SQLite Techniques

Ben Scheirman
@subdigital

Code / Slides:
github.com/subdigital/iphonedevcon-sandiego

SQLite is....

- Single-user
- File-based
- Simple
- Cross Platform
- A subset of full ANSI SQL

SQLite API

- Pure C API
- Lots of strange-looking, hard to debug code
- Best choice is to abstract it away

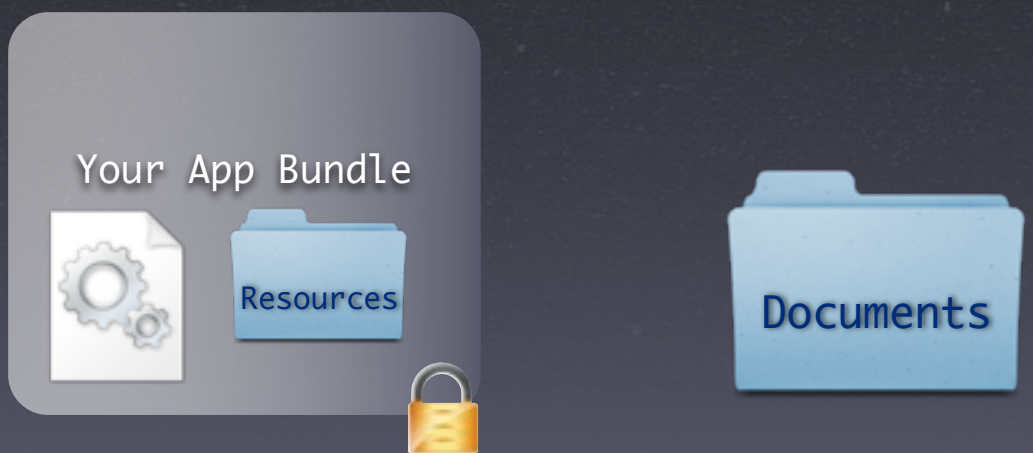
SQLite Tools

- SQLite Manager Add-in for Firefox
- sqlite3 command line utility

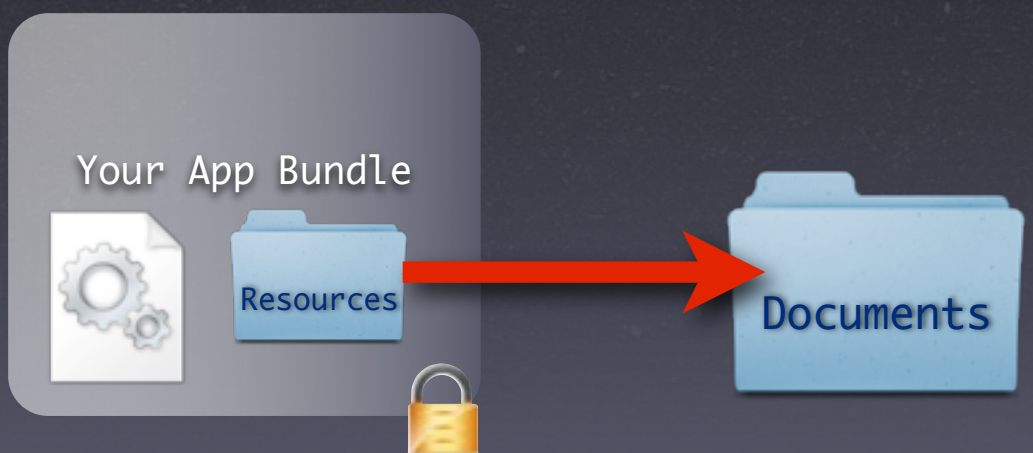
Writable Databases

- Your app bundle is signed!
- (That means the contents can't change)

Your App Sandbox



Your App Sandbox



Creating the database

Getting Resource Paths

Getting Resource Paths

```
//fetches path for foo.db
```

Getting Resource Paths

```
//fetches path for foo.db
NSString *resourcePath = [[NSBundle mainBundle]
```

Getting Resource Paths

```
//fetches path for foo.db
NSString *resourcePath = [[NSBundle mainBundle]
                          pathForResource:@"foo"
```

Getting Resource Paths

```
//fetches path for foo.db
NSString *resourcePath = [[NSBundle mainBundle]
                          pathForResource:@"foo"
                          ofType:@"db"];
```

