

iOS SQLite数据库

视频源代码交流网址

www.1000phone.net

千里寻他众百度 锋自苦寒磨砺出

Contents

1. 数据库介绍

2. SQLite和SQL语句

3. FMDB开源数据库介绍

4. 数据库、多线程和缓存

千里寻他众百度 锋自苦寒磨砺出

Contents

1. 数据库介绍

2. SQLite和SQL语句

3. FMDB开源数据库介绍

4. 数据库、多线程和缓存

千里寻他众百度 锋自苦寒磨砺出

数据库介绍

- 数据库(Database/DB)是一种某种数据模型组织起来并存放存储管理的数据仓库
- 数据库是数据管理的高级阶段，它是由文件管理系统发展起来的。
- 数据库操作对数据的增、删、改、查。由统一数据库软件来进行管理和控制。

千里寻他众百度 锋自苦寒磨砺出

常见服务器数据库

- Access 关系型数据库
- Oracle
- Microsoft SQL Server
- DB2
- Mysql

千里寻他众百度 锋自苦寒磨砺出

手机SQLite数据库

- SQLite是一款嵌入式的轻量关系型文件数据库
- 数据库所有内容存在一个文件中，一个数据库就是一个文件
- 占用资源资源很少、适合小型手机设备使用
- SQLite是几乎所有手机系统(iOS, Android, Symbian, Blackberry, Lumia, Limo等)的标准数据库
- SQLite官方网址<http://www.sqlite.org>

千里寻他众百度 锋自苦寒磨砺出

iOS数据持久化

- ✿ iOS数据持久化就是把数据存在文件中永久保存
- ✿ iOS的几种数据持久化保存方式
 - 1) plist文件(适合小型数据)
 - 2) 归档Archive(适合小型数据,存储对象)
 - 3) NSUserDefaults(适合小型数据)
 - 4) **SQLite数据库**(适合大型数据)
 - 5) CoreData数据库(适合大型数据)
 - 6) 普通文件(适合任何类型数据)(NSFileManager/NSFileHandle)

