CS 696 Mobile Application Development
Fall Semester, 2010
Doc 15 Data
Oct 14, 2010

References

Stanford iPhone Course, Winter 2010, CS193P - Lecture 9

Beginning iPhone 3 Development: Exploring the iPhone SDK by Jeff LaMarche, and David Mark

http://www.sqlite.org/

Local Data

Property Lists, NSUserDefaults & Settings

iPhone files

Archiving Objects

SQLite

Core Data

Property lists

Allowed data types
NSArray
NSDictionary
NSString
NSData
NSDate
NSNumber (int, float, bool)

Formats
XML
Binary

NSArray & NSDictionary Connivence methods

Writing

- (BOOL)writeToFile:(NSString *)aPath atomically:(BOOL)flag;
- (BOOL)writeToURL:(NSURL *)aURL atomically:(BOOL)flag;

Reading

- (id)initWithContentsOfFile:(NSString *)aPath;
- (id)initWithContentsOfURL:(NSURL *)aURL;

Example

```
NSArray * data = [[NSArray alloc] initWithObjects:@"Cat", @"Dog", nil];
[data writeToFile:@"data.plist" atomically:YES];
```

```
NSArray * readData = [[NSArray alloc] initWithContentsOfFile:@"data.plist"];
NSLog(@"%@", readData);
```

NSPropertyListSerialization

File format

More descriptive errors

Mutability

```
+ (NSData *)dataFromPropertyList:(id)plist
    format:(NSPropertyListFormat)format
    errorDescription:(NSString **)errorString;
```

```
+ (id)propertyListFromData:(NSData *)data
    mutabilityOption:(NSPropertyListMutabilityOptions)opt
    format:(NSPropertyListFormat *)format
    errorDescription:(NSString **)errorString;
```

Example

```
NSString *path = [[NSBundle mainBundle] pathForResource:@"data" ofType:@"plist"];
NSArray * data = [[NSArray alloc] initWithObjects:@"Cat", @"Dog", nil];

NSString * error;
NSData * plist = [NSPropertyListSerialization
                        dataFromPropertyList:(id)data
                        format:NSPropertyListXMLFormat_v1_0
                        errorDescription:&error];
if (plist) {
        [plist writeToFile: path atomically:YES];
} else {
        NSLog(@"%@",error);
        [error release];
}
```

Archives

Reads/Writes objects to files

Any object that implement NSCode interface

Supports numeric types

Supports network of objects

Person Example

Two data types to illustrate differences

```
@interface Person : NSObject <NSCoding, NSCopying>{

}
```

```
@property (nonatomic, retain) NSString * name;
@property (nonatomic, assign) int age;
@end
```

NSCoding methods

```
- (void)encodeWithCoder:(NSCoder *)encoder {
      [encoder encodeObject:name forKey:@"name"];
      [encoder encodeInt:age forKey:@"age"];
}

- (id)initWithCoder:(NSCoder *)decoder {
      if (self = [super init]) {
          self.name = [decoder decodeObjectForKey:@"name"];
          self.age = [decoder    decodeIntForKey:@"age"];
      }
      return self;
}
```

NSCopying method

```
- (id)copyWithZone:(NSZone *)zone {
    Person *copy = [[[self class] allocWithZone: zone] init];
    copy.name = [[self.name copyWithZone:zone] autorelease];
    copy.age = self.age;
    return copy;
}
```

Archiving Single object

```
Person * roger = [Person new];
roger.name = @"Roger";
roger.age = 29;

NSMutableData *data = [[NSMutableData alloc] init];
NSKeyedArchiver *archiver =
        [[NSKeyedArchiver alloc] initForWritingWithMutableData:data];

[archiver encodeObject: roger forKey: @"roger"];
[archiver finishEncoding];
[data writeToFile: @"PersonFile" atomically:YES];
[roger release];
[archiver release];
[data release];
```

UnArching Object

```
NSData *data = [[NSMutableData alloc] initWithContentsOfFile: @"PersonFile"];
NSKeyedUnarchiver *unarchiver =
    [[NSKeyedUnarchiver alloc] initForReadingWithData:data];

Person * recoveredPerson = [unarchiver decodeObjectForKey: @"roger"];

[unarchiver finishDecoding];
[unarchiver release];
[data release];
```