Finding the Documents Directory

Finding the Documents Directory

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(
```

Finding the Documents Directory

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(
                                    NSDocumentDirectory,
```

Finding the Documents Directory

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(
                                NSDocumentDirectory,
                                NSUserDomainMask,
```

Finding the Documents Directory

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(
                                NSDocumentDirectory,
                                NSUserDomainMask,
                                YES);
```

Finding the Documents Directory

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(
                                NSDocumentDirectory,
                                NSUserDomainMask,
                                YES);

NSString *documentsDirectory = [paths objectAtIndex:0];
```

Copy a default database on startup

Copy a default database on startup

- //in applicationDidFinishLaunching

```
NSString *sourcePath = [[NSBundle mainBundle]
    pathForResource:@"app" ofType:@"db"];

NSString *targetPath = ...
    NSFileManager *fm = [NSFileManager defaultManager];

if(![fm fileExistsAtPath:targetPath]) {
  [fm copyItemAtPath:sourcePath toPath:targetPath error:&error];
}
```

Copy a default database on startup

- //in applicationDidFinishLaunching

```
NSString *sourcePath = [[NSBundle mainBundle]
    pathForResource:@"app" ofType:@"db"];

NSString *targetPath = ...
    NSFileManager *fm = [NSFileManager defaultManager];

if(![fm fileExistsAtPath:targetPath]) {
  [fm copyItemAtPath:sourcePath toPath:targetPath error:&error];
}
```

Copy a default database on startup

- //in applicationDidFinishLaunching

```
NSString *sourcePath = [[NSBundle mainBundle]
    pathForResource:@"app" ofType:@"db"];

NSString *targetPath = ...
    NSFileManager *fm = [NSFileManager defaultManager];

if(![fm fileExistsAtPath:targetPath]) {
  [fm copyItemAtPath:sourcePath toPath:targetPath error:&error];
}
```

Copy a default database on startup

- //in applicationDidFinishLaunching

```
NSString *sourcePath = [[NSBundle mainBundle]
    pathForResource:@"app" ofType:@"db"];

NSString *targetPath = ...
    NSFileManager *fm = [NSFileManager defaultManager];

if(![fm fileExistsAtPath:targetPath]) {
  [fm copyItemAtPath:sourcePath toPath:targetPath error:&error];
}
```

Copy a default database on startup

- //in applicationDidFinishLaunching

```
NSString *sourcePath = [[NSBundle mainBundle]
    pathForResource:@"app" ofType:@"db"];

NSString *targetPath = ...
    NSFileManager *fm = [NSFileManager defaultManager];

if(![fm fileExistsAtPath:targetPath]) {
  [fm copyItemAtPath:sourcePath toPath:targetPath error:&error];
}
```

Copy a default database on startup

- //in applicationDidFinishLaunching

```
NSString *sourcePath = [[NSBundle mainBundle]
    pathForResource:@"app" ofType:@"db"];

NSString *targetPath = ...
    NSFileManager *fm = [NSFileManager defaultManager];

if(![fm fileExistsAtPath:targetPath]) {
  [fm copyItemAtPath:sourcePath toPath:targetPath error:&error];
}
```

Copy a default database on startup

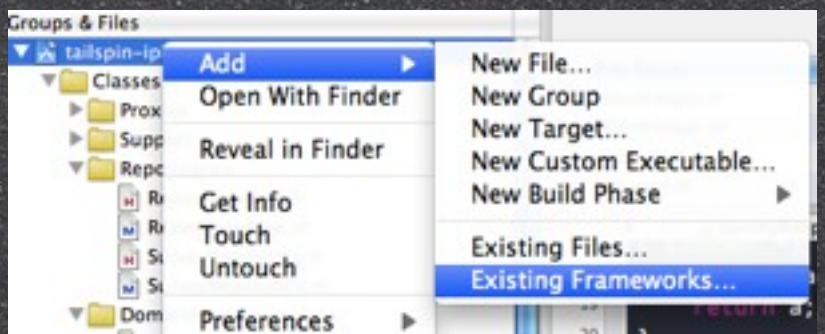
- //in applicationDidFinishLaunching

```
NSString *sourcePath = [[NSBundle mainBundle]
    pathForResource:@"app" ofType:@"db"];

