[PRAGMA command](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/pragma.html#syntax)   是一个特殊的命令。 它用于修改 SQLite 库操作或查询库以取得内部（非表）数据。 PRAGMA 命令可以使用与其它 SQLite 命令（如 SELECT、INSERT） 相同的接口，只在有如下几个重要方面有所不同：

pragma 接受一个整数值  ***value***  或一个名字符号。 字符串 "**on** ", "**true** " 以及 "**yes** " 与  **1**   是等价的。字符串 "**off** ", "**false** ", 和 "**no** " 则等价于  **0** 。 这些字符串是不区分大小写的，也不需要双引号。 不能识别的字符串将被认为是  **1** ， 而不会提示错误。 当返回  *value*   时，它被看作是一个整数。

 **PRAGMA auto\_vacuum;  
PRAGMA auto\_vacuum =***0 | none | 1 | full | 2 | incremental* **;**

在数据库中查询或设置 auto-vacuum 。

通常（也就是说在 auto\_vacuum 是 0 或 none的时候）， 当一个从数据库中删除数据的事务提交时，数据库文件大小不会改变， 未被使用的数据库文件页空间被记入一个“空闲列表”中， 这样，这些空间在以后的 INSERT 中就可以重用。所以， 数据库文件不会缩小。 在这种模式下，可以使用[VACUUM](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/lang_vacuum.html)   命令来回收未用的空间。

如果 auto-vacuum 标志是 1 (full，完全模式)，空闲列表中的页将会移到文件的尾部， 那么，当每次提交事务时，就可以通过将文件截断来翻译空闲列表中的页。 但请注意， auto-vacuum 只会从数据库文件中载断空闲列表中的页， 而不会回收数据库中的碎片，也不会像[VACUUM](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/lang_vacuum.html)   命令那样重新整理数据库内容。实际上，由于需要在数据库文件中移动页， auto-vacuum 会产生更多的碎片。

要使用 auto-vacuum，需要一些前题条件。 数据库中需要存储一些额外的信息以记录它所跟踪的每个数据库页都找回其指针位置。 所以，auto-vacumm 必须在建表之前就开启。在一个表创建之后， 就不能再开启或关闭 auto-vacumm。

如果 auto-vacuum 的值是 2 (incremental，增量模式) ，那么，自动 vacuum 需要使用存放在数据库文件中的另外的信息。但它不会像 auto\_vacuum == full 那样在每次事务提交的时候发生。在增量模式下，必须使用独立的 [incremental\_vacuum](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/pragma.html#pragma_incremental_vacuum)   pragma 来引发 vacuum。

数据库连接可以自由地在完全和增量模式间切换。但是， 如果数据库中已经建了一个表，连接就不能切换到 none 模式， 也不能从 none 模式切换成别的模式。

 **PRAGMA cache\_size;    
PRAGMA cache\_size =***Number-of-pages* **;**

查询可修改 SQLite 一次可以在内存中保持的数据库磁盘页数量的最大值。 每一页大约需要 1.5K 的内存空间。缺省值是2000。 如果你需要做大的 UPDATE 或 DELETE 操作，每次会影响相当多的数据库行， 并且你也不在乎 SQLite 占用更多的内存，那么，你就可以增加缓存的大小， 这样，以获得可能的速度的提升。

当你使用 cache\_size pragma 修改缓存大小以后， 改变只会对当前的会话起作用。数据库连接关闭并重新连接后， 它又会变成缺省的大小。如果要永久修改缓存值，则需要使用[**default\_cache\_size**](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/pragma.html#pragma_default_cache_size)

 **PRAGMA case\_sensitive\_like;    
PRAGMA case\_sensitive\_like =***0 | 1* **;**

对于 latin1 字符，LIKE 操作符缺省的行为会忽略大小写。 也就是说，默认情况下，  **'a' LIKE 'A'**   的结果是true。 可以通过开启该 pragma 来改变这种行为。当启用 case\_sensitive\_like 后，**'a' LIKE 'A'**   会是 false 但  **'a' LIKE 'a'**   仍然是 true.

