CS 696 Mobile Application Development
Fall Semester, 2010
Doc 16 Core Data
Oct 28, 2010

O-R Mapping - Vietnam of Computer Science

http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx

http://www.codinghorror.com/blog/archives/000621.html

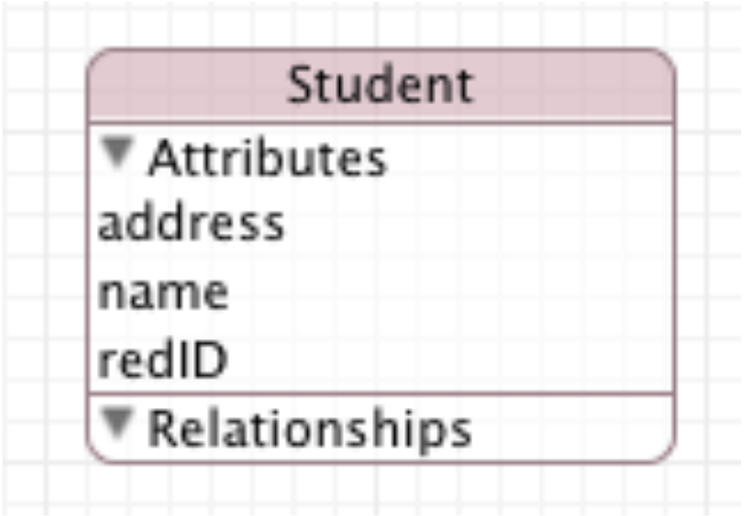
Last mile problem & OR mapping Problem

- Dual-Schema Problem
- Entity Identity Issues
- The Data Retrieval Mechanism
  - Query-By-Example (QBE)
  - Query-By-API (QBA)
  - Query-By-Language (QBL)

Core Data

- Object-relational mapping layer
- Object-graph management and persistence framework
  - Makes it easy to save & load objects
  - Supports undo/redo
  - Higher level than SQLite
  - Claim 50% to 70% less code
- On Mac OS & iOS

Entities & Attributes

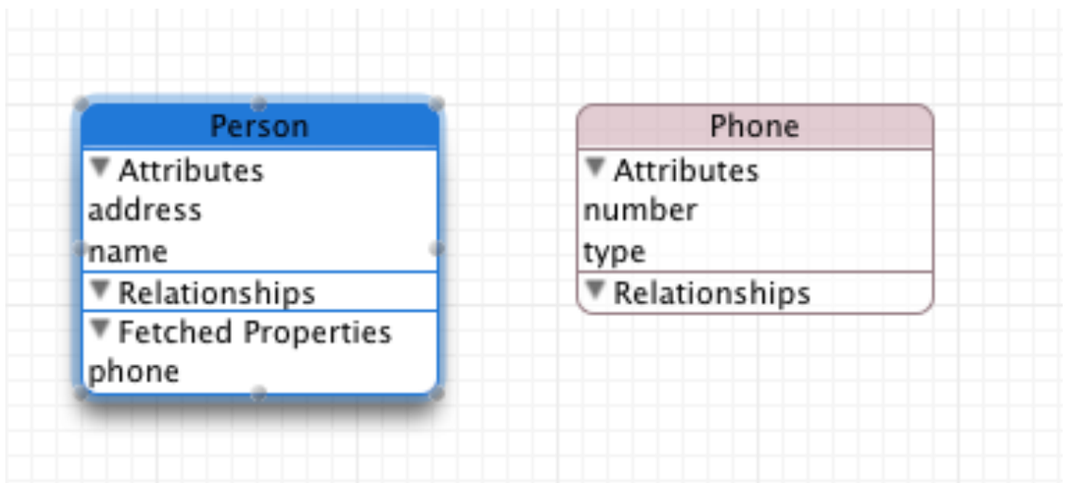


Entities (table)

