由于FMDB是建立在SQLite的之上的，所以你至少也该把这篇文章从头到尾读一遍。与此同时，把SQLite的文档页 加到你的书签中。

FMDB支持ARC和MRC，FMDB会在编译的时候知道你是用的哪一种，然后进行相应处理。

**使用方法**

**FMDB有三个主要的类**

1.FMDatabase – 表示一个单独的SQLite数据库。 用来执行SQLite的命令。

2.FMResultSet – 表示FMDatabase执行查询后结果集

3.FMDatabaseQueue – 如果你想在多线程中执行多个查询或更新，你应该使用该类。这是线程安全的。

**数据库创建**

创建FMDatabase对象时参数为SQLite数据库文件路径。该路径可以是以下三种之一：

1..文件路径。该文件路径无需真实存，如果不存在会自动创建。

2..空字符串(@”")。表示会在临时目录创建一个空的数据库，当FMDatabase 链接关闭时，文件也被删除。

3.NULL. 将创建一个内在数据库。同样的，当FMDatabase连接关闭时，数据会被销毁。

(如需对临时数据库或内在数据库进行一步了解，请继续阅读：http://www.sqlite.org/inmemorydb.html)

1. FMDatabase \*db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];

**打开数据库**

在和数据库交互 之前，数据库必须是打开的。如果资源或权限不足无法打开或创建数据库，都会导致打开失败。

1. **if** (![db open]) {
2. [db release];
3. **return**;
4. }

**执行更新**

一切不是SELECT命令的命令都视为更新。这包括  CREATE, UPDATE, INSERT,ALTER,COMMIT, BEGIN, DETACH, DELETE, DROP, END, EXPLAIN, VACUUM, and REPLACE  （等）。

简单来说，只要不是以SELECT开头的命令都是UPDATE命令。

执行更新返回一个BOOL值。YES表示执行成功，否则表示有那些错误 。你可以调用 -lastErrorMessage 和 -lastErrorCode方法来得到更多信息。

**执行查询**

SELECT命令就是查询，执行查询的方法是以 -excuteQuery开头的。

执行查询时，如果成功返回FMResultSet对象， 错误返回nil. 与执行更新相当，支持使用 NSError\*\*参数。同时，你也可以使用 -lastErrorCode和-lastErrorMessage获知错误信息。

为了遍历查询结果，你可以使用while循环。你还需要知道怎么跳到下一个记录。使用FMDB，很简单实现，就像这样：

1. FMResultSet \*s = [db executeQuery:@"SELECT \* FROM myTable"];
2. **while** ([s next]) {
3. //retrieve values for each record
4. }

你必须一直调用   -[FMResultSet next]   在你访问查询返回值之前，甚至你只想要一个记录：

1. FMResultSet \*s = [db executeQuery:@"SELECT COUNT(\*) FROM myTable"];
2. **if** ([s next]) {
3. **int** totalCount = [s intForColumnIndex:0];
4. }

FMResultSet  提供了很多方法来获得所需的格式的值：

    intForColumn:

    longForColumn:

    longLongIntForColumn:

    boolForColumn:

    doubleForColumn:

    stringForColumn:

    dataForColumn:

    dataNoCopyForColumn:

    UTF8StringForColumnIndex:

    objectForColumn:

这些方法也都包括 {type}ForColumnIndex 的这样子的方法，参数是查询结果集的列的索引位置。

你无需调用  [FMResultSet close]来关闭结果集, 当新的结果集产生，或者其数据库关闭时，会自动关闭。

**关闭数据库**

当使用完数据库，你应该 -close 来关闭数据库连接来释放SQLite使用的资源。

    [db close];

**事务**

**FMDatabase是支持事务的。**

**数据净化（数据格式化）**

使用FMDB，插入数据前，你不要花时间审查你的数据。你可以使用标准的SQLite数据绑定语法。

1. INSERT INTO myTable VALUES (?, ?, ?)

SQLite会识别 “?” 为一个输入的点位符， 这样的执行会接受一个可变参数（或者表示为其他参数，如NSArray, NSDictionary,或va\_list等），会正确为您转义。

你也可以选择使用命名参数语法。

1. INSERT INTO myTable VALUES (:id, :name, :value)

参数名必须以冒名开头。SQLite本身支持其他字符，当Dictionary key的内部实现是冒号开头。注意你的NSDictionary key不要包含冒号。

1. NSDictionary \*argsDict = [NSDictionary dictionaryWithObjectsAndKeys:@"My Name", @"name", nil];
2. [db executeUpdate:@"INSERT INTO myTable (name) VALUES (:name)" withParameterDictionary:argsDict];

而且，代码不能这么写（为什么？想想吧。）

1. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", @"this has \" lots of ' bizarre \" quotes '"];

你应该：

1. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", @"this has " lots of ' bizarre " quotes '"];

提供给 -executeUpdate: 方法的参数都必须是对象。就像以下的代码就无法工作，且会产生崩溃。

1. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", 42];

 正确有做法是把数字打包成 NSNumber对象

1. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber numberWithInt:42]];

或者，你可以使用  -execute\*WithFormat: ，这是NSString风格的参数

1. [db executeUpdateWithFormat:@"INSERT INTO myTable VALUES (%d)", 42];

 -execute\*WithFormat:  的方法的内部实现会帮你封装数据， 以下这些修饰符都可以使用： %@, %c, %s, %d, %D,%i, %u, %U, %hi, %hu, %qi, %qu, %f, %g, %ld, %lu, %lld, and %llu.  除此之外的修饰符可能导致无法预知的结果。 一些情况下，你需要在SQL语句中使用 % 字符，你应该使用 %%。

**使用FMDatabaseQueue 及线程安全**

在多个线程中同时使用一个FMDatabase实例是不明智的。现在你可以为每个线程创建一个FMDatabase对象。 不要让多个线程分享同一个实例，它无法在多个线程中同时使用。 若此，坏事会经常发生，程序会时不时崩溃，或者报告异常，或者陨石会从天空中掉下来砸到你Mac Pro.  总之很崩溃。所以，不要初始化FMDatabase对象，然后在多个线程中使用。请使用 FMDatabaseQueue，它是你的朋友而且会帮助你。以下是使用方法：

首先创建队列。

1. FMDatabaseQueue \*queue = [FMDatabaseQueue databaseQueueWithPath:aPath];

这样使用。

1. [queue inDatabase:^(FMDatabase \*db) {
2. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber numberWithInt:1]];
3. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber numberWithInt:2]];
4. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber numberWithInt:3]];
5. FMResultSet \*rs = [db executeQuery:@"select \* from foo"];
6. **while**([rs next]) {
7. …
8. }
9. }];

像这样，轻松地把简单任务包装到事务里：

1. [queue inTransaction:^(FMDatabase \*db, BOOL \*rollback) {
2. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber numberWithInt:1]];
3. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber numberWithInt:2]];
4. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber numberWithInt:3]];
5. **if** (whoopsSomethingWrongHappened) {
6. \*rollback = YES; **return**;
7. }
8. // etc…
9. [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber numberWithInt:4]];
10. }];

 FMDatabaseQueue  后台会建立系列化的G-C-D队列，并执行你传给G-C-D队列的块。这意味着 你从多线程同时调用调用方法，GDC也会按它接收的块的顺序来执行。谁也不会吵到谁的脚 ，每个人都幸福。