CORE DATA

Core Data

- Schema-driven object graph management
- Uses SQLite backing store
- Provides visual mapping and object-oriented API

Core Data Terminology

- Entity
  - Table in database
  - Object in code
- Attributes
  - Column in database
  - Property in code
- Managed object
  - Row/record in database
  - Instance of object in code

Adding Data Model

- Create new project and select "Use Core Data for storage" option
  - Automatically generates a lot of code needed in order to use Core Data
- Alternatively, add a data model to project by going to New File and choosing Data Model
  - Not recommended
- Generates a .xcdatamodel file
  - Like a xib file, provides visual abstraction of database objects, and maps to objects in code

Core Data Stack

- Persistent store coordinator
  - Single point of contact to connect managed object context to backing store
- Managed object model
  - Database schema definitions
- Managed object context
  - Scratch pad for objects
  - Where developer typically interacts with core data
  - NSManagedObjectContext

Core Data Schema

- Defined in visual editor of .xcdatamodel
- Click ✚ below Entity window to add a new Entity (table)
  - Enter a name in configuration window on right
- Click ✚ below Property window and choose "Add Attribute" to add an attribute (column) to Entity
  - Enter name for attribute
  - Choose Type
- To add a relationship, select desired Entity and click ✚ below Property window and choose "Add Relationship"
  - Set destination to Entity this relationship should link to
  - Check "To-Many Relationship" if appropriate
  - Add relationship to linked Entity, and choose initial relationship in "Inverse" field
  - Set "Delete Rule" for desired operation when removing records

Core Data

- Obtain NSManagedObjectContext from delegate @property managedObjectContext
- To create new object in database

```
NSManagedObject *record = [NSEntityDescription
  insertNewObjectForEntityForName:@"EntityName"
  inManagedObjectContext:(NSManagedObjectContext
  *)ctxt];
```

- To access attribute values

```
- (id)valueForKey:(NSString *)key
- (void)setValue:(id)value forKey:(NSString *)key
```

  - Key is Attribute name (column)

Core Data

- Changes to managedObjectContext not saved to persistent storage until *save* is called

```
- (BOOL)save:(NSError **)errors;
```

- Check for changes using

```
- (BOOL)hasChanges;
```

```
if ([context hasChanges] && ![context save:&error])
    . . . // check error
```

Subclass NSManagedObject

- Provide @property for each attribute
  - Simplify set/get methods
- Auto-generate subclass objects using xcode
  - In data model editor, choose any Entity
  - Go to New File and select Managed Object Class
  - Choose Next at next screen, then select desired Entities (leave checkmarks at bottom)
  - Creates header and implementation files for each Entity
  - For individual attribute changes, highlight attribute and select Design->Data Model->Copy Obj-C 2.0 . . .
- Uses @dynamic rather than @synthesize

Fetching Data

- Query using NSFetchRequest
- Four primary components to NSFetchRequest
  - NSEntityDescription
    - Which table to use (required)
  - NSPredicate
    - Which records in table (optional, default is all)
  - NSSortDescriptor
    - Order of returned items (optional, default is random)
  - Batch size and maximum size (optional, default is all)

NSFetchRequest

- Example of fetch request

```
NSFetchRequest *request = [[NSFetchRequest alloc] init];

request.entity = [NSEntityDescription entityForName:@”entityName”
                                inManagedObjectContext:ctxt];
request.fetchBatchSize = 20;
request.fetchLimit = 100;
request.sortDescriptors = [NSArray arrayWithObject:sortDescriptor];
request.predicate = . . . ;
```

Fetch Requests

- NSSortDescriptor

```
NSSortDescriptor *sortDescriptor =
                                [NSSortDescriptor alloc]
                                initWithKey:@”attributeName”
                                ascending:YES
                                selector:@selector(. . .)];
```

- Selector is optional, default is compare:

Fetch Requests

- NSPredicate
  - Query for which records to include

NSPredicate *predicate = [NSPredicate predicateWithFormat:]

- Potential formats

@"uniqueId == %@", [record objectForKey:@"id"]
@"%@ **in** tags", (NSManagedObject *) // tags is to-many
@"viewed **>** %@", (NSDate *)
@"name **contains** %@", (NSString *)

- NSCompoundPredicate
  - AND and OR compounds

- Developer.apple.com/mac/library/documentation/cocoa/conceptual/Predicates/Articles/pSyntax.html

FetchRequests

- Perform fetch using

```
NSError *error = nil;
NSArray *results = [managedObjectContext
                        executeFetchRequest:request
                        error:&error];
```

- Returns nil on error
- Returns empty array if no matches
- Array of NSManagedObjects (or subclass)

NSFetchedResultsController

- Connect NSFetchRequest to UITableView to display records of query
- Provides helper functions for tableview

  [fetchedResultsController sections]
- How to create controller

```
NSFetchedResultsController *frc =
                [[NSFetchedResultsController alloc]
                initWithFetchRequest:(NSFetchRequest *)request
        managedObjectContext:(NSManagedObjectContext *)ctxt
                sectionNameKeyPath:(NSString *)attributeName
                cacheName:(NSString *)cache];
```

SQLite on Android

- Create subclass of SQLiteOpenHelper
  - Provides useful methods for connecting with and setting up database
- Need to override two methods
  - onCreate
    - Called when database created, not on each instantiation of SQLiteOpenHelper class
  - onUpgrade
    - Called when first request is made using a new version than existing database
- Also implement Constructor

```
public DatabaseHelper (Context context) {
  super (context, DBName, cursorFactory, version);
}
```

SQLite on Android

- SQLiteDatabase
  - Provides methods to allow sqlite-type commands
    - execsql()
    - insert()
    - delete()
    - update()
    - query()
    - rawQuery()
- Get SQLiteDatabase from SQLiteOpenHelper
    - getReadableDatabase()
    - getWriteableDatabase()

SQLite on Android

- execSQL can be used to implement any SQL command
  - See SQLite documentation for syntax
    - www.sqlite.org/quickstart.html
- ContentValues may be used for creating new records
  - Key-value pairs, where the key is a string representing the column name

SQLite on Android

- Results sets returned in Cursor object
  - Enumeration
    - moveToNext()
    - moveToFirst()
    - moveToPosition(int)
    - moveToPrevious()
    - moveToLast()
  - Get values
    - getInt(), getShort(), getString(), etc.