- Like class with only instance variables
- Define data for managed objects

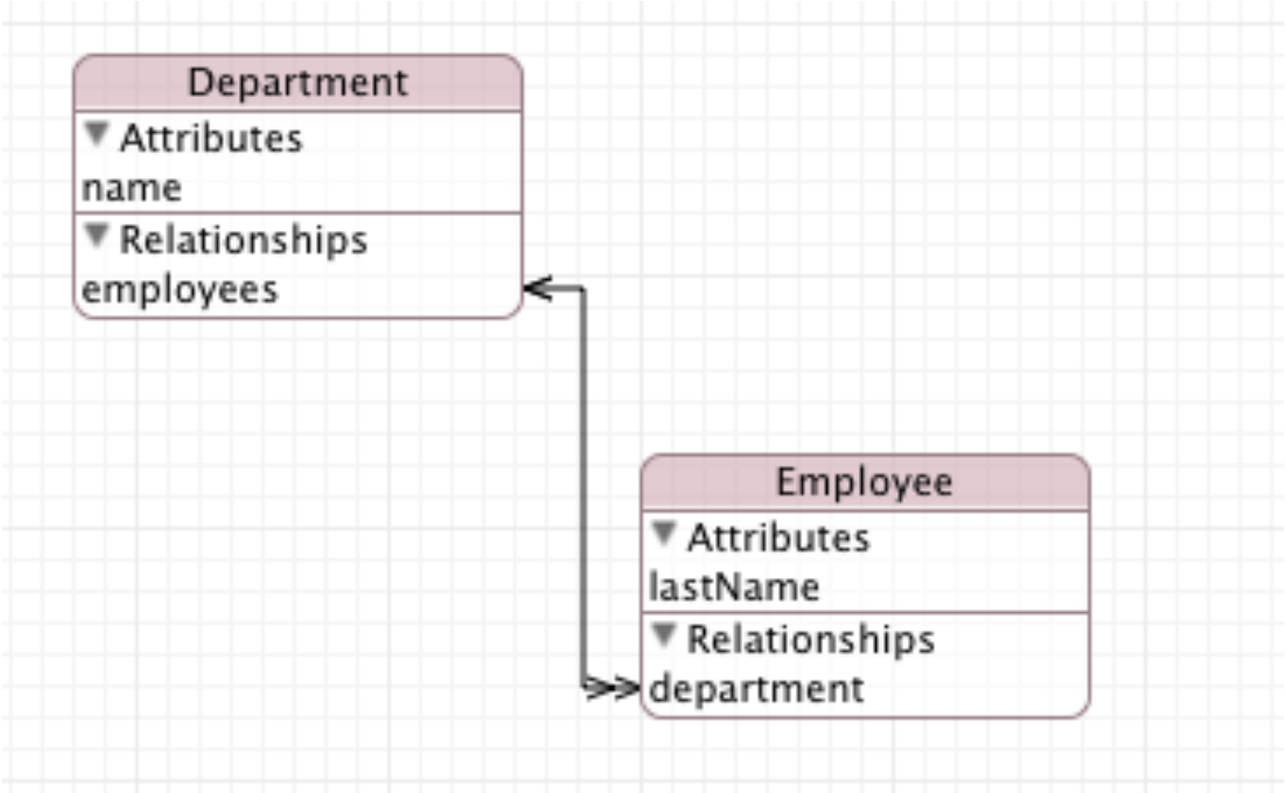
Attribute (column)

- Like field in class
- Basic types
  - Int, decimal, float, bool, date, binary
- Can use other types (with more work)



Relationships

(Foreign key)