NSString *targetPath = ...
    NSFileManager *fm = [NSFileManager defaultManager];

if(![fm fileExistsAtPath:targetPath]) {
  [fm copyItemAtPath:sourcePath toPath:targetPath error:&error];
}
```

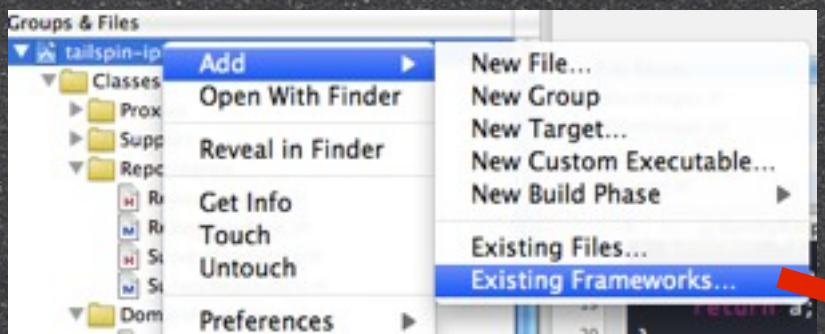
SQLite API Basics

Add the Framework

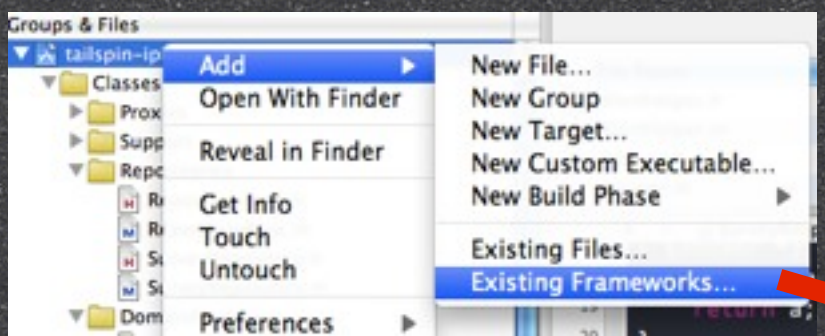
Add the Framework

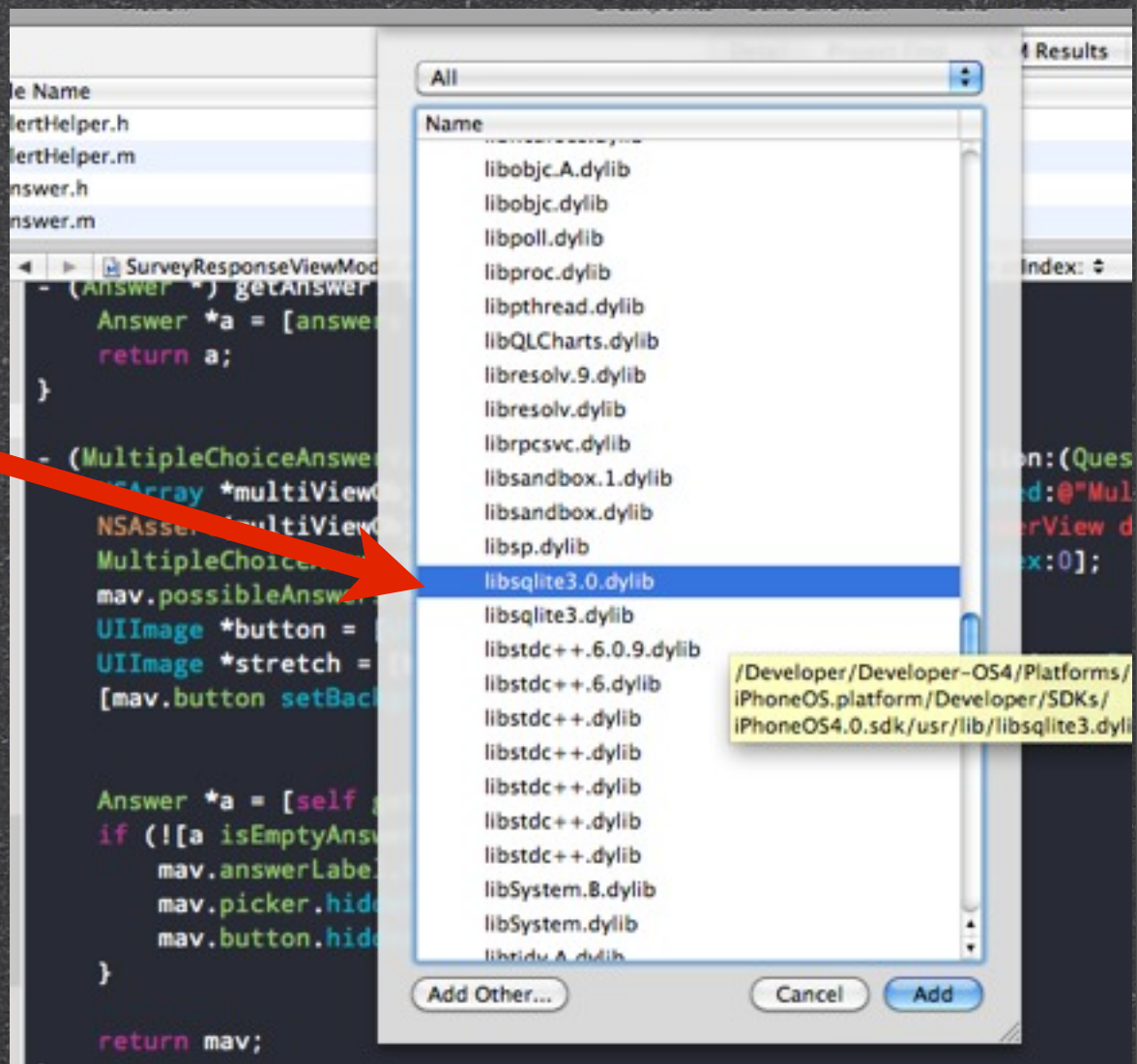


Add the Framework



Add the Framework





SQLite API - Opening a connection

SQLite API - Opening a connection