Multiple Objects

One archive can contain many different objects

All objects are read from disk when archiving

Class Versions

What happens when class changes

- Adds/removes instance variables

Archiving framework does not provide support for changes

Your code has to deal with changes

New version of class reading data

```
- (id)initWithCoder:(NSCoder *)decoder {
      if (self = [super init]) {
            self.name = [decoder decodeObjectForKey:@"name"];
            self.age = [decoder    decodeIntForKey:@"age"];
     if ([decoder containsValueForKey: @"address"])
          self.address = [decoder decodeObjectForKey:@"address"];
     else
          self.address = nil;
      }
      return self;
}
```

File System

Each App has separate file system

<Application Home>
- AppName.app
  - AppName
  - MainWindow.nib
- Documents
- Library
  - Preferences
  - Caches
- tmp

Can only read/write files in home directory

- Security
- Privacy
- Ease of removing app

Mostly backed up with iTunes sync

Foundation Functions

Functions for

- Assertions
- Byte Ordering
- Exception handling
- Logging output (NSLog)
- Managing File Paths
- etc

File Foundation Functions

```
NSString *homePath = NSHomeDirectory();
NSString *tmpPath = NSTemporaryDirectory();
```

File Foundation Functions

```
NSArray * NSSearchPathForDirectoriesInDomains (
   NSSearchPathDirectory directory,
   NSSearchPathDomainMask domainMask,
   BOOL expandTilde
);
```

```
NSArray *paths = NSSearchPathForDirectoriesInDomains
        (NSDocumentDirectory, NSUserDomainMask, YES);
NSString *documentsPath = [paths objectAtIndex:0];
```

```
NSString *fooPath =
    [documentsPath stringByAppendingPathComponent:@“foo.plist”];
```

NSFileManager

More file operations

- Create files/directories
- Move files
- Copy files
- Directory contents

Writeable Files & Nib

Files included in the nib are read only

If you want the file to be read/write copy it to users documents directory

Chech on startup if file has been copied

SQLite http://www.sqlite.org/about.html

Free/Open source

Embedded in application

Compact, fast, reliable

Storage Classes

NULL

INTEGER
: Signed integer in 1, 2, 3, 4, 6, or 8 bytes

REAL
: 8-byte IEEE floating point number

TEXT
: Text stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE)

BLOB
: Stored as it is entered into the database

Dynamic Typing

Any column may be used to store a value of any storage class
except an INTEGER PRIMARY KEY column

So what happens when you insert text into an integer column?

Each column has its preferred datatype (affinity)

