SQLite 中文指南之FAQ第1/6页

sqllite使用过程中碰到的一些问题解决，中文版

1. 如何创建自增字段？

2. SQLite 支持哪些数据类型？

3. 为什么能向 SQLite 数据库的整型字段中插入字符串？

4. 为什么 SQLite 认为表达式 '0'=='00' 为真？

5. 为什么 SQLite 不允许在同一张表里使用 '0' 和 '0.0' 作为两个不同的行的主键？

6. 为什么不能在 Linux box 中读取在 SparcStation 中创建的 SQLite 数据库？

7. 多个应用程序或者同一个应用程序的多个例程能同时存取同一个数据库文件吗？

8. SQLite是线程安全的吗?

9. 如何列出一个 SQLite 数据库中的所有的表/索引？

10. SQLite数据库是否有已知的大小限制？

11. 在 SQLite 中 VARCHAR 的最大长度是多少？

12. SQLite 是否支持 BLOB 类型？

13. 如何从一个已存在的 SQLite 数据表中添加/删除字段？

14. 我删除了很多数据但是数据库文件并没有减小，是不是 Bug？

15. 是否能将 SQLite 用于商业用途而不用交版权费用？

16. 我如何使用含有单引号(')的字符串？

17. SQLITE\_SCHEMA 错误代表什么？

18. 为什么ROUND(9.95,1) 返回 9.9 而不是 10.0? 难道9.95 不该向上进位么？

(1) 如何创建自增字段？

简单的回答：一个声明为 INTEGER PRIMARY KEY 的字段将自动增加。

这里是详细的答案： 从 SQLite 的 2.3.4 版本开始，如果你将一个表中的一个字段声明为 INTEGER PRIMARY KEY，那么无论你何时向该表的该字段插入一个 NULL 值，这个 NULL 值将自动被更换为比表中该字段所有行的最大值大 1 的整数；如果表为空，那么将被更换为 1。比如，假设你有这样的一张数据表:

CREATE TABLE t1(

a INTEGER PRIMARY KEY,

b INTEGER

);

在这张数据表里，声明

INSERT INTO t1 valueS(NULL,123);

在逻辑意义上等价于：

INSERT INTO t1 valueS((SELECT max(a) FROM t1)+1,123);

一个新的API函数 sqlite3\_last\_insert\_rowid() 返回最近的插入操作的整形键

注意这个整型键始终比之前插入表中的最后一个键大1。新键相对于表中的已有键来说是唯一的， 但它可能与之前从表中删除的键值重叠。要始终得到在整个表中唯一的键，在INTEGER PRIMARY KEY的声明之前加关键词AUTOINCREMENT.这样被选的键将总是比表中已存在的最大键大1。若可能的 最大键已存在于表中，INSERT操作将失败并返回一个SQLITE\_FULL错误码.

(2) SQLite 支持哪些数据类型？

参见 http://www.sqlite.org/datatype3.html.

(3) 为什么能向 SQLite 数据库的整型字段中插入字符串？

这是一个功能，不是一个 bug。你可以在任何字段中放任何信息，而不用管字段声明为什么类型。 你可以往整型字段中插入任意长度的字符串，或者往布尔字段中插入浮点数，或者往字符字段中 插入日期。在 CREATE TABLE 命令中你指定给这个字段的数据类型不会限制插入这个字段的数据。 所有的字段可以插入任意长度的字符串。但对于 INTEGER PRIMARY KEY 字段例外。这种字段只能 存放一个64位的整数，否则会出错。

但SQLite会默认你希望使用声明的字段类型。所以，比如你希望在一个声明为INTEGER的字段 中插入一个字符串，SQLite会试图将其转换为一个整数。如果转换成功，那么整数将被插入，否 则插入字符串，这种特性有时被称作type or column affinity.

