More Persistence & Advanced Core Data

LIGHTHOUSE LABS

Data Persistence Options on iOS

- Files
- NSUserDefaults
- plist files
- Keychain
- Direct SQL
- Archiving
- Core Data

Archiving/Unarchiving

This option lets you store your own objects (including relationships) on disk and then instantiate them again later. It is commonly referred to as “serialization” in some other languages/platforms. Interface Builder also uses this technique for storing your view hierarchies.

Objects that are being archived are responsible for knowing how to archive themselves. They must conform to the `NSCoding` protocol.

Archiving/Unarchiving

```
@interface Person : NSObject <NSCoding>
@property (nonatomic, copy) NSString* firstName;
@property (nonatomic, copy) NSString* lastName;
@property (nonatomic, copy) NSNumber* idNumber;
@end
```

LIGHTHOUSE LABS

Archiving/Unarchiving

```
@implementation Person

-(void)encodeWithCoder:(NSCoder*)coder {
    [coder encodeObject:self.firstName forKey:@”first”];
    [coder encodeObject:self.lastName forKey:@”last”];
    [coder encodeObject:self.idNumber forKey:@”idNum”];
}

@end
```

LIGHTHOUSE LABS

Archiving/Unarchiving

```
-(id)initWithCoder:(NSCoder*)coder {
    self = [super init];
    if (!self) return nil;
    self.firstName = [coder decodeObjectForKey:@”first”];
    self.lastName  = [coder decodeObjectForKey:@”last”];
    self.idNumber  = [coder decodeObjectForKey:@”idNum”];
    return self;
}

// cont’d
```

LIGHTHOUSE LABS

Archiving/Unarchiving

`NSCoder` is an abstract class. Apple also provides `NSKeyedArchiving` and `NSKeyedUnarchiving` subclasses.

You can use the first to archive your objects into `NSData` objects, which you can then write to file; and the second to unarchive data that you've read from a file.

Archiving/Unarchiving - 2 ways

```
// Saving to file Path
[NSKeyedArchiver archiveRootObject:books toFile:@"
/path/to/archive"];

// NSUserDefaults
NSData *data = [NSKeyedArchiver archivedDataWithRootObject:
books];
[[NSUserDefaults standardUserDefaults] setObject:data forKey:
@"books"];
```

LIGHTHOUSE LABS

Keychain

Each application has its own keychain to which only the application has access.

This ensures that the data stored in the keychain is safe and unaccessible by third parties.

LIGHTHOUSE LABS

A Note On Preserving UI

UIKit provides functionality to help you preserve the UI of your app and restore it the next time your app gets launched. This is handy for allowing the user to pick up where they left off.

DEMO

- Archiving a custom object

Persistence Recap

So far we’ve seen a few different ways to persist data. Most of these involve saving data to files somewhere on disk. But none of them really work well at large scale. For example, how would you search for a particular item if you have a lot of archived objects on disk? You’d have to load all of them in memory and then loop over them to find ones that match certain conditions.

LIGHTHOUSE LABS

Persistence Recap

That problem gets alleviated when using a relational database (SQLite is the only real choice on mobile). Database queries make finding specific records much faster. However, databases work with data, not objects. You’d have to write a lot of boilerplate code to ensure the integrity of your data relationships at the object layer.

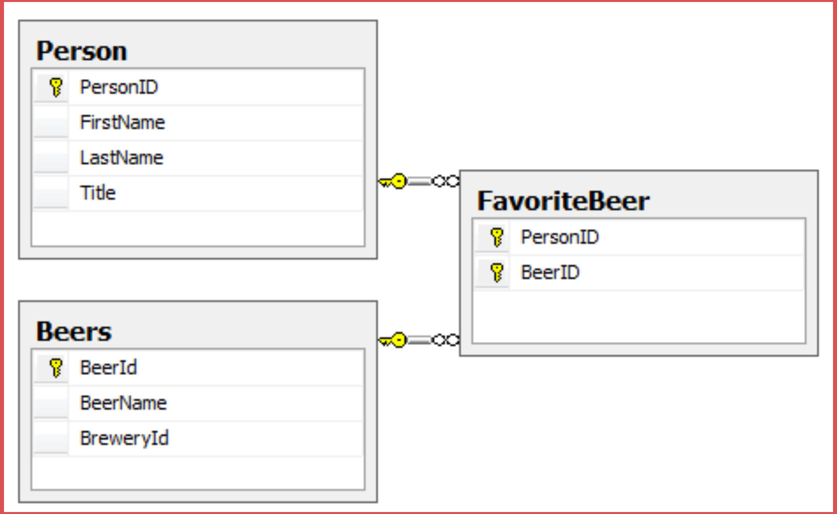
Back to Core Data

Core Data is an “Object Graph” and object lifecycle management framework, not a database. It *can* persist to a database.

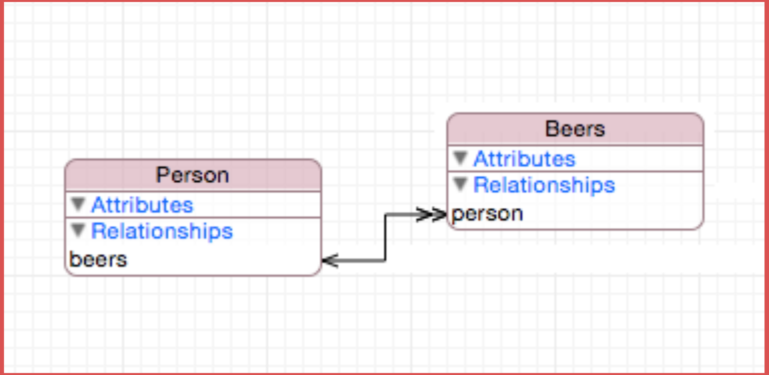
That means it gives you much of the same win as a database, but uses a slightly different paradigm.

Core Data != Database

SQL Database Join Table



Core Data



LIGHTHOUSE LABS

Querying Core Data

- Fetch request (NSFetchRequest)
- Fetched results controller (NSFetchedResultsController)
- Search predicate (NSPredicate)

```
NSString *firstNameString = @”Rick”;
NSString *lastNameString = @”Deckard”;

[NSPredicate predicateWithFormat:@”(firstName like %@) OR
(lastName like %@)”, firstNameString, lastNameString ];
```

LIGHTHOUSE LABS

Fetch requests

- Entity and context to search (NSManagedObject, or subclass)
- Fetch batch size (0 is infinite)
- Sort description (NSSortDescriptor)
- Predicate (optional)

Can be embedded in an NSFetchedResultsController for more “smart” behaviour.

DEMO

- Searching with predicates
- One-to-many relationships
- Many-to-many relationships

LIGHTHOUSE LABS

Migrations & Versioning