```
#import "sqlite3.h"
```

SQLite API - Opening a connection

```
#import "sqlite3.h"

sqlite3 *db;
```

SQLite API - Opening a connection

```
#import "sqlite3.h"

sqlite3 *db;

int result = sqlite3_open(PATH, &db);
```

SQLite API - Opening a connection

```
#import "sqlite3.h"

sqlite3 *db;

int result = sqlite3_open(PATH, &db);

if (result != SQLITE_OK) {
```

SQLite API - Opening a connection

```
#import "sqlite3.h"

sqlite3 *db;

int result = sqlite3_open(PATH, &db);

if (result != SQLITE_OK) {
    [NSException raise:@"SQLITE ERROR"
```

SQLite API - Opening a connection

```
#import "sqlite3.h"

sqlite3 *db;

int result = sqlite3_open(PATH, &db);

if (result != SQLITE_OK) {
    [NSException raise:@"SQLITE ERROR"
                format:@"Error %d", result];
```

SQLite API - Opening a connection

```
#import "sqlite3.h"

sqlite3 *db;

int result = sqlite3_open(PATH, &db);

if (result != SQLITE_OK) {
    [NSException raise:@"SQLITE ERROR"
                format:@"Error %d", result];
}
```

SQLite API - Opening a connection

```
#import "sqlite3.h"

sqlite3 *db;

int result = sqlite3_open(PATH, &db);

if (result != SQLITE_OK) {
    [NSException raise:@"SQLITE ERROR"
                format:@"Error %d", result];
}


//later
```

SQLite API - Opening a connection

```
#import "sqlite3.h"

sqlite3 *db;

int result = sqlite3_open(PATH, &db);

if (result != SQLITE_OK) {
    [NSException raise:@"SQLITE ERROR"
                format:@"Error %d", result];
}


//later
sqlite3_close(db);
```

SQLite API - executing queries

SQLite API - executing queries

```
int sqlite3_exec(
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
    int (*callback)(void *, int, char**, char**) callbackFunc,
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
    int (*callback)(void *, int, char**, char**) callbackFunc,
    void *context,
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
    int (*callback)(void *, int, char**, char**) callbackFunc,
    void *context,
    char **errorMessage
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
    int (*callback)(void *, int, char**, char**) callbackFunc,
    void *context,
    char **errorMessage
    );
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
    int (*callback)(void *, int, char**, char**) callbackFunc,
    void *context,
    char **errorMessage
    );


//callback signature
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
    int (*callback)(void *, int, char**, char**) callbackFunc,
    void *context,
    char **errorMessage
    );


//callback signature
int RowCallback(void * context,
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
    int (*callback)(void *, int, char**, char**) callbackFunc,
    void *context,
    char **errorMessage
    );


