对数据库进行SQL操作:**  
sqlite3数据库 "SQL语句 或 .命令;"

**输出 HTML 表格：**  
sqlite3 -html film.db "select \* from film;"

**将数据库「导出来」：**sqlite3 film.db ".dump" > output.sql  
利用输出的资料，建立一个一模一样的数据库（加上以上指令，就是标准的SQL数据库备份了）：

**将数据库「导进去」：**  
sqlite3 film.db < output.sql  
在大量插入资料时，你可能会需要先打这个指令：  
begin;  
插入完资料后要记得打这个指令，资料才会写进数据库中：  
commit;

# 十五.SQLITE深入------常见问题

## 1.如何建立自动增长字段?

简短回答：声明为 INTEGER PRIMARY KEY 的列将会自动增长 。

(如果是最大可能的主键 9223372036854775807，那个，将键值将是随机未使用的数。）如，有下列表：

CREATE TABLE t1(  
a INTEGER PRIMARY KEY,  
b INTEGER  
);  
在该表上，下列语句

INSERT INTO t1 VALUES(NULL,123);  
在逻辑上等价于：

INSERT INTO t1 VALUES((SELECT max(a) FROM t1)+1,123);  
有一个新的API叫做 sqlite3\_last\_insert\_rowid()， 它将返回最近插入的整数值。 注意该整数会比表中该列上的插入之前的最大值大1。 该键值在当前的表中是唯一的。但有可能与已从表中删除的值重叠。要想建立在整个表的生命周期中唯一的键值，需要在 INTEGER PRIMARY KEY 上增加AUTOINCREMENT声明。那么，新的键值将会比该表中曾能存在过的最大值大1。如果最大可能的整数值在数据表中曾经存在过，INSERT将会失败， 并返回SQLITE\_FULL错误代码。

## 2.多个应用程序或一个应用程序的多个实例可以同时访问同一个数据库文件吗？

多个进程可同时打开同一个数据库。多个进程可以同时进行SELECT 操作，但在任一时刻，只能有一个进程对数据库进行更改。

SQLite使用读、写锁控制对数据库的访问。（在Win95/98/ME等不支持读、写锁的系统下，使用一个概率性的模拟来代替。）但使用时要注意：如果数据库文件存放于一个NFS文件系统上，这种锁机制可能不能正常工作。 这是因为 fcntl() 文件锁在很多NFS上没有正确的实现。 在可能有多个进程同时访问数据库的时候，应该避免将数据库文件放到NFS上。在Windows上，Microsoft的文档中说：如果使用 FAT 文件系统而没有运行 share.exe 守护进程，那么锁可能是不能正常使用的。那些在Windows上有很多经验的人告诉我：对于网络文件，文件锁的实现有好多Bug，是靠不住的。如果他们说的是对的，那么在两台或多台Windows机器间共享数据库可能会引起不期望的问题。

我们意识到，没有其它嵌入式的 SQL 数据库引擎能象 SQLite 这样处理如此多的并发。SQLite允许多个进程同时打开一个数据库，同时读一个数据库。当有任何进程想要写时，它必须在更新过程中锁住数据库文件。但那通常只是几毫秒的时间。其它进程只需等待写进程干完活结束。典型地，其它嵌入式的SQL数据库引擎同时只允许一个进程连接到数据库。

但是，Client/Server数据库引擎（如 PostgreSQL, MySQL, 或 Oracle）通常支持更高级别的并发，并且允许多个进程同时写同一个数据库。这种机制在Client/Server结构的数据库上是可能的，因为总是有一个单一的服务器进程很好地控制、协调对数据库的访问。如果你的应用程序需要很多的并发，那么你应该考虑使用一个Client/Server 结构的数据库。但经验表明，很多应用程序需要的并发，往往比其设计者所想象的少得多。

当SQLite试图访问一个被其它进程锁住的文件时，缺省的行为是返回 SQLITE\_BUSY。 可以在C代码中使用 sqlite3\_busy\_handler() 或 sqlite3\_busy\_timeout() API 函数调整这一行为。

## 3.在SQLite数据库中如何列出所有的表和索引？

如果你运行 sqlite3 命令行来访问你的数据库，可以键入 “.tables”来获得所有表的列表。或者，你可以输入 “.schema” 来看整个数据库模式，包括所有的表的索引。输入这些命令，后面跟一个LIKE模式匹配可以限制显示的表。