千里寻他众百度 锋自苦寒磨砺出

Contents

1. 数据库介绍

2. SQLite和SQL语句

3. FMDB开源数据库介绍

4. 数据库、多线程和缓存

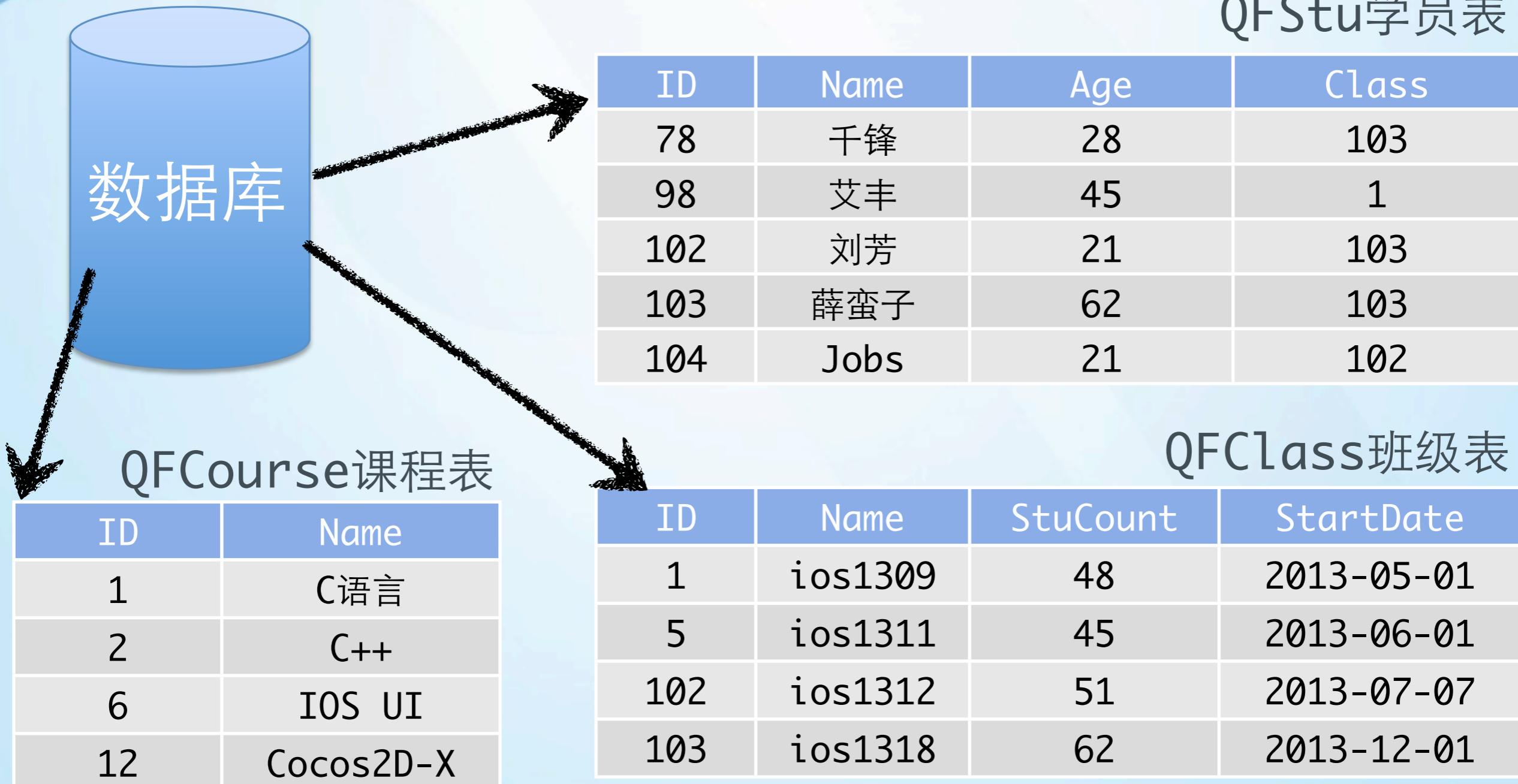
千里寻他众百度 锋自苦寒磨砺出

SQL语句介绍

- SQL Structured Query Language
- SQL是专门操作数据库的一种特定的语言
- SQL对于操作服务器数据库和手机SQLite数据库都是标准的
- SQL语句不区分大小写
- SQL语句主要做**增加、删除、修改、查询**操作