//callback signature
int RowCallback(void * context,
    int numColumns,
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
    int (*callback)(void *, int, char**, char**) callbackFunc,
    void *context,
    char **errorMessage
    );


//callback signature
int RowCallback(void * context,
    int numColumns,
    char** colValues,
```

SQLite API - executing queries

```
int sqlite3_exec(
    sqlite3 *db,
    char *sql,
    int (*callback)(void *, int, char**, char**) callbackFunc,
    void *context,
    char **errorMessage
    );


//callback signature
int RowCallback(void * context,
    int numColumns,
    char** colValues,
    char ** colNames);
```

SQLite API - Prepared Statements

More Powerful

Unfortunately much more code!

SQLite API-Prepared Statements

SQLite API-Prepared Statements

```
sqlite3 *db;
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}

sqlite3_stmt *stmt;
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}

sqlite3_stmt *stmt;
result = sqlite3_prepare_v2(db, sql_cstring, -1, &stmt, NULL);
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}

sqlite3_stmt *stmt;
result = sqlite3_prepare_v2(db, sql_cstring, -1, &stmt, NULL);
if(result != SQLITE_OK) {
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}

sqlite3_stmt *stmt;
result = sqlite3_prepare_v2(db, sql_cstring, -1, &stmt, NULL);
if(result != SQLITE_OK) {
    //handle error
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}

sqlite3_stmt *stmt;
result = sqlite3_prepare_v2(db, sql_cstring, -1, &stmt, NULL);
if(result != SQLITE_OK) {
    //handle error
}
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}

sqlite3_stmt *stmt;
result = sqlite3_prepare_v2(db, sql_cstring, -1, &stmt, NULL);
if(result != SQLITE_OK) {
    //handle error
}

//...
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}

sqlite3_stmt *stmt;
result = sqlite3_prepare_v2(db, sql_cstring, -1, &stmt, NULL);
if(result != SQLITE_OK) {
    //handle error
}

//...
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}

sqlite3_stmt *stmt;
result = sqlite3_prepare_v2(db, sql_cstring, -1, &stmt, NULL);
if(result != SQLITE_OK) {
    //handle error
}

//...
```

SQLite API-Prepared Statements

```
sqlite3 *db;
int result = sqlite3_open(DB_PATH, &db);
if(result != SQLITE_OK) {
    //handle error
}

sqlite3_stmt *stmt;
result = sqlite3_prepare_v2(db, sql_cstring, -1, &stmt, NULL);
if(result != SQLITE_OK) {
    //handle error
}

//...
```

SQLite API-Prepared Statements

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
    NSMutableDictionary *row = [NSMutableDictionary dictionary];
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
    NSMutableDictionary *row = [NSMutableDictionary dictionary];
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
    NSMutableDictionary *row = [NSMutableDictionary dictionary];

    [self processRow:row forStatement:stmt];
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
    NSMutableDictionary *row = [NSMutableDictionary dictionary];

    [self processRow:row forStatement:stmt];

    [results addObject:row];
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
    NSMutableDictionary *row = [NSMutableDictionary dictionary];

    [self processRow:row forStatement:stmt];

    [results addObject:row];
    result = sqlite3_step(stmt);
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
    NSMutableDictionary *row = [NSMutableDictionary dictionary];

    [self processRow:row forStatement:stmt];

    [results addObject:row];
    result = sqlite3_step(stmt);
}
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
    NSMutableDictionary *row = [NSMutableDictionary dictionary];

    [self processRow:row forStatement:stmt];

    [results addObject:row];
    result = sqlite3_step(stmt);
}

sqlite_finalize(stmt);
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
    NSMutableDictionary *row = [NSMutableDictionary dictionary];

    [self processRow:row forStatement:stmt];

    [results addObject:row];
    result = sqlite3_step(stmt);
}

sqlite_finalize(stmt);
```

SQLite API-Prepared Statements

```
//INSERT INTO CARS(year, make) VALUES(?, ?)

//PARAMETERS HAVE A 1-BASED INDEX!
sqlite3_bind_int(stmt,  1 /* idx */, 5);
sqlite3_bind_text(stmt, 2 /* idx */,"value", -1, SQLITE_TRANSIENT);

result = sqlite3_step(stmt);
if(result == SQLITE_DONE)
    return [NSArray array]; //nothing to do!

NSMutableArray *results = [NSMutableArray array];
while(result == SQLITE_ROW) {
    NSMutableDictionary *row = [NSMutableDictionary dictionary];

    [self processRow:row forStatement:stmt];

    [results addObject:row];
    result = sqlite3_step(stmt);
}

sqlite_finalize(stmt);

return results;
```

