SQLite这么娇小可爱，不多了解点都不行啊

22 AUGUST 2015 on SQLite, WAL

在我眼里，MySQL和Oracle是这样的

而SQLite在是这样的

所以这么萌的数据库，我真的应该多了解她的。

简介

SQLite，是一款轻型的数据库，是遵守ACID的关系型数据库管理系统。它的设计目标是嵌入式的，目前Android和iOS的设备内置的都是SQLite数据库。SQLite虽然娇小，但也支持事务和多数的SQL92标准。

主要特点

- Zero-Configuration 无需安装和管理配置。
- Serverless 无需服务器支持。
- Single Database File 数据文件存储在一个单一的磁盘文件。
- Stable Cross-Platform Database File 数据库文件格式跨平台，无论是大小端，或者是32bit或64bit机器都没有关系
- Compact 完整特性的SQLite编译出来在500KiB左右，裁剪特性甚至可以得到低于300KiB的库（当前版本3.8.11.1）。
- Manifest typing 可以声明数据库字段类型，但是字段存储的类型实际的存储类型和实际值相关，单独的一个字段可能包含不同存储类的值。
- Variable-length records 可变长度记录，例如你存储一个字符到 VARCHAR(100) 的列，实际需要的存储空间一个字符加一个字节的存储空间。

- SQL statements compile into virtual machine code SQL语句会被编译成虚拟机代码，这种虚拟机代码直白可读，便于调试。
- Public domain 完全开源。
- SQL language extensions

主要缺点

- SQLite 只提供数据库级的锁定，所以不支持高并发。
- 不支持存储过程。
- SQLite 没有用户帐户概念，而是根据文件系统确定所有数据库的权限。这会使强制执行存储配额发生困难,强制执行用户许可变得不可能。

如果只在移动设备使用SQLite，那么他的优点足够好，并且缺点不明显，所以大叔MySQL走开。SQLite妹妹快过来 ╭(╯3╰)╮ 。

事务与锁（**< 3.7.0**）

SQLite的事务和锁是很重要的概念。

锁

SQLite有5个不同的锁状态

1. UNLOCKED（未加锁）
2. SHARED（共享）
3. RESERVED（保留）
4. PENDING（未决）
5. EXCLUSIVE（排它）

SQLite有一个加锁表，记录数据库连接的锁状态。每个数据库连接在同一时刻只能处于其中一个锁状态。每种状态(UNLOCKED)都有一种锁与之对应。

读

数据库连接最初处于UNLOCKED状态，在此状态下，连接还没有存取数据库。当连接到了一个数据库，甚至已经用BEGIN开始了一个事务时，连接都还处于UNLOCKED状态。为了能够从数据库中读取数据，连接必须必须进入SHARED状态，也就是说首先要获得一个SHARED锁。多个连接可以同时获得并保持共享锁，也就是说多个连接可以同时从同一个数据库中读数据，`SQLite是支持并发读取数据的`。

写

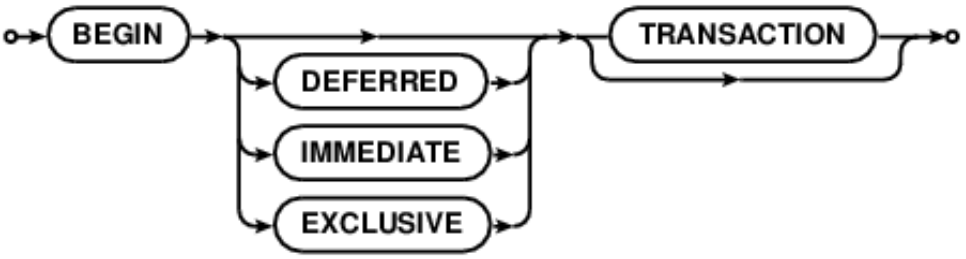
一个连接想要写数据库，它必须首先获得一个RESERVED锁。一个数据库上同时只能有一个RESERVED锁，保留锁可以与共享锁共存，RESERVED锁即不阻止其它拥有SHARED锁的连接继续读数据库，也不阻止其它连接获得新的SHARED锁。 一旦一个连接获得了RESERVED锁，它就可以将数据写入缓冲区，而不是实际地写到磁盘。 当连接想要提交修改(或事务)时，需要获得PENDING锁，之后连接就不能再获得新的SHARED锁了，但已经拥有SHARED锁的连接仍然可以继续正常读数据库。当所有其它SHARED锁都被释放时，拥有PENDING锁的连接就可以将其锁提升至EXCLUSIVE锁，此时就可以将以前对缓冲区所做的修改写到数据库文件。所以`SQLite是不支持并发写的`。

事务

SQLite有三种不同的事务

1. DEFERRED（推迟）
2. MMEDIATE（立即）
3. EXCLUSIVE（排它）

事务类型在BEGIN命令中指定：



DEFERRED

一个DEFERRED事务不获取任何锁(直到它需要锁的时候)，BEGIN语句本身也不会做什么事情——它开始于UNLOCK状态。默认情况下就是这样的，如果仅仅用BEGIN开始一个事务，那么事务就是DEFERRED的，同时它不会获取任何锁；当对数据库进行第一次读操作时，它会获取SHARED锁；同样，当进行第一次写操作时，它会获取RESERVED锁。