 **PRAGMA count\_changes;    
PRAGMA count\_changes =***0 | 1* **;**

查询或修改 count-changes 标志。通常， 没有设置 count-changes ，则 INSERT, UPDATE 和 DELETE 语句不会返回任何数据。 当设置该值时，这些命令都会返回由一个整数组成的一行数据， 该整数表示被该命令所插入，修改或删除的行数， 但不包括任何由触发器插入、修改或删除的行。

 **PRAGMA default\_cache\_size;    
PRAGMA default\_cache\_size =***Number-of-pages* **;**

查询或设置 SQLite 可在内存中同时保持的数据库磁盘文件页， 每占用 1K 磁盘空间的页需要大约 1.5K 的内存。 该 pragma 类似[**cache\_size**](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/pragma.html#pragma_cache_size)  。 只是，它会永久改变缓存的值。通过该 pragma， 可以一次设置缓存值，以后每次后新打开数据库时都将使用该值。

 **PRAGMA default\_synchronous;**

该 pragma 在 2.8 版本中可用，但在 3.0 中删除了。 使用它非常危险，因此也不推荐使用。 为帮助2.8版本的用户不再使用该 pragma， 本文档不会告诉你它怎么用。

 **PRAGMA empty\_result\_callbacks;    
PRAGMA empty\_result\_callbacks =***0 | 1* **;**

查询或设置 empty-result-callbacks 标志。

该标志只会影响 sqlite3\_exec API。 通常，当清除该标志时，如果命令返回 0 行数据， 则不会引发用于 sqlite3\_exec()上的回调函数。 但若在这种情况下设置了 empty\_result\_callbacks， 则回调函数会将第三个参数置为 0 （NULL） 并进行调用，且只会调用一次。 这可以使用调用 sqlite3\_exec() API 的程序即使在没有返回数据的情况下也可以获得列名。

 **PRAGMA encoding;    
PRAGMA encoding = "UTF-8";    
PRAGMA encoding = "UTF-16";    
PRAGMA encoding = "UTF-16le";    
PRAGMA encoding = "UTF-16be";**

第一种格式，主数据库已创建。那么该 pragma 会返回主数据库所使用的文件编码。 可能是： "UTF-8", "UTF-16le" (小印第安 UTF-16 编码) 或 "UTF-16be" (大印第安 UTF-16 编码)。 如果主数据库没有创建， 那么返回值将是将要用于创建主数据库的文本编码（如果在当前会话中创建的话）。

第二种以及以后的格式只有在主数据库未创建时有用。 在这种情况下，该 pragma 将会把数据库编码设为在当前会话中将要创建的数据库所使用的编码。 字符串“UTF-16”会被解释为“本机字节顺序的UTF-16编码”。 若第二种及以后的格式用于已创建的数据库文件上， 则它们会被默默地忽略而不起任何作用。

一旦为数据库设置了字符编码，就不能再更改。

使用 ATTACH 命令创建的数据库记录使用与主数据相同的编码。

 **PRAGMA full\_column\_names;    
PRAGMA full\_column\_names =***0 | 1* **;**

查询或修改 full-column-names 标志。 该标志会影响当 SELECT 查询后面的列表是一个 “表-列名” 或 “\*”时 SQLite返回列名的方式。通常， 如果 SELECT 语句连接两个或多个表， 结果列名将是 <表名/别名><列名>； 而若仅仅对单个表查询时，将是 <column-name>。 但如果设置了 full-column-names 标志，列名将永远是 <table-name/alias> <column-name> ， 而不管是只查询一个表或同时连接多个表。

如果 short-column-names 和 full-column-names 都设置了， 则会以 full-column-names 为准。

 **PRAGMA fullfsync    
PRAGMA fullfsync =***0 | 1* **;**

查询修改 fullfsync 标志。 该标志影响在支持 F\_FULLFSYNC 同步方法的系统上，是否使用该方法。 默认值是 off (不使用)。 在写本文的时候 (2006-02-10)，只有 Mac OS X 支持 F\_FULLFSYNC。

 **PRAGMA incremental\_vacuum** *(N)* **;**

Incremental\_vacuum pragma 会导致从空闲列表中最多移除  *N*   页。 数据库文件也会按该数量截断。如果数据库不在  [auto\_vacuum==incremental](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/pragma.html#pragma_auto_vacuum)   模式， 或空闲列表中没有页，则该 pragma 不起作用。 如果空闲列表中不到*N* 页，那么，整个空闲列表会被清空。

对于 3.4.0版来说 (第一个支持 incremental\_vacuum 的版本) ， 该我仍然是试验性的。未来可能会增强该功能，包括像  [VACUUM](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/lang_vacuum.html)   命令那样整理碎片以及节点重整等。 并且，增量 vacuum 可能会从 pragma 变为一个单独的 SQL 命令， 也许会是 VACUUM 命令的变体。程序员们要注意不要迷恋于当前的语法， 因为以后有可能会改变。