SQLite API-Prepared Statements

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
      if(type == SQLITE_INTEGER) {
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
      if(type == SQLITE_INTEGER) {
         value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
      if(type == SQLITE_INTEGER) {
         value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
      } else if (type == SQLITE_TEXT) {
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
     //0-based index!
     const char * colName = sqlite3_column_name(stmt, i);
     int type = sqlite3_column_type(stmt, i);

     id value = [NSNull null];
     if(type == SQLITE_INTEGER) {
        value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
     } else if (type == SQLITE_TEXT) {
        const unsigned char * text = sqlite3_column_text(stmt, i);
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
      if(type == SQLITE_INTEGER) {
         value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
      } else if (type == SQLITE_TEXT) {
         const unsigned char * text = sqlite3_column_text(stmt, i);
         value = [NSString stringWithFormat:@"%s", text];
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
      if(type == SQLITE_INTEGER) {
         value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
      } else if (type == SQLITE_TEXT) {
         const unsigned char * text = sqlite3_column_text(stmt, i);
         value = [NSString stringWithFormat:@"%s", text];
      } else {
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
      if(type == SQLITE_INTEGER) {
         value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
      } else if (type == SQLITE_TEXT) {
         const unsigned char * text = sqlite3_column_text(stmt, i);
         value = [NSString stringWithFormat:@"%s", text];
      } else {
         // more type handling
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
     //0-based index!
     const char * colName = sqlite3_column_name(stmt, i);
     int type = sqlite3_column_type(stmt, i);

     id value = [NSNull null];
     if(type == SQLITE_INTEGER) {
        value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
     } else if (type == SQLITE_TEXT) {
        const unsigned char * text = sqlite3_column_text(stmt, i);
        value = [NSString stringWithFormat:@"%s", text];
     } else {
        // more type handling
     }
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
     //0-based index!
     const char * colName = sqlite3_column_name(stmt, i);
     int type = sqlite3_column_type(stmt, i);

     id value = [NSNull null];
     if(type == SQLITE_INTEGER) {
        value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
     } else if (type == SQLITE_TEXT) {
        const unsigned char * text = sqlite3_column_text(stmt, i);
        value = [NSString stringWithFormat:@"%s", text];
     } else {
        // more type handling
     }
     [row setObject:value forKey:
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
      if(type == SQLITE_INTEGER) {
         value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
      } else if (type == SQLITE_TEXT) {
         const unsigned char * text = sqlite3_column_text(stmt, i);
         value = [NSString stringWithFormat:@"%s", text];
      } else {
         // more type handling
      }
      [row setObject:value forKey:
          [NSString stringWithUTF8String:colName]];
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
      if(type == SQLITE_INTEGER) {
         value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
      } else if (type == SQLITE_TEXT) {
         const unsigned char * text = sqlite3_column_text(stmt, i);
         value = [NSString stringWithFormat:@"%s", text];
      } else {
         // more type handling
      }
      [row setObject:value forKey:
          [NSString stringWithUTF8String:colName]];
  }
```

SQLite API-Prepared Statements

```
-(void)processRow:(NSMutableDictionary *)row forStatement:
(sqlite3_stmt*)stmt {
  int columnCount = sqlite3_column_count(stmt);
  for(int i=0; i<columnCount; i++) {
      //0-based index!
      const char * colName = sqlite3_column_name(stmt, i);
      int type = sqlite3_column_type(stmt, i);