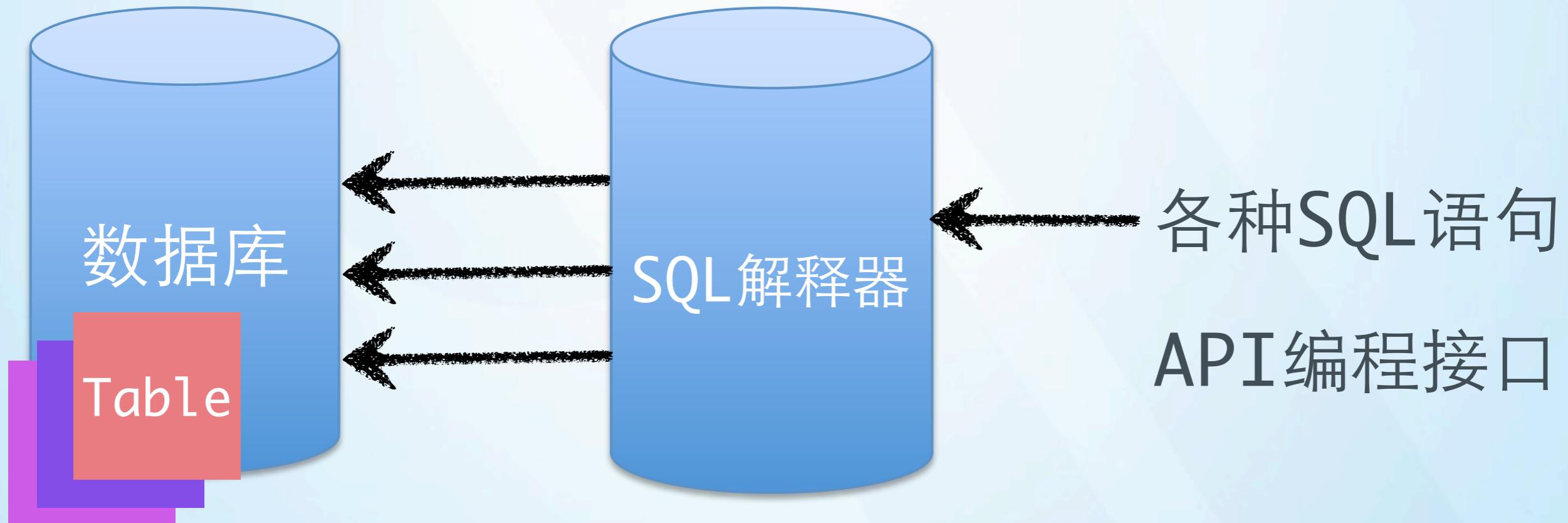
千里寻他众百度 锋自苦寒磨砺出

DB/Table关系



千里寻他众百度 锋自苦寒磨砺出

SQL语句介绍



千里寻他众百度 锋自苦寒磨砺出

SQL语句

- SQL语句：SQL是用来做数据的一些操作
- SQL语句不区分大小写
- 常见的SQL操作：
 - 增加(Insert)
 - 删除(Delete)
 - 修改(Update)
 - 查询(Select)

千里寻他众百度 锋自苦寒磨砺出

Contents

2. SQLite和SQL语句

2.1. 命令行SQL创建表

2.2. 命令行SQL Insert插入数据

2.3. 命令行SQL Select查询数据

2.4. 命令行SQL Delete删除数据

2.5. 命令行SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

创建数据库

```
YangMac-2:tmp yang$ sqlite3 qianfeng.db
SQLite version 3.7.12 2012-04-03 19:43:07
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
```

在命令行中输入 **sqlite3 qianfeng.db**

后面的qianfeng.db是数据库名称

一般命名为 xxx.db 或者 xxx.sqlite

千里寻他众百度 锋自苦寒磨砺出

创建/删除表

```
create table if not exists QFStu(  
    ID integer primary key autoincrement,  
    Name varchar(128),  
    Age integer,  
    Class interger default 0,  
    RegisterTime datetime,  
    Money float default 0,  
    Birthday date  
)
```

注意分号!!!

删除表：

```
drop table QFStu;
```

千里寻他众百度 锋自苦寒磨砺出

整数数据类型

- ◆ **integer**: 整型数据，大小为4个字节。
- ◆ **bigint**: 整型数据，大小为8个字节。
- ◆ **smallint**: 整数数据，大小为 2 个字节。
- ◆ **tinyint**: 从0到255的整数数据，存储大小为 1 字节。
- ◆ **float**: 4字节浮点数。
- ◆ **double**: 8字节浮点数。
- ◆ **real**: 8字节浮点数。

千里寻他众百度 锋自苦寒磨砺出

字符串数据类型

- **char(n)** n长度的字串，n不能超过254。
- **varchar(n)** 长度不固定且其最大长度为n的字串，n不能超过 4000。
- **text** text存储可变长度的非Unicode(统一字符编码)数据，存更大字符串
 - 尽量用**varchar**
 - 超过255字节的只能用**varchar**或者**text**
 - 能用**varchar**的地方不用**text**

千里寻他众百度 锋自苦寒磨砺出

sqlite字符串区别

1. CHAR存储定长数据很方便，CHAR字段上的索引效率级高，比如定义char(10)，那么不论你存储的数据是否达到了10个字节，都要占去10个字节的空间，不足的自动用空格填充。
2. VARCHAR存储变长数据，但存储效率没有CHAR高。如果一个字段可能的值是不固定长度的，我们只知道它不可能超过10个>字符，把它定义为VARCHAR(10)是最合算的。VARCHAR类型的实际长度是它的值的实际长度+1。为什么“+1”呢？这一个字节用于保存实际使用了多大的长度。从空间上考虑，用varchar合适；从效率上考虑，用char合适，关键是根据实际情况找到权衡点。
3. TEXT存储可变长度的非Unicode数据，最大长度为 $2^{31}-1$ (2,147,483,647)个字符。