Entity uses other entity

to many

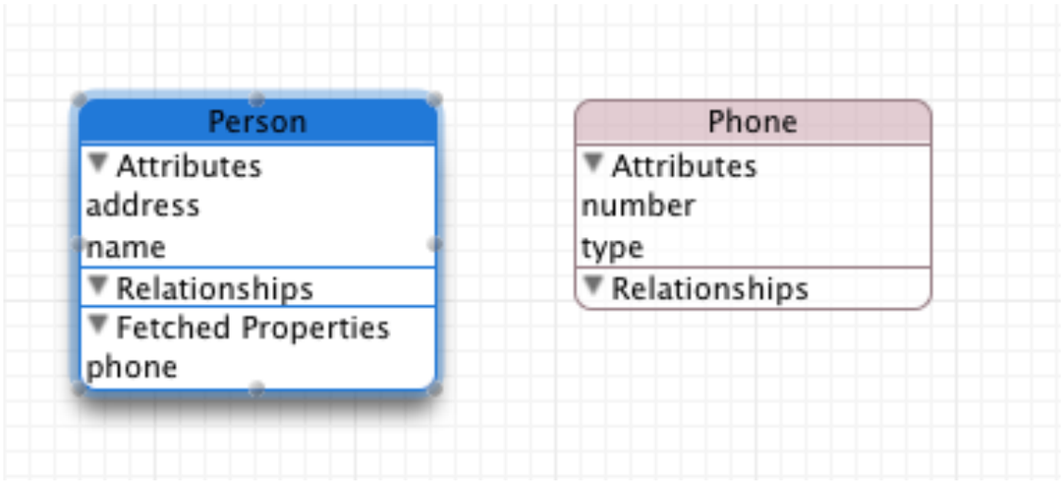
to one

Normally define relationship in both directions

When load object also load its relationships

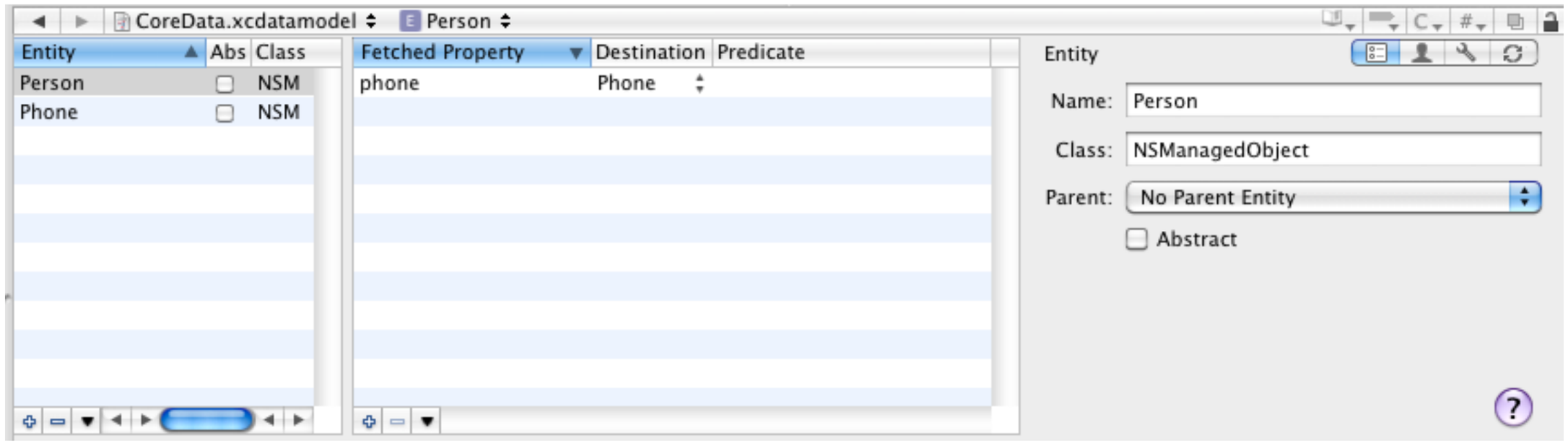
Fetched Properties

(Foreign key, lazy load)



