



ITM103 iOS Application Development

Topic 8: Persistent Storage



Objectives

- By the end of the lesson, you will be able to:
 - know the collection of tools and frameworks for storing, accessing and sharing data.
 - know the different SQLite function calls
 - be familiar with the SQLite Database Browser

Persistent Storage

Different Forms of Storage

Data Management in iOS

- iOS has a comprehensive collection of tools and frameworks for storing, access and sharing data.
 - **Sandbox**
A set of fine-grained controls that limit the app's access to files, preferences, network resources, hardware.
 - **User Defaults**
Manages the storage of preferences
 - **Core Data**
Full-featured data modelling framework for objet oriented Cocoa Touch applications
 - **SQLite**
Low level relational database

NSUserDefaults

- With **NSUserDefaults** class, you can save settings and properties related to application or user data.
- With NSUserDefaults, you can save objects from the following class types:
 - Int / Float / Double
 - Bool
 - Object (Array, Strings, Dictionaries, NSDate)

NSUserDefaults – Save/Load

- **NSUserDefaults** save key values in pair.
- Save data

```
let defaults = UserDefaults()  
defaults.set(256, forKey: "TriesLeft")  
defaults.synchronize()
```

- Load data.

```
let score = defaults.integer(forKey: "TriesLeft")  
self.scoreLabel.text = "\(score)"
```

Core Data

- Core Data framework, a persistence layer that efficiently handles the coordination of saving/ updating/deleting data and maintaining data integrity.
- With Core data, you define the schema you want your data to conform to and create objects from the schema to perform CRUD operations.
- Core Data is not a database.



Core Data

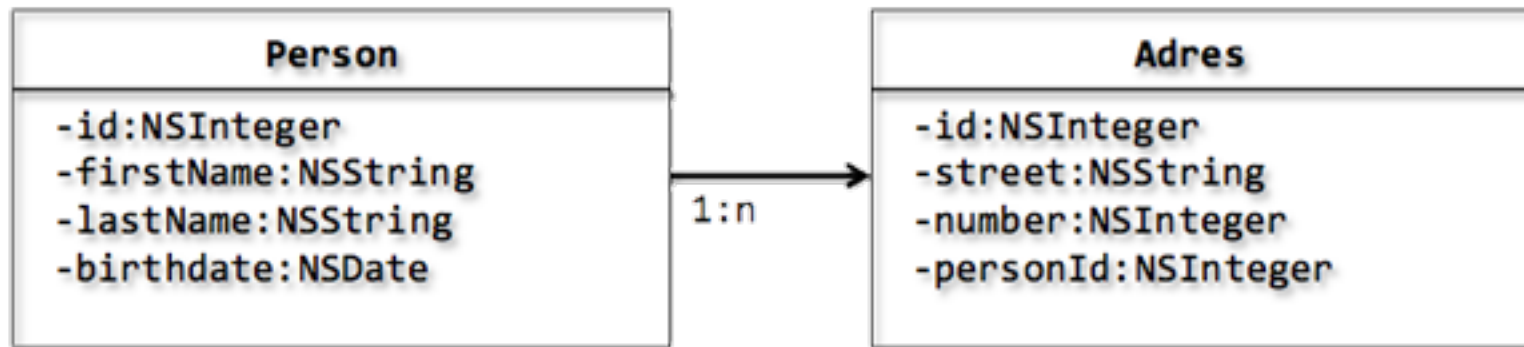
- The **NSManagedObjectModel** is where you define your schema for Core Data.
- In your model, you define the Entities, or classes of data in your schema.
- Within your Entities, you set their Attributes, which are the details of your data.
- Finally, you can link Entities together through Relationships
- If you wish to find out more, you can refer to this Core Data Tutorial online -
<http://www.raywenderlich.com/115695/getting-started-with-core-data-tutorial>

Persistent Storage

SQLite

SQLite

- An embedded implementation of SQL.
- SQL stands for Structured Query Language and is a standard language for relational databases.
- SQLite follows the principals of Relational Database Management System (RDBMS)