千里寻他众百度 锋自苦寒磨砺出

日期类型

- **date**: 包含了 年份、月份、日期。
- **time**: 包含了 小时、分钟、秒。
- **datetime**: 包含了 年、月、日、时、分、秒。
- **timestamp**: 包含了 年、月、日、时、分、秒、千分之一秒。

datetime 包含日期时间格式，必须写成 '2010-08-05' 不能写为 '2010-8-5'，否则在读取时会产生错误！

千里寻他众百度 锋自苦寒磨砺出

其它类型

- null: 空值。
- blob: 二进制对象。
- default: 缺省值
- primary key: 主键
- autoincrement: 主键自动增长

千里寻他众百度 锋自苦寒磨砺出

什么是主键

- primary key
- 主键就是一个表中，有一个字段，里面的内容不可以重复。
- 一般一个表都需要设置一个主键
- 比如QFStu中Name名字字段可以重复，但是主键不可以重复
- autoincrement这样让主键自动增长

千里寻他众百度 锋自苦寒磨砺出

各种创建Table

```
create table if not exists QFClass(Name  
varchar(256), StuCount smallint, StartDate  
date);
```

```
create table if not exists QFCourse(ID  
integer, Name varchar(256));
```

千里寻他众百度 锋自苦寒磨砺出

Contents

2. SQLite和SQL语句

2.1. 命令行SQL创建表

2.2. 命令行SQL Insert插入数据

2.3. 命令行SQL Select查询数据

2.4. 命令行SQL Delete删除数据

2.5. 命令行SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

插入语句

```
insert into QFStu(Name) values ('千锋');  
insert into QFStu(Name, Age) values ('艾丰',  
21);  
insert into QFStu(Name, Age, Class,  
RegisterTime, Money, Birthday)  
values ('yang', 35, 12,  
'2013-09-10 21:08:12',  
1092.4, '1990-01-01');  
select * from QFStu;
```

千里寻他众百度 锋自苦寒磨砺出

注意事项

- 所有字符串必须要加 ' ' 单引号
- 整数，浮点数不用加 ' '
- 日期需要加单引号 ' '
- 字段顺序没有关系
- 对于自动增长的主键不需要插入字段

千里寻他众百度 锋自苦寒磨砺出

Contents

2. SQLite和SQL语句

2.1. 命令行SQL创建表

2.2. 命令行SQL Insert插入数据

2.3. 命令行SQL Select查询数据

2.4. 命令行SQL Delete删除数据

2.5. 命令行SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

查询Select

- ◆ select * from QFStu;
- ◆ select Name, Age from QFStu;
- ◆ select Name, Age, RegisterTime,
Money, Birthday from QFstu;
- ◆ select ID, Age, Name, RegisterTime,
Money, Birthday from QFstu;

千里寻他众百度 锋自苦寒磨砺出

where查询

- ✿ select ID, Age, Name, RegisterTime, Money, Birthday from QFstu where Age>10;
- ✿ select ID, Age, Name, RegisterTime, Money, Birthday from QFstu where Age>10 and ID>2;
- ✿ select ID, Age, Name, RegisterTime, Money, Birthday from QFstu where Name like '%千锋%';
- ✿ select RegisterTime from QFStu where RegisterTime > '2013-01-01 01:01:01';

千里寻他众百度 锋自苦寒磨砺出

简单的where条件

- = 等于
- > 大于
- < 小于
- >= 大于等于
- <= 小于等于
- <> 不等于
- !> 不大于
- %千锋% 字符串匹配含有 千锋2字的

千里寻他众百度 锋自苦寒磨砺出

select统计语句

- 查找QFStu一共有多少条记录
- select count(*) from QFStu;
- select avg(Age) from QFStu;
- select sum(Age) from QFStu;