Property is loaded only when needed

Behavior



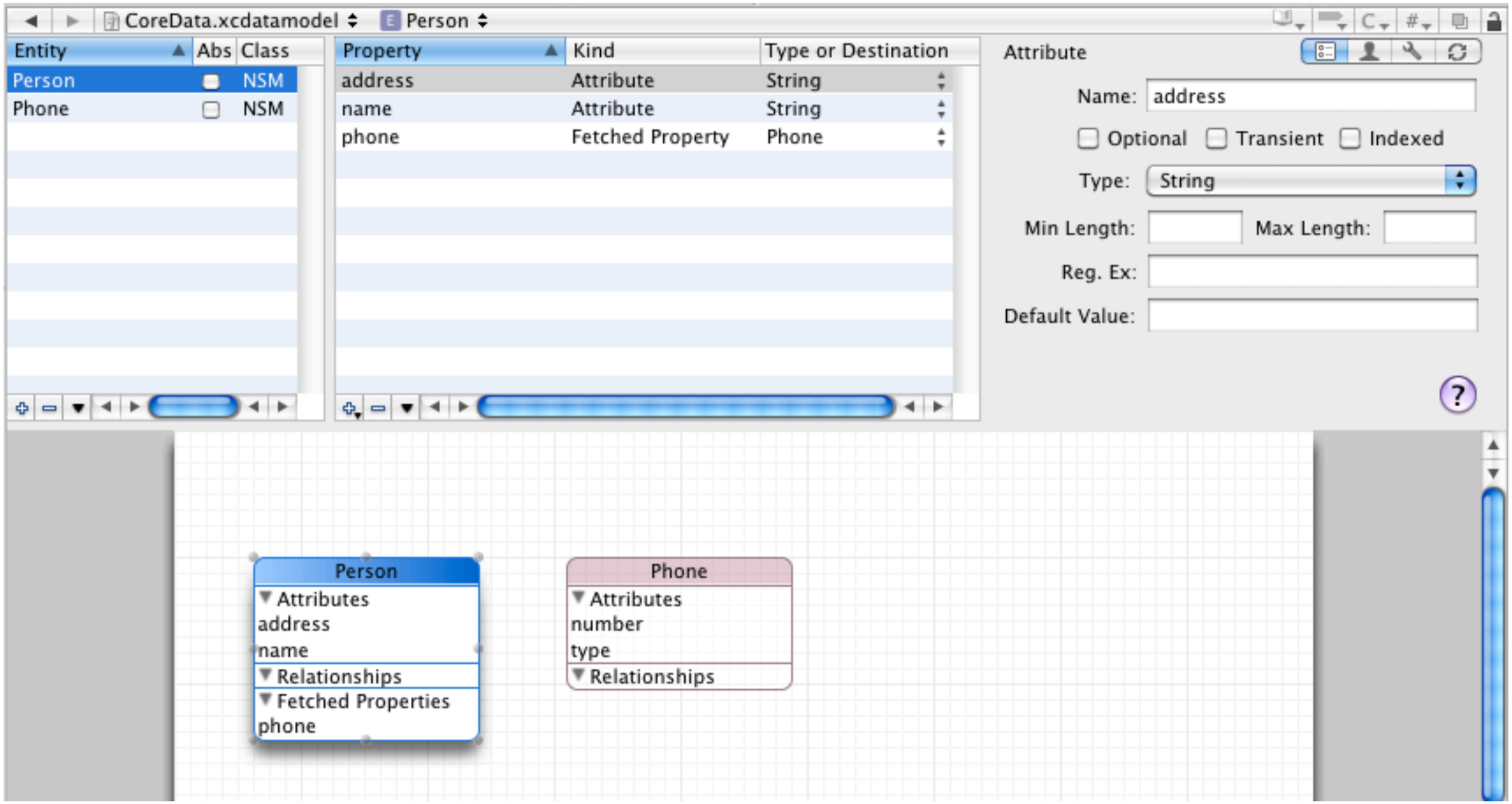
Each entity has a class

Default is NSManagedObject

If need behavior subclass NSManagedObject

Xcode will generate classes for you with accessors

Defining Entities



Terms

Managed object

- Instance of an entity

Persistent (object) store

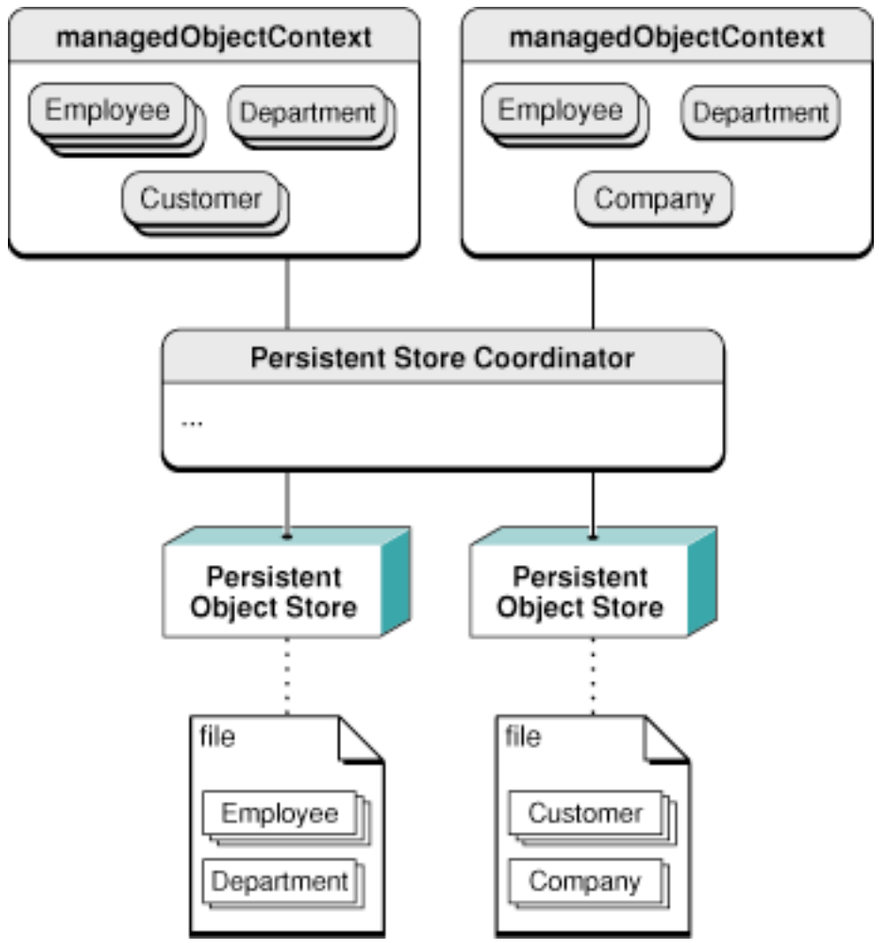
- Has file to store objects
- Maps between objects & file storage
- File storage type can be changed