在一个 C/C++ 程序中（或者脚本语言使用 Tcl/Ruby/Perl/Python 等） 你可以在一个特殊的名叫 SQLITE\_MASTER 上执行一个SELECT查询以获得所有 表的索引。每一个 SQLite 数据库都有一个叫 SQLITE\_MASTER 的表， 它定义数据库的模式。 SQLITE\_MASTER 表看起来如下：

CREATE TABLE sqlite\_master (  
type TEXT,  
name TEXT,  
tbl\_name TEXT,  
rootpage INTEGER,  
sql TEXT  
);  
对于表来说，type 字段永远是 'table'，name 字段永远是表的名字。所以，要获得数据库中所有表的列表，使用下列SELECT语句：

SELECT name FROM sqlite\_master  
WHERE type='table'  
ORDER BY name;  
对于索引，type 等于 'index', name 则是索引的名字，tbl\_name 是该索引所属的表的名字。不管是表还是索引，sql 字段是原先用 CREATE TABLE 或 CREATE INDEX 语句创建它们时的命令文本。对于自动创建的索引（用来实现 PRIMARY KEY 或 UNIQUE 约束），sql字段为NULL。

SQLITE\_MASTER 表是只读的。不能对它使用 UPDATE、INSERT 或 DELETE。 它会被 CREATE TABLE、CREATE INDEX、DROP TABLE 和 DROP INDEX 命令自动更新。

临时表不会出现在 SQLITE\_MASTER 表中。临时表及其索引和触发器存放在另外一个叫 SQLITE\_TEMP\_MASTER 的表中。SQLITE\_TEMP\_MASTER 跟 SQLITE\_MASTER 差不多，但它只是对于创建那些临时表的应用可见。如果要获得所有表的列表， 不管是永久的还是临时的，可以使用类似下面的命令：

SELECT name FROM  
   (SELECT \* FROM sqlite\_master UNION ALL  
    SELECT \* FROM sqlite\_temp\_master)  
WHERE type='table'  
ORDER BY name

## 4.在SQLite中，VARCHAR字段最长是多少？

SQLite 不强制 VARCHAR 的长度。 你可以在 SQLITE 中声明一个 VARCHAR(10)，SQLite还是可以很高兴地允许你放入500个字符。 并且这500个字符是原封不动的，它永远不会被截断。

## 5.SQLite支持二进制大对象吗？

SQLite 3.0 及以后版本允许你在任何列中存储 BLOB 数据。即使该列被声明为其它类型也可以。

在SQLite中，如何在一个表上添加或删除一列？

SQLite 有有限地 ALTER TABLE 支持。你可以使用它来在表的末尾增加一列，可更改表的名称。如果需要对表结构做更复杂的改变，则必须重新建表。重建时可以先将已存在的数据放到一个临时表中，删除原表， 创建新表，然后将数据从临时表中复制回来。

如，假设有一个 t1 表，其中有 "a", "b", "c" 三列， 如果要删除列 c ，以下过程描述如何做:

BEGIN TRANSACTION;  
CREATE TEMPORARY TABLE t1\_backup(a,b);  
INSERT INTO t1\_backup SELECT a,b FROM t1;  
DROP TABLE t1;  
CREATE TABLE t1(a,b);  
INSERT INTO t1 SELECT a,b FROM t1\_backup;  
DROP TABLE t1\_backup;  
COMMIT;

## 6.在数据库中删除了很多数据，但数据库文件没有变小，是Bug吗？

不是。当你从SQLite数据库中删除数据时， 未用的磁盘空间将会加入一个内部的“自由列表”中。 当你下次插入数据时，这部分空间可以重用。磁盘空间不会丢失，但也不会返还给操作系统。

如果删除了大量数据，而又想缩小数据库文件占用的空间，执行 VACUUM 命令。 VACUUM 将会从头重新组织数据库。这将会使用数据库有一个空的“自由链表”， 数据库文件也会最小。但要注意的是，VACUUM 的执行会需要一些时间（在SQLite开发时，在Linux上，大约每M字节需要半秒种），并且， 执行过程中需要原数据库文件至多两倍的临时磁盘空间。