千里寻他众百度 锋自苦寒磨砺出

Contents

2. SQLite和SQL语句

2.1. 命令行SQL创建表

2.2. 命令行SQL Insert插入数据

2.3. 命令行SQL Select查询数据

2.4. 命令行SQL Delete删除数据

2.5. 命令行SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

删除记录

- 删除指定ID值为2的记录
- delete from QFStu where ID=2;
- 删除QFStu表中所有的记录(慎重)
- delete from QFStu;

千里寻他众百度 锋自苦寒磨砺出

Contents

2. SQLite和SQL语句

2.1. 命令行SQL创建表

2.2. 命令行SQL Insert插入数据

2.3. 命令行SQL Select查询数据

2.4. 命令行SQL Delete删除数据

2.4. 命令行SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

更新/修改Update

- 更新ID为1的更新Name字段

```
update QFStu set Name='千锋iOS' where ID=1;
```

- 更新ID为1的Name和Age字段

```
update QFStu set Name='千锋iOS', Age=100 where ID=1;
```

- 更新所有记录的字段

```
update QFStu set Name='千锋iOS', Age=100
```

- 将ID=1的记录的Age值增加10

```
update QFStu set Age=Age+10 where ID=1;
```

千里寻他众百度 锋自苦寒磨砺出

SQLite命令行

- .help (帮助)
- .quit/.exit (退出)
- .database (列出数据库信息)
- .dump (查看所有的sql语句)
- .schema (查看表结构)
- .tables (显示所有的表)

千里寻他众百度 锋自苦寒磨砺出

Contents

1. 数据库介绍

2. SQLite和SQL语句

3. FMDB开源数据库介绍

4. 数据库和多线程

5. 多表联合查询

千里寻他众百度 锋自苦寒磨砺出

FMDB介绍

- FMDB是一个开源的第三方操作数据库的框架。因为IOS官方提供操作数据库的方法使用起来麻烦，FMDB是把官方提供的方法封装了一层，只是为了更方便开发使用，核心的代码约500多行。
- FMDB均支持ARC和Non-ARC/MRC操作
- 网址:<https://github.com/ccqus/fmdb>

千里寻他众百度 锋自苦寒磨砺出

FMDB的三个类

- FMDatabase – 表示一个单独的SQLite数据库。用来执行SQLite的命令。
- FMResultSet – 表示FMDatabase执行查询后结果集
- FMDatabaseQueue – 如果想在多线程中执行多个查询或更新，应该使用该类。这是线程安全的。

千里寻他众百度 锋自苦寒磨砺出

Contents

3. FMDB开源数据库介绍

3.1. 代码SQL创建表

3.2. 代码SQL Insert插入数据

3.3. 代码SQL Select查询数据

3.4. 代码SQL Delete删除数据

3.5. 代码SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

FMDB创建数据库

// 1. 设置数据库文件路径(这里存在沙盒里面)

```
NSString *path = [NSHomeDirectory()
    stringByAppendingPathComponent:@"/Documents/qianfeng.db"];
```

// 2. 创建数据库对象，数据库还没有打开

```
FMDatabase *fmdb = [FMDatabase databaseWithPath:path];
```

// 3. 打开创建好的数据库对象

```
BOOL ret = [fmdb open];
```

// 4. ret表示是否成功或者失败

```
if (ret == NO) {
    NSLog(@"打开数据库失败");
}
```

千里寻他众百度 锋自苦寒磨砺出

FMDB创建表

// 1. 定义创建表的SQL语句

```
NSString *sql = @"create table if not exists QFStu(ID  
integer primary key autoincrement, Name varchar(128), Age  
integer, Class interger default 0, RegisterTime datetime,  
Money float default 0, Birthday date);";
```

