Grouping a Core Data data result?

Asked 14 years, 9 months ago Modified 14 years, 9 months ago

Viewed 2k times

2

I am prototyping an idea on the iPhone but I am at the SQLite vs CoreData crossroads. The main reason is that I can't seem to figure out how to do grouping with core data.

Essentially I want to show the most recent item posted grouped by username. It is really easy to do in a SQL statement but I have not been able to make it work in core data. I figure since I am starting a new app, I might as well try to make core data work but this part is a major snag.

I added a predicate to my fetchrequest but that only gave me the single most recently added record and not the most recently added record per user.

The data model is pretty basic at this point. It uses the following fields: username (string), post (string), created (datetime)

So long story short, are these types of queries possible with CoreData? I imagine that if SQLite is under the hood, there has to be some way to do it.

objective-c cocoa sqlite core-data

Share

Improve this question

Follow

asked Feb 28, 2010 at 6:28

SonnyBurnette
680 ● 3 ● 12 ● 25

3 Answers

Sorted by: Highest score (default)

15

First of all, don't think of Core Data as another way of doing SQL. SQL is not "under the hood" of Core Data. Core Data deals with objects. Entity descriptions are not tables and entity instances are not records. Programming with Core Data has nothing to do with SQL, it merely uses SQL as one of several possible types of persistent stores. You don't deal with it directly and should never, ever think of Core Data in SQL terms.

That way lies madness.

You need drink a lot of tequila and punch yourself in the head repeatedly until you forget everything you ever knew about SQL. Otherwise, you will just end up with an object graph that is nothing but a big spread sheet.

There are several ways to accomplish what you want in Core Data. Usually you would construct fetch with a compound predicate that would return all post within a certain date range made by a specific user. Fetched results controllers are especially handy for this.

A most straightforward method would be to set up you object graph like:

```
UserEntity
--Attribute username
--Relationship post <-->> PostEntity

PostEntity
--Attribute creationDate
--Attribute content
-- Relationship user <<--> UserEntity
```

Then in your UserEntity class have a method like so:

```
- (NSArray *) mostRecentPost{
    NSPredicate *recentPred=[NSPredicate
predicateWithFormat:@"creationDate>%@", [NSDate dateWi
(60*60*24)]];
    NSSet *recentSet=[self.post filteredSetUsingPredic
    NSSortDescriptor *dateSort=[[NSSortDescriptor allo
initWithKey:@"creationDate" ascending:NO];
    NSArray *returnArray=[[recentSet allObjects] sorte
[NSArray arrayWithObject:dateSort]];
    return returnArray;
}
```

When you want a list of the most recent post of a particular user sorted by date just call:

```
NSArray *arrayForDisplay=[aUserEntityClassInstance mos
```

Edit:

> ...do I just pass each post block of data (content,creationDate) to the post entity? Do I

> also pass the username to the post entity? How does the user entity know when to create a new user?

Let me pseudo code it. You have two classes that define instances of userObj and a postObj. When a new post comes in, you:

```
Parse inputPost for a user;
Search existing userObj for that name;
if userObj with name does not exist
    create new userObj;
    set userObj.userName to name;
else
    return the existing userObj that matches the name;
Parse inputPost for creation date and content;
Search post of chosen userObj;
if an exiting post does not match content or creation
    create new postObj
    set postObj.creationDate to creation date;
    set postObj,content to content;
    set postObj.user to userObj; // the reciprocal in
automatically
else // most likely ignore as duplicate
```

You have separate userObj and postObj because while each post is unique, each user may have many post.

The important concept to grasp is that your dealing with object i.e. encapsulated instance of data AND logic. This isn't just rows and columns in a db. For example, you could write managed object subclasses in which a single instance could decide whether to form a relationship with an instance of another class unless a specific internal state of the object was reached. Records in dbs don't have that sort of logic or autonomy.

The best way to get a handle on using objects graphs for data models is to ignore not only db but Core Data itself. Instead, set out to write a small test app in which you hand code all the data model classes. It doesn't have to be elaborate just a couple of attributes per class and a reference of some sort to the other class. Think about how you would manage parsing the data out to each class, linking the classes and their data together and then getting it back out. Do that by hand once or twice and the nature of object graphs becomes readily apparent.

Share Improve this answer Follow

edited Feb 28, 2010 at 23:52

answered Feb 28, 2010 at 9:18

TechZen
64.4k ● 15 ● 120 ● 146

Thanks for the thorough explanation and +1 for "You need drink a lot of tequila and punch yourself in the head repeatedly until you forget everything you ever knew about SQL." Just to be sure I understand this from a Non-SQL angle, when I parse new data (xml,json,etc) coming in consisting of posts with their user listed, do I just pass each post block of data (content,creationDate) to the post entity? Do I also pass the username to the post entity? How does the user entity know when to create a new user? Does it?
– SonnyBurnette Feb 28, 2010 at 17:24

The pseudo code did it. I wanted to make sure I was responsible for creating the logic to look if the user existed or if that happened automagically. Thanks again. – SonnyBurnette Mar 1, 2010 at 0:42

Should that UserEntity class method above in the example code be a class method (with a +) or an instance method as you've shown? – Michael Morrison Jan 17, 2011 at 5:35

It has to be a instance method because it relies on `self.post` which refers to the `post` attribute of the UserEntity instance. – TechZen Feb 28, 2011 at 17:44

1

There are other considerations that might tip your decision in the direction of SQLite versus Core Data with a SQLite store. I found myself nodding in agreement while reading a good blog post on the subject. I've found exactly the same thing (and am consequently moving a high-performance app away from Core Data): "Core Data is the right answer, except when it's not..."

It's a great technology, but one size definitely does not fit all.

Share Improve this answer Follow

answered Feb 28, 2010 at 12:42

Joshua Nozzi
61.2k ● 14 ● 142 ● 136

Core Data is optimized to handle complexity and SQL is optimized to handle size. If you have a large number of simple, largely unrelated records, then raw SQL will give superior performance. If you have a relatively small number (absolute) of many different logical entities with many interconnecting logical relationships, then Core Data is clearly superior. – TechZen Feb 28, 2010 at 16:35

I also read that post which is why I am very much on the fence about dropping my experience with SQL. The only thing I can do for now, I guess, is mock-up both versions and see which feels more like I am painting myself into a corner. – SonnyBurnette Feb 28, 2010 at 17:30

As generalizations go, TechZen, that's a pretty sound one. :-) – Joshua Nozzi Feb 28, 2010 at 18:13

0

If 'posts' is a NSSet of User, you could get the last post with a predicate:

```
NSDate *lastDate = [userInstance valueForKeyPath:@"@ma

NSSet *resultsTemp = [setOfPosts filteredSetUsingPredi
predicateWithFormat:@"fecha==%@", lastDate] ];
```

The resultsTemp set will contain an object of type Post which has the newest date.

Share Improve this answer Follow

answered Feb 28, 2010 at 7:32

iOSCowboy
1,426 ● 2 ● 9 ● 9