对于 SQLite 3.1版本，一个 auto-vacumm 模式可以替代 VACUUM 命令。 可以使用 auto\_vacuum pragma 打开。

## 7.SQLITE\_SCHEMA error是什么错误？为什么会出现该错误？

当一个准备好的（prepared）SQL语句不再有效或者无法执行时，将返回一个 SQLITE\_SCHEMA 错误。发生该错误时，SQL语句必须使用 sqlite3\_prepare() API来重新编译. 在 SQLite 3 中, 一个 SQLITE\_SCHEMA 错误只会发生在用 sqlite3\_prepare()/sqlite3\_step()/sqlite3\_finalize() API 执行 SQL 时。而不会发生在使用 sqlite3\_exec()时。 在版本2中不是这样。

准备好的语句失效的最通常原因是：在语句准备好后， 数据库的模式又被修改了。另外的原因会发生在：

数据库离线：DETACHed.  
数据库被 VACUUMed  
一个用户存储过程定义被删除或改变。  
一个 collation 序列定义被删除或改变。  
认证函数被改变。  
在所有情况下，解决方法是重新编译并执行该SQL语句。 因为一个已准备好的语句可以由于其它进程改变数据库模式而失效，所有使用 sqlite3\_prepare()/sqlite3\_step()/sqlite3\_finalize() API 的代码都应准备处理 SQLITE\_SCHEMA 错误。下面给出一个例子：

    int rc;  
    sqlite3\_stmt \*pStmt;  
    char zSql[] = "SELECT .....";

    do {  
      /\* Compile the statement from SQL. Assume success. \*/  
      sqlite3\_prepare(pDb, zSql, -1, &pStmt, 0);

      while( SQLITE\_ROW==sqlite3\_step(pStmt) ){  
        /\* Do something with the row of available data \*/  
      }

      /\* Finalize the statement. If an SQLITE\_SCHEMA error has  
      \*\* occured, then the above call to sqlite3\_step() will have  
      \*\* returned SQLITE\_ERROR. sqlite3\_finalize() will return  
      \*\* SQLITE\_SCHEMA. In this case the loop will execute again.  
      \*/  
      rc = sqlite3\_finalize(pStmt);  
    } while( rc==SQLITE\_SCHEMA );

## 8.如何一次插入多个数据

Insert into SAMPLE(PRJNUM, PRJNAME, EMYNUM, EMYNAME, SALCATEGORY, SALPACKAGE)

values(100001, 'TPMS', 200001, 'Johnson', 'A', 2000), (100001, 'TPMS', 200002,

'Christine', 'B', 3000), (100001, 'TPMS', 200003, 'Kevin', 'C', 4000), (100002,

'TCT', 200001, 'Johnson', 'A', 2000), (100002, 'TCT', 200004, 'Apple', 'B',

3000);

## 9.如何备份数据库

先打开数据库test.db

sqlite>sqlite3 test.db

sqlite> .backup test.bak

## 10.如何恢复数据库

先打开数据库test.db

sqlite>sqlite3 test.db

sqlite> .restore test.bak

## 11.先判断表如果不存在，则再创建表

SQL可以如下写，供参考：

create table if not exists student(id integer primary key autoincrement,/

age smallint ,anchor smallint);

## 12.如何查询SQLite3小工具的版本

执行 select sqlite\_version();命令即可

# 十六.Sqlite数据库中索引的使用、索引的优缺点

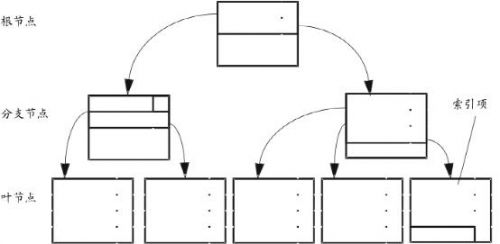
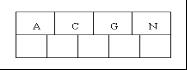
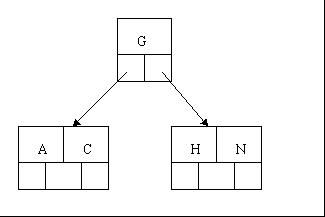
要使用索引对数据库的数据操作进行优化，那必须明确几个问题：  
1.什么是索引  
2.索引的原理  
3.索引的优缺点  
4.什么时候需要使用索引，如何使用  
围绕这几个问题，来探究索引在数据库操作中所起到的作用。