Persistent store coordinator

- Makes multiple stores appear as one store

Fetch requests

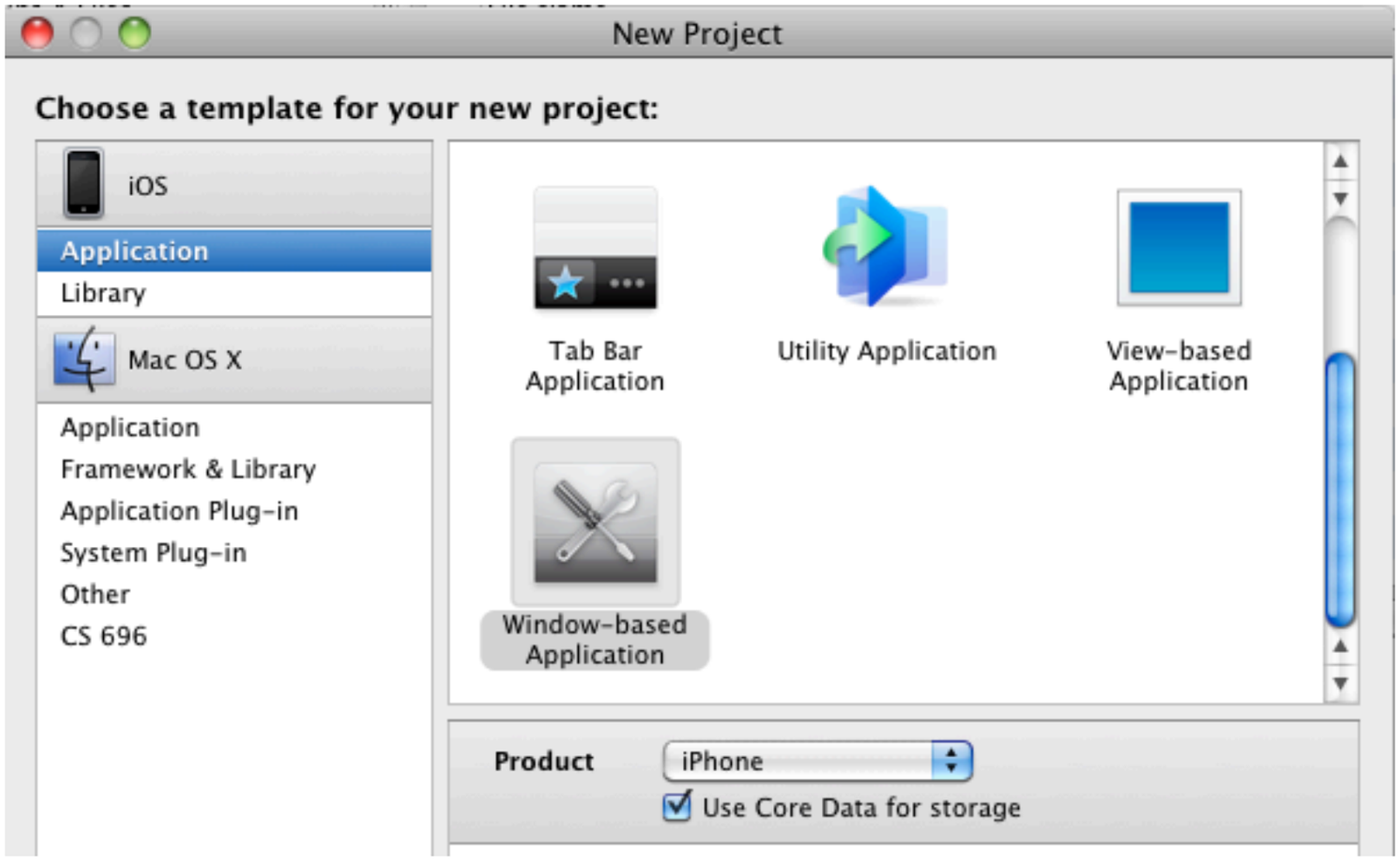
- How to request objects from a persistent store