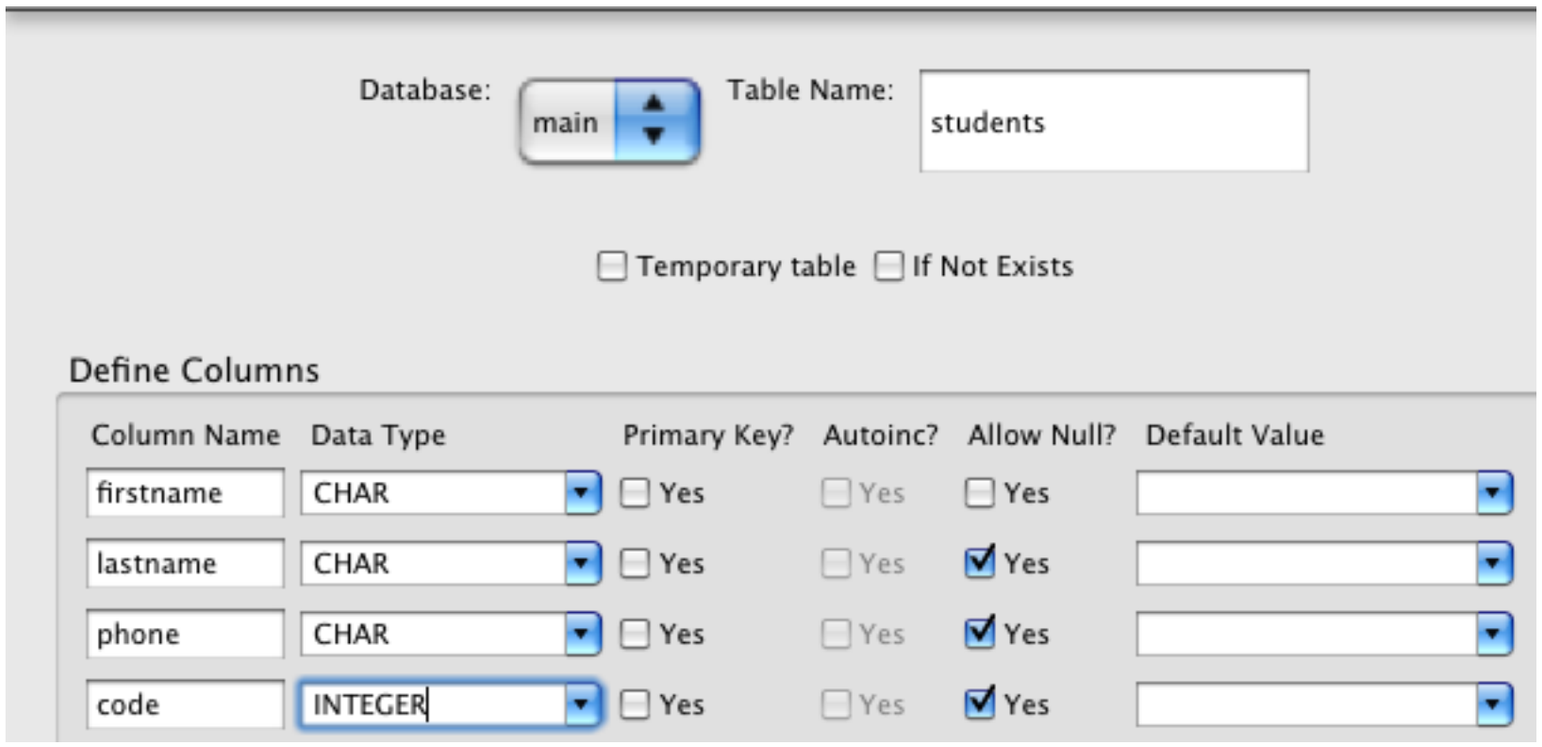
if data can be converted losslessly

SQLite stores data using the column preferred datatype

else stores type as it is

SQLType	SQLite Affinity
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT

SQLType	SQLite Affinity
BLOB no datatype specified	NONE
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC



CREATE TABLE "main"."students" ("firstname" CHAR NOT NULL ,
"lastname" CHAR, "phone" CHAR, "code" INTEGER)

https://addons.mozilla.org/en-US/firefox/addon/5817/

SELECT	Retrieves data from table(s)
INSERT	Adds row(s) to a table
UPDATE	Changes field(s) in record(s)
DELETE	Removes row(s) from a table Data Definition
CREATE TABLE	Define a table and its columns(fields)
DROP TABLE	Deletes a table
ALTER TABLE	Adds a new column, add/drop primary key
CREATE INDEX	Create an index
DROP INDEX	Deletes an index
CREATE VIEW	Define a logical table from other table(s)/view(s)
DROP VIEW	Deletes a view

SQL is not case sensitive

```
CREATE TABLE SampleTable (
    name text UNIQUE,
    age integer,
    isStudent boolean,
    description
)
```

name	age	isStudent	description

```
insert into SampleTable values(
    'Donald Knuth',
    72,
    0,
    'Computer Science deity'
)
```

```
select * from SampleTable
```

name	age	isStudent	description
	72	0	

General Form

```
CREATE TABLE table_name (
    col_name  col_type [ NOT NULL | PRIMARY KEY]
    [, col_name col_type [ NOT NULL | PRIMARY KEY]]*
)
```

Example

```
CREATE TABLE students
        (
        firstname  CHAR(20) NOT NULL,
        lastname   CHAR(20),
        phone      CHAR(10),
        code       INTEGER
        )
```

```
CREATE TABLE codes
        (
        code       INTEGER,
        name       CHAR(20)
        )
```

Add data to a table

General Form

INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
[INTO] tbl_name [(col_name,...)]
VALUES ((expression | DEFAULT),...),(...),...
[ON DUPLICATE KEY UPDATE col_name=expression, ...]