 **PRAGMA legacy\_file\_format;    
PRAGMA legacy\_file\_format =  *ON | OFF***

本 pragma 设置或查询 legacy\_file\_format 标志的值。 当它为 on 时，新创建的数据库文件格式可以被3.0.0以后的版本读写。 如果它为 off，则会使用最新的数据库文件格式，旧版本的SQLite将无法读写。

当不带参数使用该 pragma 时，它返回该标志当前的值。 它  不 会   告诉你当前数据库所使用的文件格式， 而只会表明在以后创建新数据库时将使用何种格式。

该标志只对新创建数据库有效。对已存在的数据库不起作用。

 **PRAGMA locking\_mode;    
PRAGMA locking\_mode =  *NORMAL | EXCLUSIVE***

该 pragma 设置或查询数据库连接的锁模式。 锁模式可能是 NORMAL 或 EXCLUSIVE.

在 NORMAL 模式下， (默认值), 一个数据库连接会在每次完成读或写时释放数据库文件锁。 当锁模式设为 EXCLUSIVE 时，数据库连接永远不会释放文件锁。 在该模式下，当第一次读数据库文件时，会获得并持有一个共享锁。 当第一次向数据库写时，将获得并持有一个排它锁。

在 EXCLUSIVE 模式下获得的锁可以通过关闭数据库连接来释放， 也可以通过使用该 pragma 将锁模式改为 NORMAL，并且再次访问（读或写） 数据库来释放。仅仅将锁模式置为 NORMAL 是不够的， 直到下一次访问数据库文件时才会释放已持有的锁。

有两种原因要求设置 EXCLUSIVE 锁模式。 一是一个应用程序确实不希望其它进程访问数据库文件。 二是在这种模式下可以使用优化器，它可以节省少量的磁盘文件操作。 这在嵌入式系统中可能非常重要。

当 locking\_mode pragma 指定一个特定的数据库时，如：

PRAGMA  **main.** locking\_mode=EXCLUSIVE;

那么，锁模式只会对该数据库有效。如果不指定数据库名称， 那么锁模式会应用于所有数据库，包括以后使用 [ATTACH](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/lang_attach.html) 命 令连接的数据库。

临时（ temp） 数据库 (用于存放临时表和索引)永远使用 EXCLUSIVE 锁模式。临时数据库的锁模式不能改变。所有其它的数据库则默认使用 NORMAL 锁模式，并可以通过使用 pragma 改变。

 **PRAGMA page\_size;    
PRAGMA page\_size =***bytes* **;**

查询或设置数据库的 page-size 。只能在数据库创建之间设置 page-size。 页的大小必须是 2 的幂， 且幂指数只能在 512 和 8192 之间。 其上限可以在编译时通过修改源代码中的 SQLITE\_MAX\_PAGE\_SIZE 修改， 但最大不能超过32768。

 **PRAGMA max\_page\_count;    
PRAGMA max\_page\_count =***N* **;**

查询或设置数据库文件最大的页数。 两种格式都会返回最大的页数。第二种格式试图修改最大页数。 最大页数不能修改为小于当前数据库的大小。

 **PRAGMA read\_uncommitted;    
PRAGMA read\_uncommitted =***0 | 1* **;**

查询，设置或清除 READ UNCOMMITTED （读未提义）隔离级别。 SQLite 默认的隔离级别是 SERIALIZABLE (可串行化）。 任何进程或线程都可以选择 READ UNCOMMITTED 级别。但除了在数据库连接间共享一个通过页和模式缓存外，仍然会使用 SERIALIZABLE 隔离级别。 共享缓存可以使用  [sqlite3\_enable\_shared\_cache()](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/c3ref/enable_shared_cache.html)  API 开启， 并且只能在执行同一线程的连接间开启。共享缓存默认是关闭的。

 **PRAGMA short\_column\_names;    
PRAGMA short\_column\_names =***0 | 1* **;**

查询或修改 short-column-names 标志。 该标志会影响当 SELECT 查询后面的列表是一个 “表-列名” 或 “\*”时 SQLite返回列名的方式。 通常， 如果 SELECT 语句连接两个或多个表， 结果列名将是 <表名/别名><列名>； 而若仅仅对单个表查询时，将是 <column-name>。 但如果设置了 short-column-names 标志，列名将永远是 <column-name> ， 而不管是只查询一个表或同时连接多个表。