      id value = [NSNull null];
      if(type == SQLITE_INTEGER) {
         value = [NSNumber numberWithInt:sqlite3_column_int(stmt, i)];
      } else if (type == SQLITE_TEXT) {
         const unsigned char * text = sqlite3_column_text(stmt, i);
         value = [NSString stringWithFormat:@"%s", text];
      } else {
         // more type handling
      }
      [row setObject:value forKey:
          [NSString stringWithUTF8String:colName]];
  }
}
```

FMDB

- http://github.com/ccgus/fmdb

FMDB Usage

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
    NSLog(@"Could not open db.");
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
    NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
    NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
    NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
     [db lastErrorMessage]);
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
     [db lastErrorMessage]);

[db executeUpdate:@"create table test (a text, b integer)"];
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
     [db lastErrorMessage]);

[db executeUpdate:@"create table test (a text, b integer)"];

[db beginTransaction];
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
     [db lastErrorMessage]);

[db executeUpdate:@"create table test (a text, b integer)"];

[db beginTransaction];
int i = 0;
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
     [db lastErrorMessage]);

[db executeUpdate:@"create table test (a text, b integer)"];

[db beginTransaction];
int i = 0;
while (i++ < 20) {
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
     [db lastErrorMessage]);

[db executeUpdate:@"create table test (a text, b integer)"];

[db beginTransaction];
int i = 0;
while (i++ < 20) {
  [db executeUpdate:@"insert into test (a, b) values (?, ?)" ,
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
    NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
   NSLog(@"Err %d: %@", [db lastErrorCode],
      [db lastErrorMessage]);

[db executeUpdate:@"create table test (a text, b integer)"];

[db beginTransaction];
int i = 0;
while (i++ < 20) {
   [db executeUpdate:@"insert into test (a, b) values (?, ?)" ,
      @"hi'", // look!  I put in a ', and I'm not escaping it!
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
     [db lastErrorMessage]);

[db executeUpdate:@"create table test (a text, b integer)"];

[db beginTransaction];
int i = 0;
while (i++ < 20) {
  [db executeUpdate:@"insert into test (a, b) values (?, ?)" ,
     @"hi'", // look!  I put in a ', and I'm not escaping it!
     [NSNumber numberWithInt:i]];
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
     [db lastErrorMessage]);

[db executeUpdate:@"create table test (a text, b integer)"];

[db beginTransaction];
int i = 0;
while (i++ < 20) {
  [db executeUpdate:@"insert into test (a, b) values (?, ?)" ,
     @"hi'", // look!  I put in a ', and I'm not escaping it!
     [NSNumber numberWithInt:i]];
}
```

FMDB Usage

```
FMDatabase* db = [FMDatabase databaseWithPath:@"/tmp/tmp.db"];
if (![db open])
   NSLog(@"Could not open db.");

[db executeUpdate:@"blah blah blah"];

if ([db hadError])
  NSLog(@"Err %d: %@", [db lastErrorCode],
     [db lastErrorMessage]);

[db executeUpdate:@"create table test (a text, b integer)"];

[db beginTransaction];
int i = 0;
while (i++ < 20) {
  [db executeUpdate:@"insert into test (a, b) values (?, ?)" ,
     @"hi'", // look!  I put in a ', and I'm not escaping it!
     [NSNumber numberWithInt:i]];
}
[db commit];
```

What about updates?

- Database is never copied over again
- If we did, your customers would lose data

Migrations

- Concept from Ruby on Rails
- Evolutionary database design
- Each database has a "version"
- App runs all migrations to the latest version

Migrations

number	name
1	initial_schema
2	update_foo
3	implement_blah

Migrations

number	name
1	initial_schema
2	update_foo
3	implement_blah

Hand-rolled Database Migrator

- DEMO

FMDB Migration Manager

- http://github.com/mocra/fmdb-migration-manager

FMDB Migration Manager

```
NSArray *migrations = [NSArray arrayWithObjects:
    [CreateStudents migration], // 1
    [CreateStaffMembers migration], // 2
    [AddStudentNumberToStudents migration], // 3
    nil
  ];
[FmdbMigrationManager executeForDatabasePath:@"/tmp/tmp.db"
withMigrations:migrations];
```

FMDB Migration Manager

```
@interface CreateStudents : FmdbMigration
{
}
@end

@implementation CreateStudents
- (void)up {
  [self createTable:@"students" withColumns:[NSArray arrayWithObjects:
      [FmdbMigrationColumn columnWithColumnName:@"first_name" columnType:@"string"],
      [FmdbMigrationColumn columnWithColumnName:@"age" columnType:@"integer"
            defaultValue:21], nil];
}

- (void)down {
  [self dropTable:@"students"];
}

@end
```

ActiveRecord

```
Person *p = [[Person alloc] init];
p.name = @"Joe";
p.occupation = @"Dishwasher";

[p save];
```

```
Person *p = [Person personWithId:5];
==> Person {id:5, name:Joe, occupation:Dishwasher}
```

Aptiva's ActiveRecord

- http://github.com/aptiva/activerecord

(YMMV)

Questions?