// 2. 执行所有这条SQL语句(除了select的所有语句)

```
ret = [fmdb executeUpdate:sql];
```

// 3. ret表示是否成功或者失败

```
if (ret == NO) {  
    NSLog(@"创建表失败");  
}
```

千里寻他众百度 锋自苦寒磨砺出

Contents

3. FMDB开源数据库介绍

3.1. 代码SQL创建表

3.2. 代码SQL Insert插入数据

3.3. 代码SQL Select查询数据

3.4. 代码SQL Delete删除数据

3.5. 代码SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

FMDB插入记录(1)

// 1. 定义创建表的SQL语句

```
NSString *sql = @"insert into QFStu(Name, Age, Class, "
    " RegisterTime, Money, Birthday) "
    " values ('yang', 35, 12, "
    " '2013-09-10 21:08:12', "
    " 1092.4, '1990-01-01');";
```

// 2. 执行所有这条SQL语句(除了select的所有语句)

```
ret = [fmdb executeUpdate:sql];
```

// 3. ret表示是否成功或者失败

```
if (ret == NO) {
    NSLog(@"创建表失败");
}
```

千里寻他众百度 锋自苦寒磨砺出

FMDB插入记录(2)

```
NSString *sql = @"insert into QFStu(Name, Age, Class,  
RegisterTime, Money, Birthday) values  
(?, ?, ?, ?, ?, ?);";
```

```
[fmdb executeUpdate:sql,  
 @"yang", @"35", @"12",  
 @"2013-09-10 21:08:12",  
 @"1092.4",  
 @"1990-01-01"];
```

这里使用了?通配符来动态的插入

千里寻他众百度 锋自苦寒磨砺出

Contents

3. FMDB开源数据库介绍

3.1. 代码SQL创建表

3.2. 代码SQL Insert插入数据

3.3. 代码SQL Select查询数据

3.4. 代码SQL Delete删除数据

3.5. 代码SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

FMDB查询记录

```
// 1. 定义SQL语句
NSString *sql = @"select * from QFStu;";
// 2. 执行所有这条select SQL语句
FMResultSet *set = [fmdb executeQuery:sql];
while ([set next]) {
    NSUInteger pid = [set intForColumn:@"ID"];
    NSString *name = [set stringForColumn:@"Name"];
    NSString *datetime = [set
        stringForColumn:@"RegisterTime"];
    NSDate *d = [self dateString:datetime];
    NSLog(@"%@", pid, name, date);
}
[set close];
```

千里寻他众百度 锋自苦寒磨砺出

NSDate<->NSString

```
- (NSDate *)dateFromString:(NSString *)dateString{
    NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
    [dateFormatter setDateFormat: @"yyyy-MM-dd HH:mm:ss"];
    NSDate *destDate= [dateFormatter dateFromString:dateString];
    [dateFormatter release];
    return destDate;
}

- (NSString *)stringFromDate:(NSDate *)date{
    NSDateFormatter *dateFormatter = [[NSDateFormatter alloc] init];
    //zzz表示时区， zzz可以删除，这样返回的日期字符将不包含时区信息 +0000。
    [dateFormatter setDateFormat:@'"yyyy-MM-dd HH:mm:ss zzz"'];
    NSString *destDateString = [dateFormatter stringFromDate:date];
    [dateFormatter release];
    return destDateString;
}
```

千里寻他众百度 锋自苦寒磨砺出

对于查询结果是标量

```
// 1. 定义SQL语句
NSString *sql = @"select count(*) from QFStu;";
// 2. 执行所有这条select SQL语句
FMResultSet *set = [fmdb executeQuery:sql];
NSUInteger count = 0;
while ([set next]) {
    count = [set intForColumnIndex:0];
}
[set close];

NSLog(@"%@", @"count is %d ", count);

/*
    对于标量是数据返回的是 1x1矩阵 所以这一列，这里使用0
*/
```