如果 short-column-names 和 full-column-names 都设置了， 则以 full-column-names 标志为准。

 **PRAGMA synchronous;    
PRAGMA synchronous = FULL;**(2)**PRAGMA synchronous = NORMAL;**(1)**PRAGMA synchronous = OFF;**(0)

查询或设置 synchronous 标志。 第一种格式返回一个整数，当 synchronous 为 FULL (2) 时， SQLite 数据库引擎将会在重要的时刻暂停， 以保证在继续运行之前数据确实已经写到磁盘上去了。 这能保证在遇到操作系统崩溃可电源故障时， 重新启动机器数据库文件不会被破坏。FULL 同步方式是非常安全的， 但也是很慢的。若 synchronous 设为 NORMAL，则 SQLite 数据库引擎将在大多数重要时刻暂停，但比在 FULL 模式要少。 在极少情况下，处理这种模式的数据库文件可能会由于“不是时候” 的电源故障而导致受损。但在实际应用中， 更有可有遭受到灾难性的磁盘故障或其它无法恢复的硬件故障。 若将 synchronous 设为 OFF (0)，那么 SQLite 从操作系统取得数据后将会立即进行处理而不会暂停。 如果使用 SQLite 的应用程序崩溃，数据将是安全的。 但如果在数据写入磁盘之前，操作系统死机或机器断电， 数据库文件可能会损坏。此外，在该模式下， 某些操作会比其它情况下快 50 倍。

在 SQLite 版本 2 中，默认值是 NORMAL。 对于版本 3， 默认值是 FULL。

 **PRAGMA temp\_store;    
PRAGMA temp\_store = DEFAULT;**   (0)**PRAGMA temp\_store = FILE;**   (1)**PRAGMA temp\_store = MEMORY;**   (2)

查询或改变  **temp\_store**   参数的设置。 当 temp\_sore 为 DEFAULT (0) 时，将根据编译时的 C 语言预处理宏决定如何存储临时表的索引。 若 temp\_store 为 MEMORY (2) ，临时表和索引会存储在内存中。 当 temp\_store 为 FILE (1) 时，临时表和索引将存储在文件中。 可以使用  [temp\_store\_directory](http://www.dujinfang.com/sqlite_docs_3_5_4_zh_CN/pragma.html#pragma_temp_store_directory)   pragma 来指定保存文件的路径。 指定  **FILE**   。当 temp\_store 设置改变时， 所有已存在的表、索引、触发器以及视图都将被立即删除。

编译时的 C 语言预处理宏 TEMP\_STORE 可以覆盖 该 pragma 的设置。 下表列出它们之间的关系：

|  |  |  |
| --- | --- | --- |
| **TEMP\_STORE** | **PRAGMA temp\_store** | **Storage used for TEMP tables and indices** |
| 0 | *any* | file |
| 1 | 0 | file |
| 1 | 1 | file |
| 1 | 2 | memory |
| 2 | 0 | memory |
| 2 | 1 | file |
| 2 | 2 | memory |
| 3 | *any* | memory |

 **PRAGMA temp\_store\_directory;    
PRAGMA temp\_store\_directory = 'directory-name';**

查询或改变 temp\_store\_directory 的值。 如果将临时表存放于文件中，它用于指定文件的存放路径。 该设置仅在当前的数据库连接中有效，当数据库重新在新的连接中打开时， 将回到其默认值。

当改变 temp\_store\_directory 设置时，所有已存在的临时表、索引、 触发器以及视图都将立即删除。在实际应用中，打开数据库后应该立即设置该值。

目录名  *directory-name*   应该用单引号引起来。 如果想回到默认值，将  *directory-name*   设为空字符串，如 *PRAGMA temp\_store\_directory = ''* 。 如果路径找不到或目录  *directory-name*   无法写入， 会提示错误。

默认的临时文件存放的目录依赖于操作系统。 对于 Unix/Linux/OSX 来说，默认可路径是**/var/tmp, /usr/tmp, /tmp,**   and  **以及当前目录  *current-directory***中 第一个可写的目录。 对于 Windows NT，默认路径由 Windows决定，通常是  **C:/Documents and Settings/*user-name* /Local Settings/Temp/** 。 SQLite 创建的临时文件在打开后会被立即删除（unlink), 这样，当 SQLite 进程退出时，操作系统就可以自动删除这些文件。 所以，正常状态下，使用*ls*   或  *dir*   命令是无法看到这些临时文件的。