Gets data from one or more tables

General Form

```
SELECT [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
    [SQL_BUFFER_RESULT] [SQL_CACHE | SQL_NO_CACHE]
    [SQL_CALC_FOUND_ROWS] [HIGH_PRIORITY]
    [DISTINCT | DISTINCTROW | ALL]
  select_expression,...
  [INTO {OUTFILE | DUMPFILE} 'file_name' export_options]
  [FROM table_references
   [WHERE where_definition]
   [GROUP BY {unsigned_integer | col_name | formula} [ASC | DESC], ...
     [WITH ROLLUP]]
   [HAVING where_definition]
   [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] ,...]
   [LIMIT [offset,] row_count | row_count OFFSET offset]
   [PROCEDURE procedure_name(argument_list)]
   [FOR UPDATE | LOCK IN SHARE MODE]]
```

```
INSERT
        INTO students (firstname, lastname, phone, code)
        VALUES ('Roger', 'Whitney', '594-3535', 2000 )

INSERT
        INTO codes (code, name)
        VALUES (2000, 'marginal' )
```

```
SELECT * FROM students;
+-----------+----------+----------+------+
| firstname | lastname | phone    | code |
+-----------+----------+----------+------+
| Roger     | Whitney  | 594-3535 | 2000 |
+-----------+----------+----------+------+
```

```
SELECT firstname , phone FROM students
+-----------+----------+
| firstname | phone    |
+-----------+----------+
| Roger     | 594-3535 |
+-----------+----------+
1 row in set (0.00 sec)
```

```
SELECT lastname, name
        FROM students, codes
        WHERE students.code = codes.code

+----------+----------+
| lastname | name     |
+----------+----------+
| Whitney  | marginal |
+----------+----------+
1 row in set (0.00 sec)
```

```
SELECT students.lastname, codes.name
        FROM students, codes
        WHERE students.code = codes.code

+----------+----------+
| lastname | name     |
+----------+----------+
| Whitney  | marginal |
+----------+----------+
1 row in set (0.00 sec)
```

Modify existing data in a database

General Form

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name [, tbl_name ...]
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
```

```
UPDATE students
        SET firstname='Sam'
        WHERE lastname='Whitney'
```

ALTER TABLE students ADD column foo CHAR(40);

DROP TABLE students;

name	faculty_id
Whitney	1
Beck	2
Anantha	3

CREATE TABLE "faculty" ("name" VARCHAR NOT NULL , "faculty_id" INTEGER PRIMARY KEY AUTOINCREMENT)

Indices make accessing faster

Primary keys automatically have an index

The CREATE INDEX command creates indices

```
CREATE INDEX faculty_name_key on faculty (name);
```

```
INSERT INTO  faculty ( name) VALUES    ('Whitney')
INSERT INTO  faculty ( name) VALUES    ('Beck')
INSERT INTO  faculty ( name) VALUES    ('Lewis')
INSERT INTO faculty ( name) VALUES ('Eckberg')

select * from faculty;
```

Result

```
          name           | faculty_id
-------------------------+------------
 Whitney                 |          1
 Beck                    |          2
 Lewis                   |          3
 Eckberg                 |          4
(4 rows)
```

start_time	end_time	day	faculty_id	office_hour_id
10:00	11:00	Wed	1	1
8:00	12:00	Mon	2	2
17:00	18:30	Tue	1	3
9:00	10:30	Tue	3	4
9:00	10:30	Thu	3	5
15:00	16:00	Fri	1	6

name	faculty_id
Whitney	1
Beck	2
Lewis	3
Eckberg	4

```
CREATE TABLE "office_hours" (
    "start_time" TEXT NOT NULL ,
    "end_time" TEXT NOT NULL ,
    "day" TEXT NOT NULL ,
    "faculty_id" INTEGER NOT NULL  check(typeof("faculty_id") = 'integer') , "office_hour_id"
    INTEGER PRIMARY KEY  AUTOINCREMENT  NOT NULL
    )
```

Simple Insert