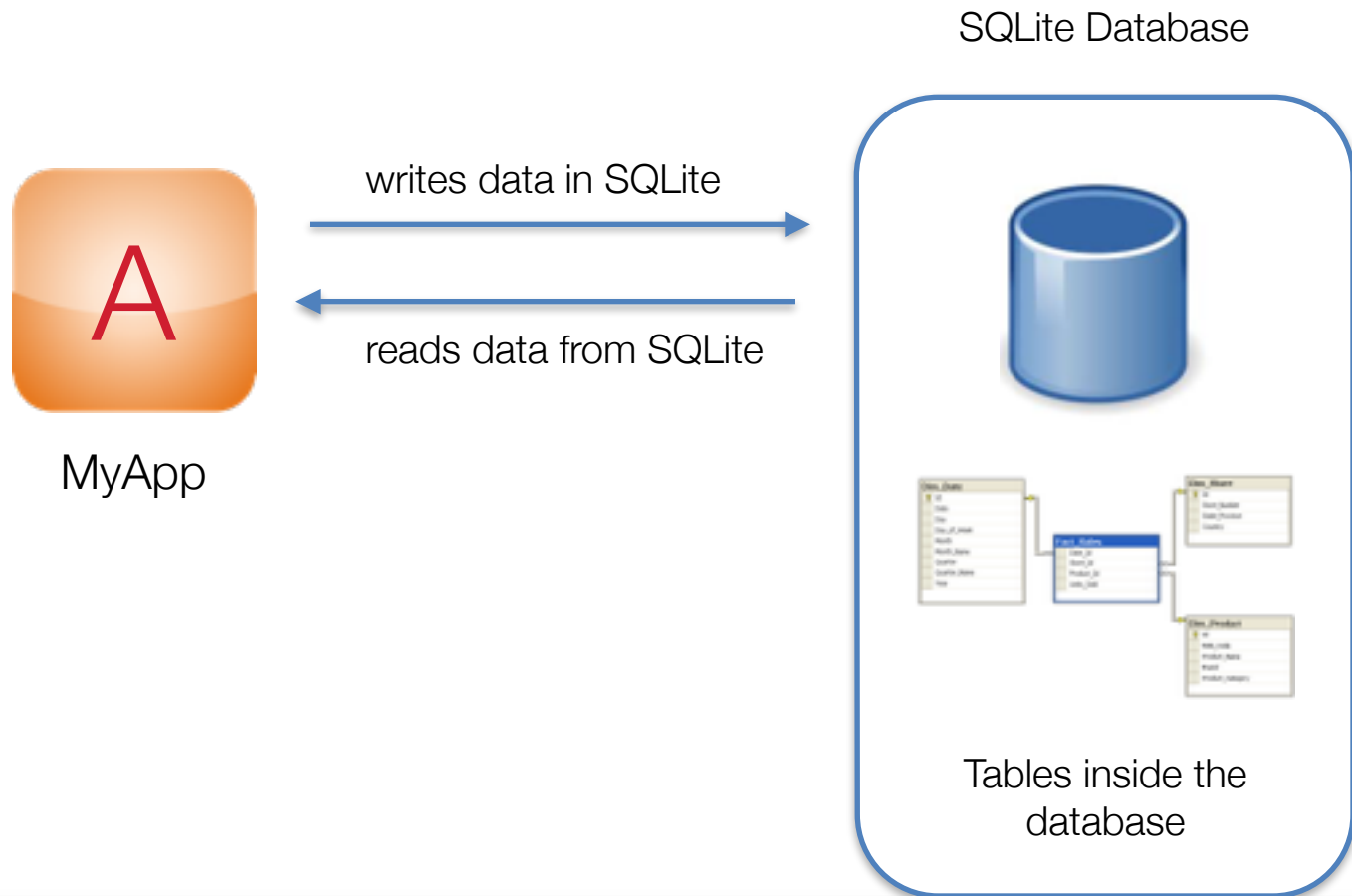


SQLite

- SQLite is the **database engine** used in iOS application:
 - Software library
 - Self-contained, server-less, zero-configuration
 - Transactional
 - Written in C & Objective-C
 - Can be incorporated into Swift projects

SQLite

- SQLite is the **database engine** used in iOS application:



SQLite Functions

- **Data Types:**

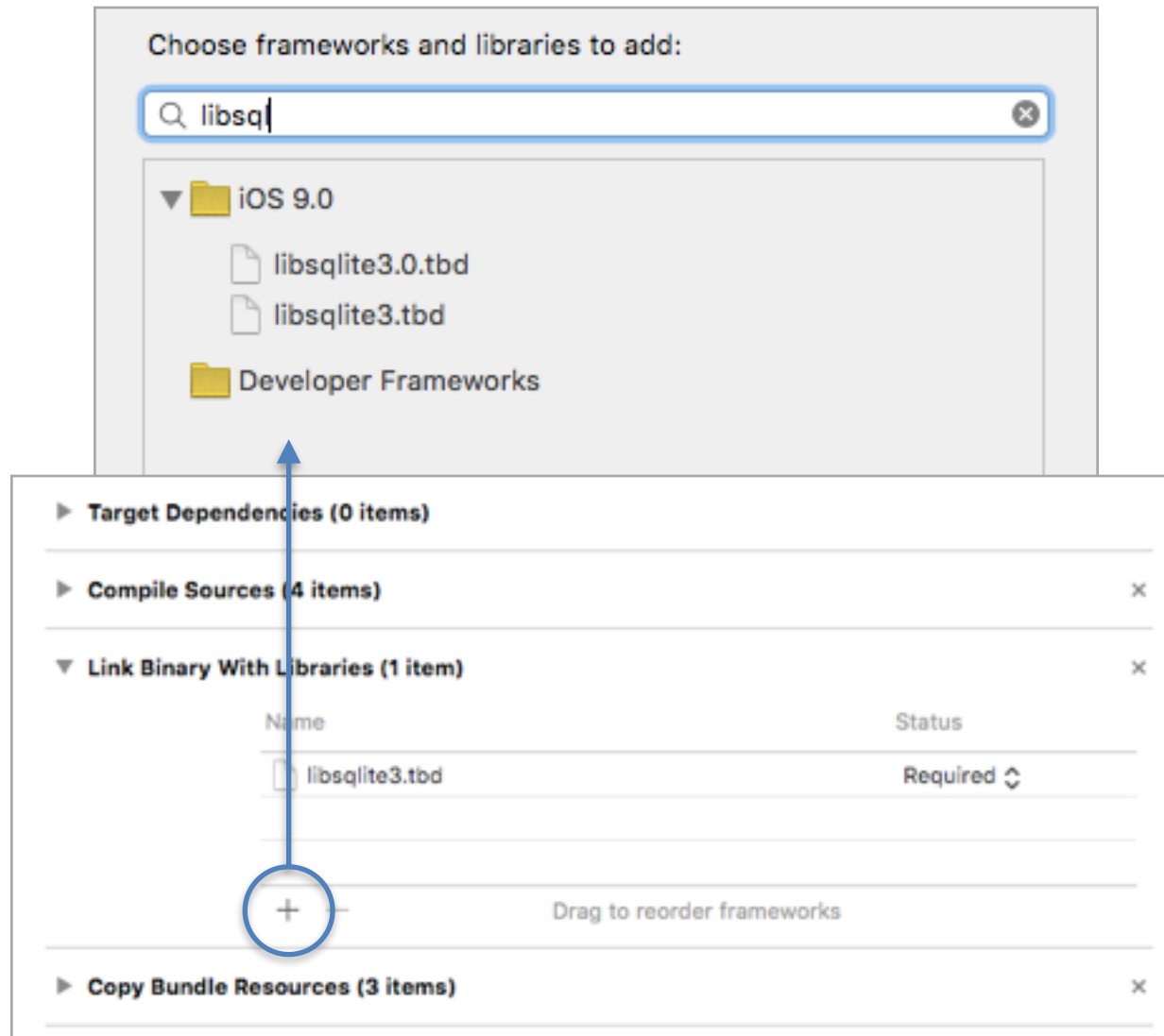
- Null null values
- INTEGER whole numbers, 1, 2, 3, 4, etc.
- REAL numbers with decimal places
- TEXT strings using UTF-8/16 encoding
do not need to specify length
- BLOB binary data / images / etc

<https://www.sqlite.org/datatype3.html>

Persistent Storage


Including SQLite Library into Xcode

libsqlite3.tbd Framework



Create bridging header

- SQLite 3 is a Objective-C library
- To use it in Swift, you need a bridging header.
- To create a bridging header:
 - Add a new Header file in Objective-C.
 - Add the path to the header file in your project Build Settings
 - Then add the `#include "sqlite3.h"` in the bridging header file.