Terms

- Managed object context
  - Intelligent scratch pad
  - Keeps fetched objects
  - Track changes for undo
  - Changes temporary until objects saved
  - Validates object state before saving

Configuring Project for Core Data



What we get in App Delegate

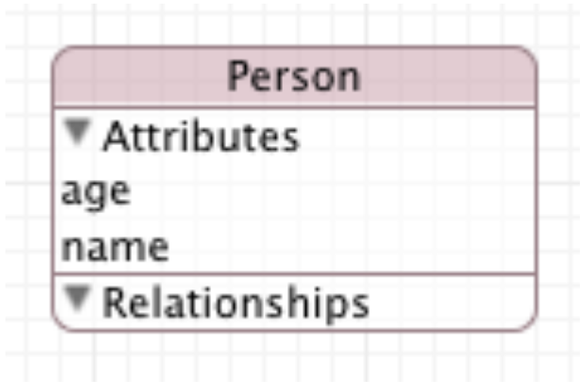
Auto-generated code to access

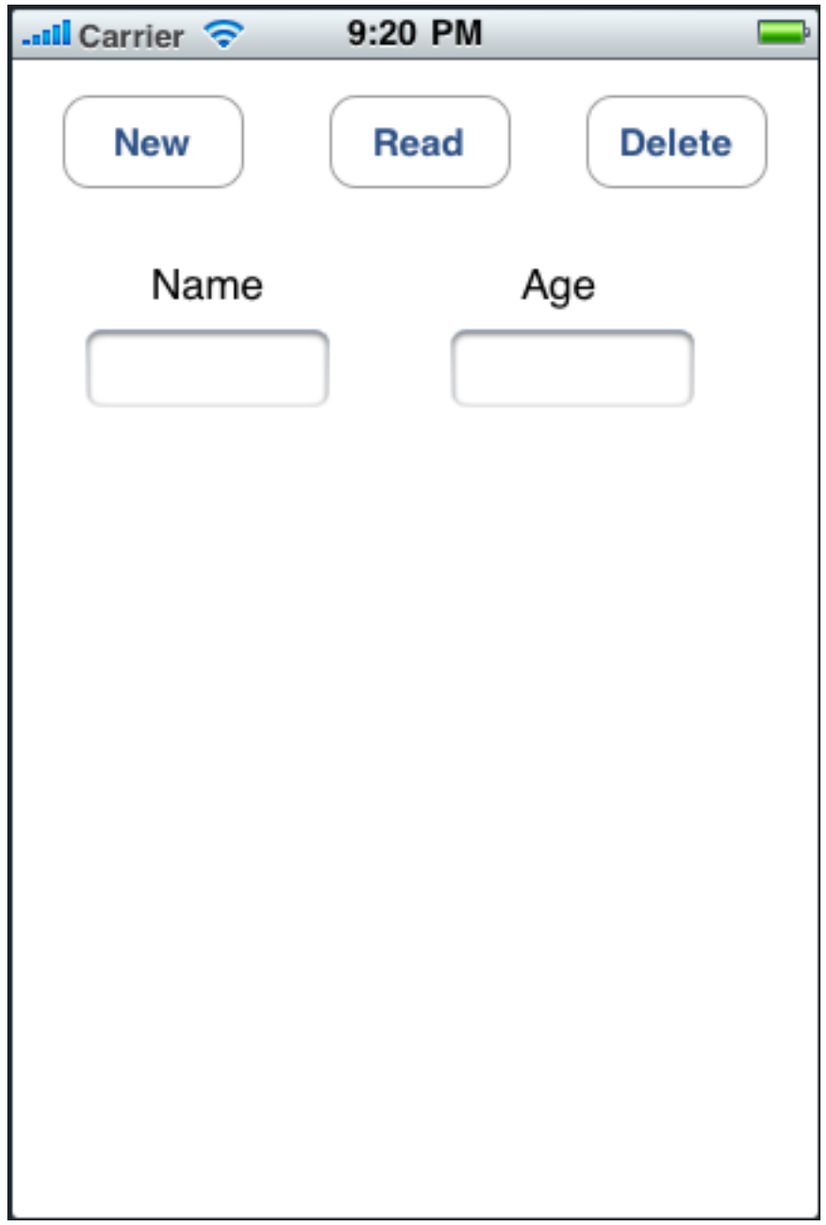
- NSManagedObjectContext
- NSManagedObjectModel
- NSPersistentStoreCoordinator