## 数据库索引简介

回忆一下小时候查字典的步骤，索引和字典目录的概念是一致的。字典目录可以让我们不用翻整本字典就找到我们需要的内容页数，然后翻到那一页就可以。索引也是一样，索引是对记录按照多个字段进行排序的一种展现。对表中的某个字段建立索引会创建另一种数据结构，其中保存着字段的值，每个值还包括指向与它相关记录的指针。这样，就不必要查询整个数据库，自然提升了查询效率。同时，索引的数据结构是经过排序的，因而可以对其执行二分查找，那就更快了。

## 2. B-树与索引

大多数的数据库都是以B-树或者B+树作为存储结构的，B树索引也是最常见的索引。先简单介绍下B-树，可以增强对索引的理解。  
B-树是为磁盘设计的一种多叉平衡树，B树的真正最准确的定义为：一棵含有t（t>=2）个关键字的平衡多路查找树。一棵M阶的B树满足以下条件：  
1）每个结点至多有M个孩子；  
2）除根结点和叶结点外，其它每个结点至少有M/2个孩子；  
3）根结点至少有两个孩子（除非该树仅包含一个结点）；  
4）所有叶结点在同一层，叶结点不包含任何关键字信息，可以看作一种外部节点；  
5）有K个关键字的非叶结点恰好包含K+1个孩子；  
 B树中的每个结点根据实际情况可以包含大量的关键字信息和分支(当然是不能超过磁盘块的大小，根据磁盘驱动(disk drives)的不同，一般块的大小在1k~4k左右)；这样树的深度降低了，这就意味着查找一个元素只要很少结点从外存磁盘中读入内存，很快访问到要查找的数据。B-树上操作的时间通常由存取磁盘的时间和CPU计算时间这两部分构成。而相对于磁盘的io速度，cpu的计算时间可以忽略不计，所以B树的意义就显现出来了，树的深度降低，而深度决定了io的读写次数。  
B树索引是一个典型的树结构，其包含的组件主要是：  
1）叶子节点（Leaf node）：包含条目直接指向表里的数据行。  
2）分支节点（Branch node）：包含的条目指向索引里其他的分支节点或者是叶子节点。  
3)  根节点（Root node）：一个B树索引只有一个根节点，它实际就是位于树的最顶端的分支节点。  
如下图所示：

  
  
   
每个索引都包含两部分内容，一部分是索引本身的值，第二部分即指向数据页或者另一个索引也的指针。每个节点即为一个索引页，包含了多个索引。  
当你为一个空表建立一个索引，数据库会分配一个空的索引页，这个索引页即代表根节点，在你插入数据之前，这个索引页都是空的。每当你插入数据，数据库就会在根节点创建索引条目，。当根节点插满的时候，再插入数据时，根节点就会分裂。举个例子，根节点插入了如图所示的数据。（超过4个就分裂），这时候插入H，就会分裂成2个节点，移动G到新的根节点，把H和N放在新的右孩子节点中。如图所示：  
      
　　　　根节点插满4个节点  
      
　　　　　　插入H，进行分裂。

大致的分裂步骤如下：  
1）创建两个儿子节点  
2）将原节点中的数据近似分为两半，写入两个新的孩子节点中。  
3）在跟节点中放置指向页节点的指针  
  
当你不断向表中插入数据，根节点中指向叶节点的指针也被插满，当叶子还需要分裂的时候，根节点没有空间再创建指向新的叶节点的指针。那么数据库就会创建分支节点。随着叶子节点的分裂，根节点中的指针都指向了这些分支节点。随着数据的不断插入，索引会增加更多的分支节点，使树结构变成这样的一个多级结构。

## 3. 索引的种类

1）聚集索引：表中行的物理顺序与键值的逻辑（索引）顺序相同。因为数据的物理顺序只能有一种，所以一张表只能有一个聚集索引。如果一张表没有聚集索引，那么这张表就没有顺序的概念，所有的新行都会插入到表的末尾。对于聚集索引，叶节点即存储了数据行，不再有单独的数据页。就比如说我小时候查字典从来不看目录，我觉得字典本身就是一个目录，比如查裴字，只需要翻到p字母开头的，再按顺序找到e。通过这个方法我每次都能最快的查到老师说的那个字，得到老师的表扬。

