FMDB

> *如果直接使用sqlite3操作数据库（c/c++ API），操作麻烦，使用难度大，工作效率低。所以，在Objective-c中使用比较麻烦，下面是一个简单的基本查询：*

```
sqlite3 *oldDatabase = nil;
NSString *filePath = @"/File/Path/For/Sqlite/Database";
sqlite3_open([filePath UTF8String], &oldDatabase);
sqlite3_stmt* statement = nil;
const char* sqlQuery = "SELECT NAME FROM TABLENAME";
if (sqlite3_prepare_v2(oldDatabase, sqlQuery, -1, &statement, NULL) ==
SQLITE_OK)
{
    while (sqlite3_step(statement) == SQLITE_ROW) // get row one by one
    {
        const unsigned char* zdatabaseName = sqlite3_column_text(statement,
0);
        NSLog(@"%s %s", __FUNCTION__, zdatabaseName);
    }
}
sqlite3_finalize(statement);
sqlite3_close(oldDatabase);
```

> *而FMDB数据库操作类对sqlite3的操作进行了便利地封装并保证了多线程下安全地操作数据库*

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执行查询后的结果集

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

FMDatabase

数据库创建

创建FMDatabase对象时参数为SQLite数据库文件路径。该文件路径无需真实存在，如果不存在会自动创建

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *docDir = [paths objectAtIndex:0];
NSString *dataBasePath = [docDir
stringByAppendingPathExtension:@"tataBase.db"];
FMDatabase *db = [FMDatabase databaseWithPath:dataBasePath];
```

打开数据库

```
if (![db open]) {
        NSLog(@"Could not open db.");
        return 0;
}
```

执行更新

- executeUpdate

一切不是SELECT命令的命令都视为更新。这包括 CREATE, UPDATE, INSERT,ALTER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，NO表示有错误。你可以调用 -lastErrorMessage 和 -lastErrorCode方法来得到更多信息。以插入操作举例：

```
[db executeUpdate:@"INSERT INTO PersonList(Name, Age, Sex, Phone, Address,
Photo) VALUES (?,?,?,?,?,?)",@"Jone", [NSNumber numberWithInt:20], [NSNumber
numberWithInt:0], @“091234567”, @“Taiwan, R.O.C”, [NSData
dataWithContentsOfFile: filepath]];
```

执行查询

- executeQuery

SELECT命令就是查询操作

执行查询时，如果成功返回**FMResultSet**对象，错误返回nil

与执行更新相同，也可以使用 -lastErrorCode和-lastErrorMessage获知错误信息

获得FMResultSet对象rs后，即使只有一条记录，一样要使用[rs next];

```
FMResultSet *rs = [db executeQuery:@"SELECT Name, Age, FROM PersonList"];
while ([rs next]) {
    NSString *name = [rs stringForColumn:@"Name"];
    int age = [rs intForColumn:@"Age"];
}
```

FMResultSet根据类型提取数据的方法:

```
- intForColumn:
- longForColumn:
- nlongLongIntForColumn:
- boolForColumn:
- doubleForColumn:
- stringForColumn:
- dateForColumn:
- dataForColumn:
- dataNoCopyForColumn:
- UTF8StringForColumnName:
- objectForColumnName:
```

以上这些方法，都有个`{type}ForColumnIndex:`版本，根据`colum`的位置来提取数据

有些时候，只是需要query某一个row里特定的一个数值（比如只是要找John的年龄），FMDB提供了几个比较简便的方法。这些方法定义在FMDatabaseAdditions.h，如果要使用，记得先import进来

```
//找地址
NSString *address = [db stringForQuery:@"SELECT Address FROM PersonList
WHERE Name = ?",@"John”];
//找年龄
int age = [db intForQuery:@"SELECT Age FROM PersonList WHERE Name =
?",@"John”];
```

使用完数据库，[FMDatabase close]，关闭数据库连接来释放SQLite使用的资源

批量操作

使用FMDatabase 的executeStatements:或者executeStatements:withResultBlock:（是否需要返回结果）

```
NSString *sql = @"create table bulktest1 (id integer primary key
autoincrement, x text);"
                  "create table bulktest2 (id integer primary key
autoincrement, y text);"
                  "create table bulktest3 (id integer primary key
autoincrement, z text);"
                  "insert into bulktest1 (x) values ('XXX');"
                  "insert into bulktest2 (y) values ('YYY');"
                  "insert into bulktest3 (z) values ('ZZZ');";

success = [db executeStatements:sql];
```

或者

```
sql = @"select count(*) as count from bulktest1;"
       "select count(*) as count from bulktest2;"
       "select count(*) as count from bulktest3;";

success = [self.db executeStatements:sql withResultBlock:^int(NSDictionary
*dictionary) {
    NSInteger count = [dictionary[@"count"] integerValue];
    XCTAssertEqual(count, 1, @"expected one record for dictionary %@",
dictionary);
    return 0;
}];
```

绑定参数

INSERT INTO myTable VALUES (?, ?, ?)

问号只是占位，执行操作可以使用NSArray, NSDictionary, or a va_list来匹配参数

你也可以选择使用命名参数语法：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（$,@），Dictionary key的内部实现是冒号开头。注意你的NSDictionary key不要包含冒号

```
NSDictionary *argsDict = @{@"name":@"Jason"};
    [db executeUpdate:@"INSERT INTO myTable VALUES (:name)"
withParameterDictionary:argsDict];
```

FMDatabaseQueue 及线程安全

不能使用同一个FMDatabase在不同线程中操作，多线程的操作是通过FMDatabaseQueue实现

首先创建队列，然后把单任务包装到事务里，串行执行

```
FMDatabaseQueue *queue = [FMDatabaseQueue databaseQueueWithPath:aPath];
[queue inDatabase:^(FMDatabase *db) {
          [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber
numberWithInt:1]];
          [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber
numberWithInt:2]];
          [db executeUpdate:@"INSERT INTO myTable VALUES (?)", [NSNumber
numberWithInt:3]];
          FMResultSet *rs = [db executeQuery:@"select * from foo"];
         while([rs next]) {
            …
         }
}];
```

事务的回滚: （当前的队列的操作的取消）

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

FMDatabaseQueue会在同一个队列里同步执行任务， GCD也会按它接收的块的顺序来执行