What we get in App Delegate

```
@interface YourClassName : NSObject <UIApplicationDelegate> {

    UIWindow *window;

@private
    NSManagedObjectContext *managedObjectContext_;
    NSManagedObjectModel *managedObjectModel_;
    NSPersistentStoreCoordinator *persistentStoreCoordinator_;
}

@property (nonatomic, retain) IBOutlet UIWindow *window;

@property (nonatomic, retain, readonly) NSManagedObjectContext *managedObjectContext;
@property (nonatomic, retain, readonly) NSManagedObjectModel *managedObjectModel;
@property (nonatomic, retain, readonly) NSPersistentStoreCoordinator
            *persistentStoreCoordinator;
```

Example





Person Entity

age - int

name - string

Create & Save

Read

Delete

Auto generated Class for Person Entity

```
#import <CoreData/CoreData.h>


@interface Person :  NSManagedObject
{
}

@property (nonatomic, retain) NSString * name;
@property (nonatomic, retain) NSNumber * age;

@end
```

```
#import "Person.h"


@implementation Person

@dynamic name;
@dynamic age;

@end
```

Class for Entity

Can treat entity as a class

Allows us to use standard accessor methods

```
person.name = @"Roger";
[person setName: @"Roger";
```

rather than key-value access

```
[person setValue: @"Roger forKey: @"name"];
```

Property access is slightly faster

Can add logic to class

Creating new person in controller

```
- (IBAction) newPerson {

    NSManagedObjectContext *context = [self managedObjectContext];
    Person *person = (Person *)[NSEntityDescription
                                insertNewObjectForEntityForName:@"Person"
                                inManagedObjectContext:context];

    person.name = name.text;
    person.age = [NSNumber numberWithInt:  [age.text integerValue]];

    NSError *error;
    if (![context save:&error]) {
        NSLog(@"Error on save: %@", error);
    }
}
```

Int Attributes

```
person.age = [NSNumber numberWithInt:  [age.text integerValue]];
```

Stored as NSNumber

```
- (NSManagedObjectContext *) managedObjectContext {
        Core_Data_PersistenceAppDelegate *appDelegate =
            [[UIApplication sharedApplication] delegate];
        NSManagedObjectContext *context = [appDelegate managedObjectContext];
        return context;
}
```

Reading all

```
- (IBAction) readPerson {
    NSManagedObjectContext *context = [self managedObjectContext];
    NSFetchRequest *request = [[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription entityForName:@"Person"
                                   inManagedObjectContext:context];
    [request setEntity:entity];

    NSError *error = nil;
    NSMutableArray *people = [[context executeFetchRequest:request error:&error]
                              mutableCopy];
    if (people == nil) {
        // Handle the error.
    }
    Person * first = [people objectAtIndex:0];
    name.text = first.name;
    age.text = [first.age stringValue];
}
```

Delete

```
NSManagedObjectContext *context = [self managedObjectContext];

Person      * toDelete = code to select person to delete;

[context deleteObject:toDelete];

NSError *error;
if (![context save:&error]) {
      NSLog(@"Error on save: %@", error);
}
```

Changing data

```
NSManagedObjectContext *context = [self managedObjectContext];

Person   * toChange = code to get person to change;
toChange.name = @"mud";

NSError *error;
if (![context save:&error]) {
      NSLog(@"Error on save: %@", error);
}
```