```
INSERT
    INTO office_hours ( start_time, end_time, day, faculty_id )
    VALUES ( '10:00:00',  '11:00:00' ,  'Wed',  1 );
```

The problem is that we need to know the id for the faculty

Using Select

```
INSERT INTO
        office_hours  (start_time, end_time, day, faculty_id )
SELECT
        '8:00:00' AS start_time,
        '12:00:00' AS end_time,
        'Mon' AS day,
        faculty_id AS faculty_id
FROM
        faculty
WHERE
        name = 'Beck'
```

```
SELECT
    name, start_time, end_time, day
FROM
    office_hours, faculty
WHERE
    faculty.faculty_id = office_hours.faculty_id;
```

name	start_time	end_time	day
Whitney	10:00:00	11:00:00	Wed
Beck	08:00:00	12:00:00	Mon
Whitney	17:00:00	18:30:00	Tue
Whitney	15:00:00	16:00:00	Fri
Lewis	09:00:00	10:30:00	Tue
Eckberg	09:00:00	10:30:00	Thu

```
SELECT
    name, start_time, end_time, day
FROM
    office_hours, faculty
WHERE
    faculty.faculty_id = office_hours.faculty_id
 AND
    start_time > '09:00:00'
 AND
    end_time < '16:30:00'
ORDER BY
    Name;
```

name	start_time	end_time	day
Whitney	10:00:00	11:00:00	Wed
Whitney	15:00:00	16:00:00	Fri

SQLite C API used on iPhone

http://www.sqlite.org/cintro.html

sqlite3_open

sqlite3_prepare

sqlite3_step

sqlite3_column_NNN

sqlite3_finalize

sqlite3_close

sqlite3_exec

SQLite & Objective C

SQLite is written in C

Does not know about Objective C types

So need to covert between C & Objective C types

Opening database

```
#import <sqlite3.h>

   NSString * databasePath = some path;
      sqlite3 *database;
      const char* databasePathUTF8 = [databasePath UTF8String];
      if(sqlite3_open(databasePathUTF8, &database) == SQLITE_OK) {
            //access database
      }
      sqlite3_close(database);
```


http://dblog.com.au/iphone-development-tutorials/iphone-sdk-tutorial-reading-data-from-a-sqlite-database/

databasePath is path to the database file in the user's iPhone directory

Creating Table

```
char * errorMsg;
const char *createSQL = “CREATE TABLE IF NOT EXISTS PEOPLE
(ID INTEGER PRIMARY KEY AUTOINCREMENT, FIELD_DATA TEXT)”;

int result = sqlite3_exec (database, createSQL, NULL, NULL, &errorMsg);

if (result == SQLITE_OK) {
    // command run successfully
} else {
    //handle error
}
```

sqlite3_exec is used for anythig that does not return data
updates, inserts, and deletes

third argument is a callback function to handle rows, fourth argument is 1st argument for callback

Select

```
NSString *query = @”SELECT ID, FIELD_DATA FROM FIELDS ORDER BY ROW”;
sqlite3_stmt *statement;
int result = (sqlite3_prepare_v2( database, [query UTF8String], -1, &statement, nil);


while (sqlite3_step(statement) == SQLITE_ROW) {
    int rowNum = sqlite3_column_int(statement, 0);
    char *rowData = (char *)sqlite3_column_text(statement, 1);
    NSString *fieldValue = [[NSString alloc] initWithUTF8String:rowData];
    // Do something with the data here
    [fieldValue release];
} sqlite3_finalize(statement);
```

Binding Variables

```
char *sql = “insert into foo values (?, ?);”;
sqlite3_stmt *stmt;
if (sqlite3_prepare_v2(database, sql, -1, &stmt, nil) == SQLITE_OK) {
    sqlite3_bind_int(stmt, 1, 235);
    sqlite3_bind_text(stmt, 2, “Bar”, -1, NULL);
}
if (sqlite3_step(stmt) != SQLITE_DONE)
    NSLog(@”This should be real error checking!”);
sqlite3_finalize(stmt);
```