千里寻他众百度 锋自苦寒磨砺出

Contents

3. FMDB开源数据库介绍

3.1. 代码SQL创建表

3.2. 代码SQL Insert插入数据

3.3. 代码SQL Select查询数据

3.4. 代码SQL Delete删除数据

3.5. 代码SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

FMDB删除记录

```
// 1. 定义创建表的SQL语句
NSString *sql = @"delete from QFStu where ID=2;";

// 2. 执行所有这条SQL语句(除了select的所有语句)
ret = [fmdb executeUpdate:sql];

// 3. ret表示是否成功或者失败
if (ret == NO) {
    NSLog(@"创建表失败");
}

// 4. 使用通配符进行处理
ret = [fmdb executeUpdate:
        @"delete from QFStu where ID=?", @"3"];
if (ret == NO) {
    NSLog(@"删除失败");
}
```

千里寻他众百度 锋自苦寒磨砺出

Contents

3. FMDB开源数据库介绍

3.1. 代码SQL创建表

3.2. 代码SQL Insert插入数据

3.3. 代码SQL Select查询数据

3.4. 代码SQL Delete删除数据

3.5. 代码SQL Update更新数据

千里寻他众百度 锋自苦寒磨砺出

FMDB修改记录

```
// 1. 定义创建表的SQL语句
NSString *sql = @"update QFStu set Name='千锋2' where
ID=9;";

// 2. 执行所有这条SQL语句(除了select的所有语句)
ret = [fmdb executeUpdate:sql];

// 3. ret表示是否成功或者失败
if (ret == NO) {
    NSLog(@"创建表失败");
}

// 4. 使用通配符进行处理
ret = [fmdb executeUpdate:@"update QFStu set Name='千锋2'
where ID=?", @"10"];
```

千里寻他众百度 锋自苦寒磨砺出

FMDatabase操作总结

```
+ (id)databaseWithPath:(NSString*)inPath;
- (id)initWithPath:(NSString*)inPath;
- (BOOL)open;
- (BOOL)close;
- (NSString *)databasePath;

- (sqlite_int64)lastInsertRowId;

// 执行SQL 增加 修改 删除 创建表操作调用函数
- (BOOL)executeUpdate:(NSString*)sql, ...;
// 执行SQL 查询select语句专用函数
- (FMResultSet *)executeQuery:(NSString*)sql, ...;
```

千里寻他众百度 锋自苦寒磨砺出

变长参数总结

- (BOOL)executeUpdate:(NSString*)sql, ...;
- (FMResultSet *)executeQuery:(NSString*)sql, ...;

对于变长参数：

这里需要注意的是，参数必须是NSObject的子类，所以象int,double,bool这种基本类型，需要封装成对应的包装类才行，如下所示：

```
// 错误，42不能作为参数
[db executeUpdate:@"insert into QFStu VALUES (?)", 42];
// 正确，将42封装成 NSNumber 类
[db executeUpdate:@"insert into QFStu VALUES (?)",
[NSNumber numberWithInt:42]];
```