2）非聚集索引：表中行的物理顺序与索引顺序无关。对于非聚集索引，叶节点存储了索引字段值以及指向相应数据页的指针。叶节点紧邻在数据之上，对数据页的每一行都有相应的索引行与之对应。有时候查字典，我并不知道这个字读什么，那我就不得不通过字典目录的“部首”来查找了。这时候我会发现，目录中的排序和实际正文的排序是不一样的，这对我来说很苦恼，因为我不能比别人快了，我需要先再目录中找到这个字，再根据页数去找到正文中的字。

## 4．索引与数据的查询，插入与删除

1）查询。查询操作就和查字典是一样的。当我们去查找指定记录时，数据库会先查找根节点，将待查数据与根节点的数据进行比较，再通过根节点的指针查询下一个记录，直到找到这个记录。这是一个简单的平衡树的二分搜索的过程，我就不赘述了。在聚集索引中，找到页节点即找到了数据行，而在非聚集索引中，我们还需要再去读取数据页。

2）插入。聚集索引的插入操作比较复杂，最简单的情况，插入操作会找到对于的数据页，然后为新数据腾出空间，执行插入操作。如果该数据页已经没有空间，那就需要拆分数据页，这是一个非常耗费资源的操作。对于仅有非聚集索引的表，插入只需在表的末尾插入即可。如果也包含了聚集索引，那么也会执行聚集索引需要的插入操作。

3）删除。删除行后下方的数据会向上移动以填补空缺。如果删除的数据是该数据页的最后一行，那么这个数据页会被回收，它的前后一页的指针会被改变，被回收的数据页也会在特定的情况被重新使用。与此同时，对于聚集索引，如果索引页只剩一条记录，那么该记录可能会移动到邻近的索引表中，原来的索引页也会被回收。而非聚集索引没办法做到这一点，这就会导致出现多个数据页都只有少量数据的情况。

## 5. 索引的优缺点

其实通过前面的介绍，索引的优缺点已经一目了然。  
先说优点：  
    1）大大加快数据的检索速度，这也是创建索引的最主要的原因  
    2）加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

    3）在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

   再说缺点：  
  1）创建索引需要耗费一定的时间，但是问题不大，一般索引只要build一次  
  2）索引需要占用物理空间，特别是聚集索引，需要较大的空间

  3）当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，降低了数据的维护速度，这个是比较大的问题。

## 6. 索引的使用

 根据上文的分析，我们大致对什么时候使用索引有了自己的想法（如果你没有，回头再看一遍。。。）。一般我们需要在这些列上建立索引：  
1）在经常需要搜索的列上，这是毋庸置疑的；   
2）经常同时对多列进行查询，且每列都含有重复值可以建立组合索引，组合索引尽量要使常用查询形成索引覆盖（查询中包含的所需字段皆包含于一个索引中，我们只需要搜索索引页即可完成查询）。 同时，该组合索引的前导列一定要是使用最频繁的列。对于前导列的问题，在后面sqlite的索引使用介绍中还会做讨论。  
3）在经常用在连接的列上，这些列主要是一些外键，可以加快连接的速度，连接条件要充分考虑带有索引的表。；

4）在经常需要对范围进行搜索的列上创建索引，因为索引已经排序，其指定的范围是连续的，同样，在经常需要排序的列上最好也创建索引。

6）在经常放到where子句中的列上面创建索引，加快条件的判断速度。要注意的是where字句中对列的任何操作（如计算表达式，函数）都需要对表进行整表搜索，而没有使用该列的索引。所以查询时尽量把操作移到等号右边。

对于以下的列我们不应该创建索引：  
1）很少在查询中使用的列  
2）含有很少非重复数据值的列，比如只有0，1，这时候扫描整表通常会更有效  
3）对于定义为TEXT，IMAGE的数据不应该创建索引。这些字段长度不固定，或许很长，或许为空。  
当然，对于更新操作远大于查询操作时，不建立索引。也可以考虑在大规模的更新操作前drop索引，之后重新创建，不过这就需要把创建索引对资源的消耗考虑在内。总之，使用索引需要平衡投入与产出，找到一个产出最好的点。