Undo

```
NSManagedObjectContext *context = [self managedObjectContext];
Person   * a = code to get a;
Person   * b = get b;
a.name = @"foo";
b.name = @"bar";
a.name = @"foofoo";
[context undo];
[context redo];
[context undo];
[context rollback];
[context redo];
```

a	b
foo	b
foo	bar
foofoo	bar
a	b
foofoo	bar
a	b
a	b
foofoo	bar

Fine grain undo

Need NSUndoManager

NSUndoManager has undo groups (regions)

undo

- ends the current undo group
- Undoes all operations in previous undo group
- Places group in redo stack

redo

- Does all operations in the top undo group in redo stack

rollback

- Removed everything from undo stack
- Updates objects to their last commited (saved) state

In App Delegate

```
- (NSManagedObjectContext *) managedObjectContext {

    if (managedObjectContext != nil) {
        return managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoordinator];
    if (coordinator != nil) {
        managedObjectContext = [[NSManagedObjectContext alloc] init];

            NSUndoManager *anUndoManager = [[NSUndoManager  alloc] init];
        [managedObjectContext setUndoManager:anUndoManager];
        [anUndoManager release];

        [managedObjectContext setPersistentStoreCoordinator: coordinator];
    }
    return managedObjectContext;
}
```


in iOS there is no default NSUndoManager

Fine grain undo

```
NSManagedObjectContext *context = [self managedObjectContext];
Person   * a = code to get a;
Person   * b = code to get b;                    //a.name ==a, b.name == b
NSUndoManager * undoManager = [context undoManager];
a.name = @"foo";                                 //a.name==foo, b.name== b
[undoManager endUndoGrouping];
[undoManager beginUndoGrouping];
b.name = @"bar";                                 //a.name==foo, b.name== bar
[undoManager endUndoGrouping];
[undoManager beginUndoGrouping];
a.name = @"foofoo";                              //a.name==foofoo, b.name== bar
[undoManager endUndoGrouping];
[context undo];                                  //a.name==foo, b.name==bar
[context undo];                                  //a.name==foo, b.name==b
[context undo];                                  //a.name==a,    b.name==b
[context redo];                                  //a.name==foo, b.name==b
[context redo];                                  //a.name==foo, b.name==bar
[undoManager beginUndoGrouping];
[context rollback];                              //a.name==a,    b.name==b
```

Making Queries

Use NSPredicate with string queries

Generate queries directly using

- NSComparisonPredicate
- NSCompoundPredicate

Sample Query with order

```
NSManagedObjectContext *context = [self managedObjectContext];
    NSFetchRequest *request = [[NSFetchRequest alloc] init];
    NSEntityDescription *entity = [NSEntityDescription entityForName:@"Person"
                                   inManagedObjectContext:context];
    [request setEntity:entity];

    NSNumber *minumumAge = [NSNumber numberWithInt:5];
    NSPredicate *selectOldRoger = [NSPredicate predicateWithFormat:
                                   @"(name LIKE[c] 'Roger') AND (age > %@)", minumumAge];
    [request setPredicate:selectOldRoger];

    NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWithKey:@"age" ascending:NO];
    NSArray *sortDescriptors = [[NSArray alloc] initWithObjects:sortDescriptor, nil];
    [request setSortDescriptors:sortDescriptors];
    [sortDescriptor release];
    [sortDescriptors release];

    NSError *error = nil;
    NSMutableArray *people = [[context executeFetchRequest:request error:&error] mutableCopy];
    if (people == nil) {
        // Handle the error.
    }
```

Query String format

Basic Comparisons

=,==

>=,=>

<=, =<

>

<

!=, <>

BETWEEN

Compound Predicates

AND, &&

OR, ||

NOT, !

Aggregate Operations

AND, SOME

ALL

NONE

IN

array[index]

array[FIRST]

array[LAST]

String Comparisons

BEGINSWITH

CONTAINS

ENDSWITH

LIKE

*, ? - wildcard characters

MATCHES

regular expressions

See Predicate Programming Guide chapter 6 Predicate Format String Syntax for more details

Managed Object IDs and URIs

Each object has unique ID
ID is temporay until object is saved

Can be used to read the object
Useful in restoring state of application

```
NSManagedObjectID *moID = [managedObject objectID];
```

```
[context objectWithID: moID]
```

Versions & Migrations

- Core Data Supports
  - Different versions of your data
  - Migrating from old version of data to new version