(4) 为什么 SQLite 认为表达式 '0'=='00' 为真？

在 2.7.0 之后，表达式不成立。参见文档 datatypes in SQLite version 3

(5) 为什么 SQLite 不允许在同一张表里使用 '0' 和 '0.0' 作为两个不同的行的主键？

你的主键一定是数值类型的，把类型改为 TEXT 就可以了。

每一行必须有一个唯一的主键。作为一个数字类型的字段，SQLite 认为 '0' 和 '0.0'的值是相同的， 因为他们在数字上的比较是相等的（看前面的问题）因此值不是唯一的。

(6) 为什么不能在 Linux box 中读取在 SparcStation 中创建的 SQLite 数据库？

你需要升级你的 SQLite 库到 2.6.3 或更新版本。

x86 处理器是 little-endian 型的而 Sparc 是 big-endian 型的。新版本的 SQLite 解决了这个问题。

注： 　　big endian和little endian是CPU处理多字节数的不同方式。例如“汉”字的Unicode编码是 6C49。那么写到文件里时，究竟是将6C写在前面，还是将49写在前面？如果将6C写在前面，就是big endian。还是将49写在前面，就是 little endian。

(7) 多个应用程序或者同一个应用程序的多个例程能同时存取同一个数据库文件吗？

多进程可以同时打开同一个数据库，也可以同时 SELECT 。但只有一个进程可以立即改数据库。

SQLite使用读/写锁定来控制数据库访问。(Win95/98/ME 操作系统缺乏读/写锁定支持，在低于 2.7.0 的版本中，这意味着在 windows 下在同一时间内只能有一个进程读数据库。在版本 2.7.0 中 这个问题通过在 windows 接口代码中执行一个用户间隔几率读写锁定策略解决了。) 但如果数据库文件在一个 NFS 文件系统中，控制并发读书的锁定机制可以会出错。因为 NFS 的fcntl() 文件锁定有时会出问题。如果有多进程可能并发读数据库则因当避免把数据库文件放在 NFS 文件系统中。 根据微软的文档，如果不运行 Share.exe 后台程序则 FAT 文件系统中的锁定可能不工作。对 Windows 非常有经验的人告诉我网络文件的锁定有许多问题并且不可靠。如果是这样，在2个或以上 Windows 系统中共享一个 SQLite 数据库文件会导致不可预知的问题。

我们知道没有其他的嵌入式 SQL数据库引擎比SQLite支持更多的并发性。 SQLite允许多进程 同时打开和读取数据库。任何一个进程需要写入时，整个数据库将在这一过程中被锁定。但这一般仅耗时 几毫秒。其他进程只需等待然后继续其他事务。其他嵌入式SQL数据库引擎往往只允许单进程访问数据库。

但是，client/server型的数据库引擎 (如 PostgreSQL, MySQL, 以及 Oracle) 通常支持更高的并发度， 并支持多进程同时写入同一个数据库。由于总有一个控制良好的服务器协调数据库的访问，这才保证了以上特性的实现。如果你的应用需要很高的并发度，你应该考虑使用client/server数据库。事实上，经验告诉 我们大多数应用所需要的并发度比他们的设计者们想象的要少得多。

当 SQLite 尝试操作一个被另一个进程锁定的文件时，缺省的行为是返回 SQLITE\_BUSY。你可以用 C代码更改这一行为。 使用 sqlite3\_busy\_handler() 或sqlite3\_busy\_timeout() API函数。

如果两个或更多进程同时打开同一个数据库，其中一个进程创建了新的表或索引，则其它进程可能不能立即看见新的表。其它进程可能需要关闭并重新连结数据库。

(8) SQLite是线程安全的吗?