 DataStorageApp-Bridging-Header.h

```
//  
// Use this file to import your target's public headers that you would like to expose to Swift.  
//  
  
#include "sqlite3.h"  
|
```


Create data layers

- A multi-tier design in your app is recommended:
 - Easier to write
 - Easier to maintain
- To create a multi-tier design:
 - Create a simple SQLite wrapper
 - Create a Data layer for data access
 - Create a Logic layer for user specific logic (optional)

Persistent Storage

SQLite Wrapper

SQLite Functions

- For this module, we will use an modified version of an open source wrapper for SQLite.
- Original available at:
<https://github.com/FahimF/SQLiteDB>
- Modified version available at:
<http://learn.nyp.edu.sg>

SQLite Functions

- **Initialise** the SQLite database:

```
let db = SQLiteDatabase.sharedInstance()
```

Creates and opens an empty database of the filename “data.db”.

If the database already exists, it will only open the database.

SQLite Functions

- **Query** the SQL database:

```
db.query(sql: String, parameters:[Any]?)  
-> [[String: Any]]
```

Performs a query on the database

```
let data = db.query(sql: "SELECT * FROM customers WHERE name=?",  
    parameters:["John"])  
  
let row = data[0]  
if let name = row["name"] {  
    textLabel.text = name as! String  
}
```

SQLite Functions

- **Execute** a non-query on the SQL database:
`db.execute(sql: String, parameters:[Any]?)`

Executes a non-query such as INSERT, DELETE, or UPDATE on the database. You can also use this to create tables.

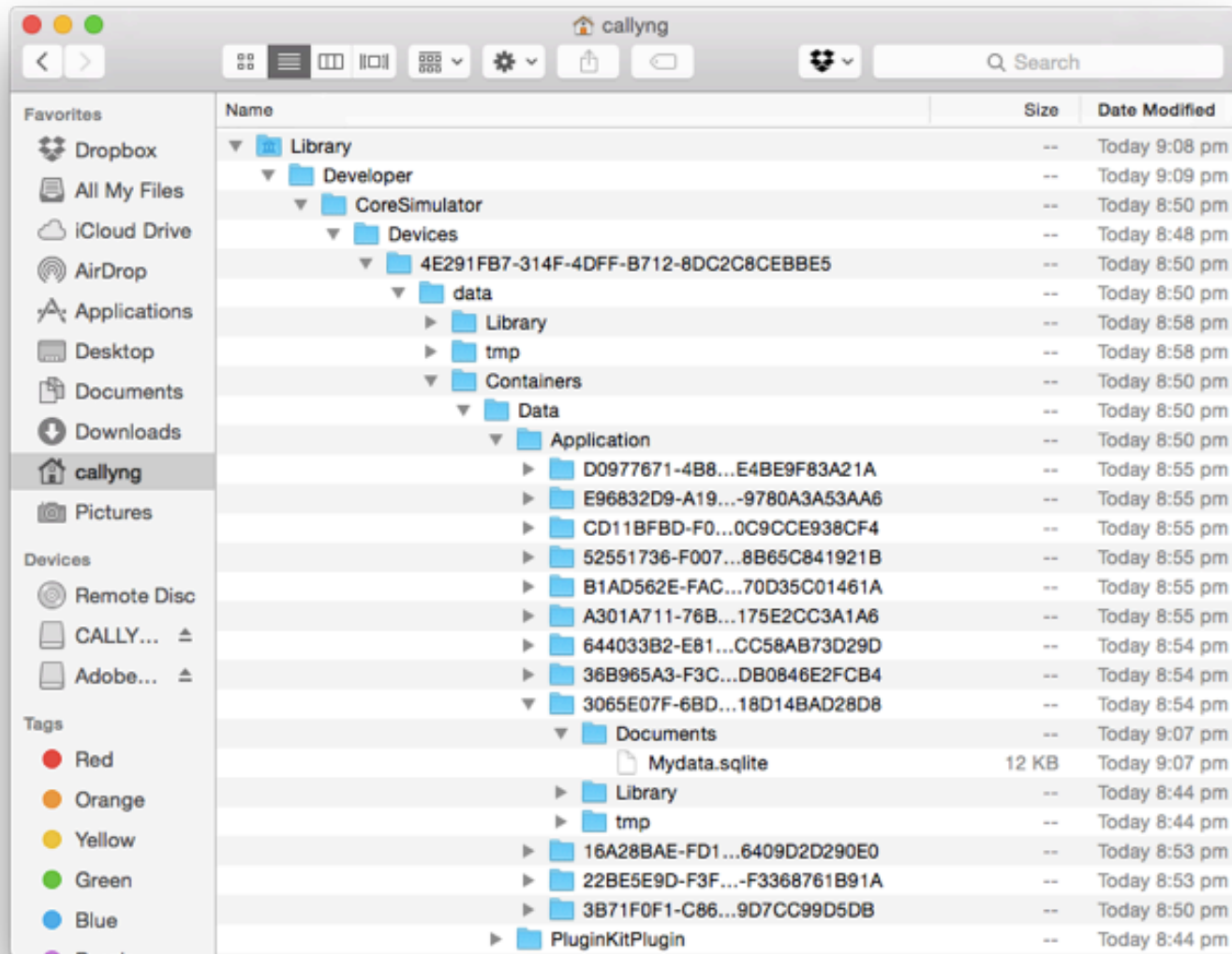
For UPDATE and DELETE, this returns the number of rows affected.
For INSERT, this returns the ID of the record inserted.

```
let result = db.execute(sql:  
    "DELETE FROM customers WHERE last_name='Smith'")
```