## 7. 在sqlite中使用索引

1）Sqlite不支持聚集索引，android默认需要一个\_id字段，这保证了你插入的数据会按“\_id”的整数顺序插入，这个integer类型的主键就会扮演和聚集索引一样的角色。所以不要再在对于声明为：INTEGER PRIMARY KEY的主键上创建索引。

2）很多对索引不熟悉的朋友在表中创建了索引，却发现没有生效，其实这大多数和我接下来讲的有关。对于where子句中出现的列要想索引生效，会有一些限制，这就和前导列有关。所谓前导列，就是在创建复合索引语句的第一列或者连续的多列。比如通过：CREATE INDEX comp\_ind ON table1(x, y, z)创建索引，那么x,xy,xyz都是前导列，而yz，y，z这样的就不是。下面讲的这些，对于其他数据库或许会有一些小的差别，这里以sqlite为标准。在where子句中，前导列必须使用等于或者in操作，最右边的列可以使用不等式，这样索引才可以完全生效。同时，where子句中的列不需要全建立了索引，但是必须保证建立索引的列之间没有间隙。举几个例子来看吧：

用如下语句创建索引：  
CREATE INDEX idx\_ex1 ON ex1(a,b,c,d,e,...,y,z);  
这里是一个查询语句：  
...WHERE a=5 AND b IN (1,2,3) AND c IS NULL AND d='hello'  
这显然对于abcd四列都是有效的，因为只有等于和in操作，并且是前导列。  
再看一个查询语句：  
... WHERE a=5 AND b IN (1,2,3) AND c>12 AND d='hello'  
那这里只有a，b和c的索引会是有效的，d列的索引会失效，因为它在c列的右边，而c列使用了不等式，根据使用不等式的限制，c列已经属于最右边。  
最后再看一条：  
... WHERE b IN (1,2,3) AND c NOT NULL AND d='hello'

索引将不会被使用，因为没有使用前导列，这个查询会是一个全表查询。

3）对于between，or，like，都无法使用索引。  
如 ...WHERE myfield BETWEEN 10 and 20;  
这时就应该将其转换成：  
    ...WHERE myfield >= 10 AND myfield <= 20;  
再如LIKE：...mytable WHERE myfield LIKE 'sql%';;  
此时应该将它转换成：  
...WHERE myfield >= 'sql' AND myfield < 'sqm';  
    再如OR：...WHERE myfield = 'abc' OR myfield = 'xyz';  
此时应该将它转换成：  
...WHERE myfield IN ('abc', 'xyz');

其实除了索引，对查询性能的影响因素还有很多，比如表的连接，是否排序等。影响数据库操作的整体性能就需要考虑更多因素，使用更对的技巧，不得不说这是一个很大的学问。

最后在android上使用sqlite写一个简单的例子，看下索引对数据库操作的影响。  
创建如下表和索引：  
   db.execSQL("create table if not exists t1(a,b)");          
   db.execSQL("create index if not exists ia on t1(a,b)");  
插入10万条数据，分别对表进行如下操作：  
select \* from t1 where a='90012'  
插入：insert into t1(a,b) values('10008','name1.6982235534984673')  
更新：update t1 set b='name1.999999' where a = '887'

删除：delete from t1 where a = '1010'

数据如下（5次不同的操作取平均值）：  
操作   无索引    有索引  
查询   170ms  5ms  
插入   65ms   75ms  
更新   240ms  52ms  
删除   234ms  78ms

        可以看到显著提升了查询的速度，稍稍减慢了插入速度，还稍稍提升了更新数据和删除数据的速度。如果把更新和删除中的where子句中的列换成b，速度就和没有索引一样了，因为索引失效。所以索引能大幅度提升查询速度，对于删除和更新操作，如果where子句中的列使用了索引，即使需要重新build索引，有可能速度还是比不使用索引要快的。对与插入操作，索引显然是个负担。同时，索引让db的大小增加了2倍多。

       还有个要吐槽的是，android中的rawQurey方法，执行完sql语句后返回一个cursor，其实并没有完成一个查询操作，我在rawquery之前和之后计算查询时间，永远是1ms...这让我无比苦闷。看了下源码，在对cursor调用moveToNext这些移动游标方法时，都会最终先调用getCount方法，而getCount方法才会调用native方法调用真正的查询操作。这种设计显然更加合理