有时候是的。为了线程安全，SQLite 必须在编译时把 THREADSAFE 预处理宏设为1。在缺省的发行的已编译版本中 Windows 版的是线程安全的，而 Linux 版的不是。如果要求线程安全，Linux 版的要重新编译。

“线程安全”是指二个或三个线程可以同时调用独立的不同的sqlite3\_open() 返回的"sqlite3"结构。而不是在多线程中同时使用同一个 sqlite3 结构指针。

一个sqlite3结构只能在调用 sqlite3\_open创建它的那个进程中使用。你不能在一个线程中打开一个数据库然后把指针传递给另一个线程使用。这是因为大多数多线程系统的限制（或 Bugs？）例如RedHat9上。在这些有问题的系统上，一个 线程创建的fcntl()锁不能由另一个线程删除或修改。由于SQLite依赖fcntl()锁来进行并发控制，当在线程间传递数据库连接时会出现严重的问题。

也许在Linux下有办法解决fcntl()锁的问题，但那十分复杂并且对于正确性的测试将是极度困难的。因此，SQLite目前不允许在线程间共享句柄。

在UNIX下，你不能通过一个 fork() 系统调用把一个打开的 SQLite 数据库放入子过程中，否则会出错。

(9) 如何列出一个 SQLite 数据库中的所有的表/索引？

在sqlite3 命令行程序中你可以用命令 ".tables" 来显示所有的表或者用 ".schema"来显示所有的表结构和索引。但命令后不要跟 LIKE 语句，否则会限制表的显示。

在 C/C++ 程序中 (或使用 Tcl/Ruby/Perl/Python绑定的脚本中)你可以通过访问名为 "SQLITE\_MASTER的表来实现。每个 SQLite 数据库有一个 SQLITE\_MASTER 表，表内有数据库的结构。 SQLITE\_MASTER表是这样的：

CREATE TABLE sqlite\_master (

type TEXT,

name TEXT,

tbl\_name TEXT,

rootpage INTEGER,

sql TEXT

);

对于表来说，type字段的值为'table'，name 字段是表的名称。使用以下语句可以等到所有表的列表：

SELECT name FROM sqlite\_master

WHERE type='table'

ORDER BY name;

对于索引来说， type = 'index', name 是索引的名称， tbl\_name 是索引所属的表的名称。对于表和索引， sql 字段是创建表或索引的原始语句文本。对于自动创建的索引(一般是使用 PRIMARY KEY 或 UNIQUE 创建的)，sql字段为 NULL.

SQLITE\_MASTER表是只读的。你不能对该表使用 UPDATE, INSERT, 或 DELETE。该表自动由 CREATE TABLE, CREATE INDEX, DROP TABLE 和 DROP INDEX 命令更新。

临时表及其索引不在 SQLITE\_MASTER 表中而在 SQLITE\_TEMP\_MASTER 中出现。 SQLITE\_TEMP\_MASTER 与 SQLITE\_MASTER 表一样工作，但只对于创建临时表的程序可见。要得到所在表包括临时表可以使用如下命令：

SELECT name FROM

(SELECT \* FROM sqlite\_master UNION ALL

SELECT \* FROM sqlite\_temp\_master)

WHERE type='table'

ORDER BY name

(10) SQLite数据库是否有已知的大小限制？

数据库大小被限制在 2TB(241 bytes). 这是理论限制。事实上，你应该把 SQLite数据库的大小限制在100GB以下，以免出现运行性能上的问题。如果你需要储存100GB或更多数据在一个数据库中， 考虑使用为此而设计的企业版数据库吧。

一个数据库的理论行数限制是 264-1，显然你会在达到行数限制之前先超过文件大小的限制。目前一行可以存放 230 bytes 数据。而基本的文件格式可以支持行大小到约 262 bytes.

可能还会有对于表、索引的数目或表和索引中的字段数的限制，但没人知道是多少。事实上，每当新数据库打开时，SQLite需要读取和 分析所有表和索引声明的初始SQL，所以，为了调用 sqlite3\_open() 时获得最佳性能，最好减少声明的表的数目。同样的，即使 对于表中字段数没有限制，多于100个也显得太多了。 只有表开头的31个字段会得到优化。你可以在一个索引中放入任意多的字段但超过30字段的索引将不用于优化查询。