Contents

1. 数据库介绍

2. SQLite和SQL语句

3. FMDB开源数据库介绍

4. 数据库、多线程和缓存

千里寻他众百度 锋自苦寒磨砺出

Contents

4. 数据库和多线程

4.1. 数据库和UITableView关联

4.2. 封装数据库单例类

4.3. 数据库缓存图片

4.4. 数据库和多线程处理

千里寻他众百度 锋自苦寒磨砺出

TableView和数据库

- 读取数据库内容存于数据模型UserModel中
- 创建UserCell来显示每个Cell
- 关联UserCell和UserModel

千里寻他众百度 锋自苦寒磨砺出

Contents

4. 数据库和多线程

4.1. 数据库和UITableView关联

4.2. 封装数据库单例类

4.3. 数据库缓存图片

4.4. 数据库和多线程处理

千里寻他众百度 锋自苦寒磨砺出

Contents

4. 数据库和多线程

4.1. 数据库和UITableView关联

4.2. 封装数据库单例类

4.3. 数据库缓存图片

4.4. 数据库和多线程处理

千里寻他众百度 锋自苦寒磨砺出

图片数据

图片缓存的方式有2种

1. 讲图片二进制存于表中一列
2. 把图片存于沙盒中，在表中存放相对路径

对于图片存于沙盒直接使用

HeaderImage blob创建图片二进制字段

HeaderImagePath varchar(128)创建图片路径

千里寻他众百度 锋自苦寒磨砺出

图片如何写入文件

● 图片写到文件中

```
- (void) writeImage:(UIImage *)img toFile:(NSString *)file {  
    NSData *d = UIImagePNGRepresentation(img);  
    if (d == nil) {  
        d = UIImageJPEGRepresentation(img, 1);  
    }  
    [d writeToFile:file atomically:YES];  
}
```

千里寻他众百度 锋自苦寒磨砺出

写入到系统相册

```
- (void) writeImageToSystemPhoto:(UIImage *)img {
    UIImageWriteToSavedPhotosAlbum(img,
        self,
        @selector(image:didFinishSavingWithError:contextInfo:),
        nil);
}

- (void)image:(UIImage *)image
    didFinishSavingWithError:(NSError *)error
    contextInfo:(void *)contextInfo {
    if (error != NULL) {

    } else {

    }
}
```

千里寻他众百度 锋自苦寒磨砺出

图片直接写数据库

```
- (BOOL) addOneRecord:(UserModel *)model {
    NSString *sql = @"insert into QFStu(Name, Age, HeaderImage) values
(?, ?, ?)";
    NSData *headerData = UIImagePNGRepresentation(model.headerImage);
    BOOL ret = [_db executeUpdate:sql, model.name, [NSNumber
numberWithInt:model.age], headerData];
    return ret;
}

- (BOOL) modifyOneRecord:(UserModel *)model {
    NSString *sql = @"update QFStu set Name=?, Age=?, HeaderImage=?
where ID=?";
    NSData *headerData = UIImagePNGRepresentation(model.headerImage);
    BOOL ret = [_db executeUpdate:sql, model.name, [NSNumber
numberWithInt:model.age],
                headerData, [NSNumber numberWithInt:model.uid]];
    return ret;
}
```

千里寻他众百度 锋自苦寒磨砺出

Contents

4. 数据库和多线程

4.1. 数据库和UITableView关联

4.2. 封装数据库单例类

4.3. 数据库缓存图片

4.4. 数据库和多线程处理

千里寻他众百度 锋自苦寒磨砺出

多线程特殊处理

注意多个线程不能同时操作数据库(sqlite数据库是线程不安全的)

解决方法：

- 设计加锁操作(自己使用NSLock设计)
- 使用FMDatabaseQueue处理(线程安全的)

千里寻他众百度 锋自苦寒磨砺出

Thread/DB总结

- sqlite3数据库是线程不安全的Thread not-safety。在多线程中需要加锁NSLock或者使用Queue模式(实质也是加锁)。保证任何时候只能一个线程操作数据库
- 数据库封装成单例也需要加锁，这里使用了系统自带的 **@synchronized(self)**(更多单例写法参考千锋OC单例部分视频)
- 注意多界面Controller使用数据库和多线程使用数据库性质不一样，多界面本质上还是在一个线程中。

千里寻他众百度 锋自苦寒磨砺出

- 2) 主键：当有多个候选码时，可以选定一个作为主码，选定的候选码称主键
- 3) 外键：关系R中的一个属性组，它不是R的候选码，但它与另一个关系S的候选码相对应，则称这个属性组为R的外码或外键>。

举个例子：

有两个关系：

`student(s#, sname, d#)`, 即学生这个关系有三个属性：学号，姓名，所在系别

`dep(d#, dname)`, 即院系有两个属性：系号、系名

则`s #`、`d #`是主键，也是各自所在关系的唯一候选键，`d #`是`student`的外键。

千里寻他众百度 锋自苦寒磨砺出

谢谢大家！

- 千锋互联：www.1000phone.com
- 千锋3G学院：www.mobiletrain.org
- 千锋嵌入式学院：www.embedtrain.org

千里寻他众百度 锋自苦寒磨砺出