Persistent Storage

SQLite Browser for Testing and Debugging

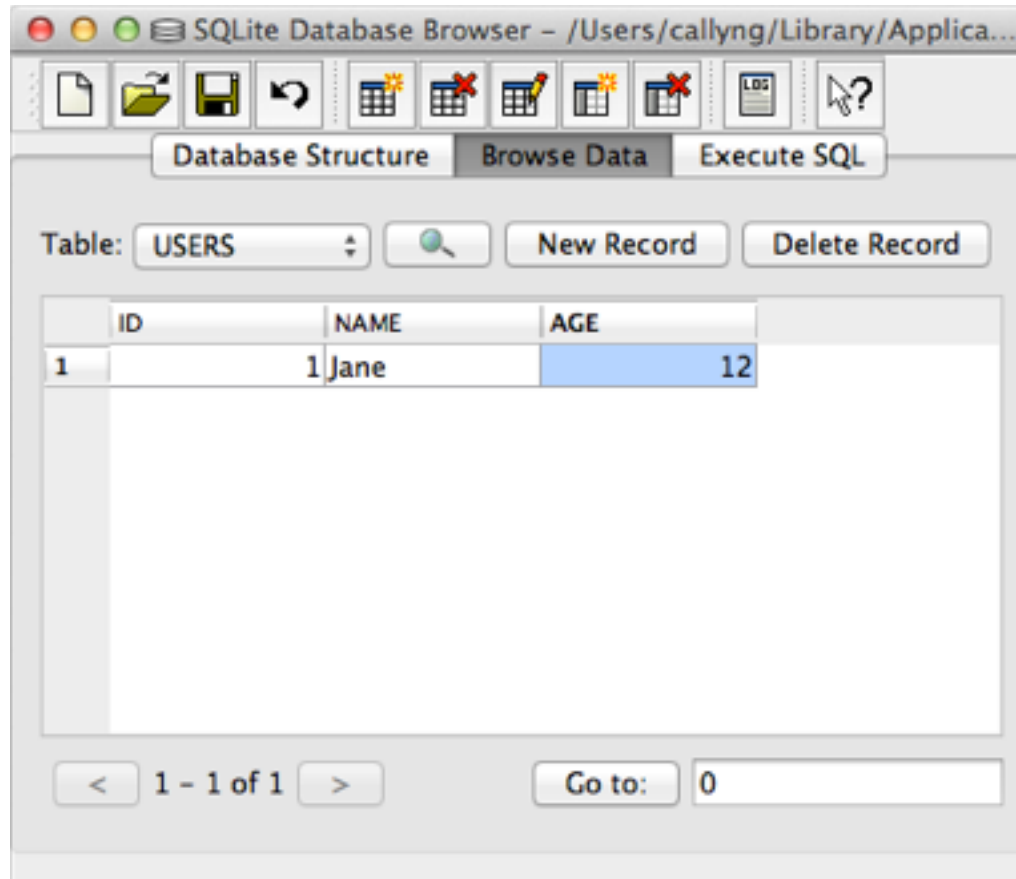
Database File Directory



~/Library/Developer/CoreSimulator/Devices/<Device_ID>/data/Containers/Data/Application/<App_ID>/Documents/

SQLite Database Browser

- Download it from:
 - <http://sqlitebrowser.org>



Summary

- Various forms of data storage:
 - Sandbox
 - User Defaults
 - Core Data
- SQLite
- SQLite Functions Calls
- SQLite Database Browser