表，索引，视图，触发器和字段名称可以任意长，但SQL 函数名 (由 sqlite3\_create\_function() API创建的)不得超过255个字符。

(11) 在 SQLite 中 VARCHAR 的最大长度是多少？

SQLite不强制VARCHAR的长度。你可以声明一个VARCHAR(10)，SQLite一样可以让你存放500个字符在里面。 并且它们会始终完整无缺——决不会被截断。

(12) SQLite 是否支持 BLOB 类型？

SQLite 3.0 版支持在任何字段存放 BLOB 数据，不管字段声明为什么类型。

(13) 如何从一个已存在的 SQLite 数据表中添加/删除字段？

SQLite有有限的ALTER TABLE支持，可以用于添加字段到表的末尾 或更改表名。如果你要对表的结构作更复杂的修改，你需要重新创建表。你可以在一个临时表中备份数据，撤销旧表，重建新表后再恢复数据。

例如，假设你有一个名为 "t1" 的表，有名为 "a", "b", 和 "c" 三个字段，你要删除字段 "c" 。可按如下步骤操作：

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;

(14) 我删除了很多数据但是数据库文件并没有减小，是不是 Bug？

不是的。当你从 SQLite 删除数据之后，未使用的磁盘空间被添加到一个内在的“空闲列表”中用于存储你下次插入的数据。磁盘空间并没有丢失，但是也不向操作系统返回磁盘空间。

如果你删除了大量的数据且想要减小数据库文件，执行 VACUUM命令。VACUUM 命令会清空“空闲列表”，把数据库尺寸缩到最小。注意， VACUUM 会耗费一些时间（在 Linux 系统下大约0.5秒/兆）并且要使用两倍于数据库文件大小的磁盘空间。

对于SQLite version 3.1, 替代VACUUM命令的一个方法是auto-vacuum模式，用 auto\_vacuum pragma语法开启该模式。

(15) 是否能将 SQLite 用于商业用途而不用交版权费用？

可以。SQLite 是公开的。代码的任何部分都没有声明所有权。你可以用它来做你想要的任何事情。

(16) 如何插入有单引号（'）的字符串？

使用双单引号即可，例如：

INSERT INTO xyz valueS('5 O''clock');

插入数据库的是：5 0'clock。

(17) SQLITE\_SCHEMA 错误代表什么？

在 SQLite 版本3中，当一个预处理 SQL 语句不合法不能执行时就会返回一个 SQLITE\_SCHEMA 错误。当这个错误发生时，该语句应当用 sqlite3\_prepare() API函数重新编译。在 SQLite 版本3中，只有使用 sqlite3\_prepare ()/sqlite3\_step()/sqlite3\_finalize() API函数执行 SQL 才会发生这个错误，而使用 sqlite3\_exec(). 则不会。这与版本2不同。

大部分发生这个错误的原因是当 SQL 预处理完时数据库已经改变了（可能是被另一个进程改变的）。还可能有如下原因：

\* 对一个数据库进行DETACH操作

\* 对一个数据库进行VACUUM操作

\* 一个用户函数定义被删除或改变了。

\* 一个排序定义被删除或改变了。

\* 一个授权函数改变了。

解决的办法是重新编译并再次尝试执行。所有涉及 sqlite3\_prepare()/sqlite3\_step()/sqlite3\_finalize() API 函数的都应当重新编译。参见下例：

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 );

(18) 为什么ROUND(9.95,1) 返回 9.9 而不是 10.0? 难道9.95 不该向上进位么？

SQLite 内部使用二进制运算，9.95用 64-bit IEEE 浮点数 ( SQLite 内部使用的) 表示为 9.949999999999999289457264239899814128875732421875。所以当你输入 "9.95"时， SQLite 就理解为上述的数字，进而四舍五入得到9.9。这个问题在处理浮点二进制数总会产生。通常的规则是十进制的有限浮点数通常无法表示为二进制有限浮点数，只能由最接近的二进制数来代替。这个近似数会非常接近原数，但总一些细微的不同，所以可能无法得到你预期的结果。