MMEDIATE

由BEGIN开始的IMMEDIATE事务会尝试获取RESERVED锁。如果成功，BEGIN IMMEDIATE保证没有别的连接可以写数据库。但是，别的连接可以对数据库进行读操作；但是，RESERVED锁会阻止其它连接的BEGIN IMMEDIATE或者BEGIN EXCLUSIVE命令，当其它连接执行上述命令时，会返回SQLITE_BUSY错误。这时你就可以对数据库进行修改操作了，但是你还不能提交，当你COMMIT时，会返回SQLITE_BUSY错误，这意味着还有其它的读事务没有完成，得等它们执行完后才能提交事务。

EXCLUSIVE

EXCLUSIVE事务会试着获取对数据库的EXCLUSIVE锁。这与IMMEDIATE类似，但是一旦成功，EXCLUSIVE事务保证没有其它的连接，所以就可对数据库进行读写操作了。

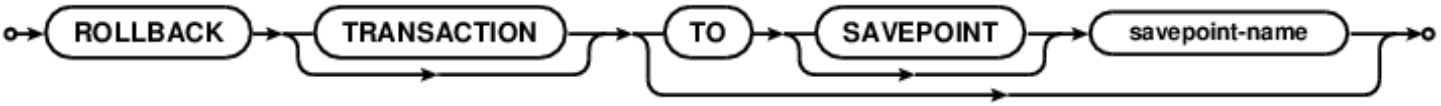
死锁

如果两个以BEGIN DEFERRED开始事务的连接都处于SHARED状态，并且都在等待对方结束SHARED从而进入RESERVED的话，就会进入死锁状态。所以`BEGIN DEFERRED开始的事务是有可能产生死锁的`.

Write-Ahead Logging (>=3.7.0)

SQLite 3.7.0之前是不支持写的时候读得。为了能够读得时候写，引入了Write-Ahead Logging（WAL）机制，这样可以支持一个写和多个读并发。

在引入WAL机制之前，SQLite使用rollback journal机制实现原子事务。



rollback journal机制的原理是：在修改数据库文件中的数据之前，先将修改所在分页中的数据备份在另外一个地方，然后才将修改写入到数据库文件中；如果事务失败，则将备份数据拷贝回来，撤销修改；如果事务成功，则删除备份数据，提交修改。

WAL机制的原理是：修改并不直接写入到数据库文件中，而是写入到另外一个称为WAL的文件中；如果事务失败，WAL中的记录会被忽略，撤销修改；如果事务成功，它将在随后的某个时间被写回到数据库文件中，提交修改。

同步WAL文件和数据库文件的行为被称为checkpoint（检查点），它由SQLite自动执行，默认是在WAL文件积累到1000页修改的时候；当然，在适当的时候，也可以手动执行checkpoint，SQLite提供了相关的接口。执行checkpoint之后，WAL文件会被清空。

在读的时候，SQLite将在WAL文件中搜索，找到最后一个写入点，记住它，并忽略在此之后的写入点（这保证了读写和读读可以并行执行）；随后，它确定所要读的数据所在页是否在WAL文件中，如果在，则读WAL文件中的数据，如果不在，则直接读数据库文件中的数据。

在写的时候，SQLite将之写入到WAL文件中即可，但是必须保证独占写入，因此写写之间不能并行执行。

WAL在实现的过程中，使用了共享内存技术，因此，所有的读写进程必须在同一个机器上，否则，无法保证数据一致性。

优点

1. 读和写可以完全地并发执行，不会互相阻塞（但是写之间仍然不能并发）。
2. WAL在大多数情况下，拥有更好的性能（因为无需每次写入时都要写两个文件）。
3. 磁盘I/O行为更容易被预测

缺点

1. 访问数据库的所有程序必须在同一主机上，且支持共享内存技术。

2. 每个数据库现在对应3个文件：.db，-wal，-shm。

3. 当写入数据达到GB级的时候，数据库性能将下降。

4. 3.7.0之前的SQLite无法识别启用了WAL机制的数据库文件。

5. WAL引入的兼容性问题。在启用了WAL之后，数据库文件格式的版本号由1升级到了2，因此，3.7.0之前的SQLite无法识别启用了WAL机制的数据库文件。禁用WAL会使数据库文件格式的版本号恢复到1，从而可以被SQLite 3.7.0之前的版本识别。

6. WAL引入的性能问题。在一般情况下，WAL会提高SQLite的事务性能；但是在某些极端情况下，却会导致SQLite事务性能的下降。

    1. 在事务执行时间较长或者要修改的数据量达到GB级的时候，WAL文件会被占用，它会暂时阻止checkpoint的执行（checkpoint会清空WAL文件），这将导致WAL文件变得很大，增加寻址时间，最终导致读写性能的下降。

2. 当checkpoint执行的时候，会降低当时的读写性能，因此，WAL可能会导致周期性的性能下降

END