|  |  |  |
| --- | --- | --- |
|  | [sqlite3 둘러보기](http://www.cocoadev.co.kr/212) [기타](http://www.cocoadev.co.kr/category/기타) 2009/02/02 14:03  iPhone SDK를 사용하면서 데이터베이스로는 내장된 [sqlite3](http://www.sqlite.org/)를 조금 사용해 보았습니다. ruby on rails나 이전에 조금 본적은 있지만, 실제로 사용하다 보니 기존의 일반적인인 데이터베이스에 비해 몇가지 재미있고 색다른 부분이 있는 것 같습니다.  **1. 특징 & 둘러보기** **1) ROWID** SQLite3의 테이블등은 생성시 기본적으로 rowid란 칼럼을 가지고 있습니다. auto increment되는 primary key와 같은 역활을 하지만 vacuum에 의해 변경될 수가 있다고 하니, 이런 용도로는 테이블의 특성에 맞게 사용해야 될 것 같습니다.  [http://cfs12.tistory.com/image/18/tistory/2009/01/20/14/03/49755b30139e0](http://cfs12.tistory.com/original/18/tistory/2009/01/20/14/03/49755b30139e0)  select \*로는 rowid가 출력되지 않고 명시를 해주어야지 값을 확인할 수있습니다. 마지막으로 인서트된 항목의 rowid는 sqlite3\_last\_insert\_rowid 함수의 반환값으로 확인할 수 있습니다.  **2) 데이터 타입** 타입이 컬럼에 일괄적으로 적용되는 다른 데이터베이스와는 달리 sqlite3는 각각의 로우별로 동적으로 타입이 결정됩니다. 그렇기 때문에 칼럼타입에 어떤 문자를 주던지 혹은 생략하더라도 테이블 생성이 가능합니다. 아래와 같이 테이블을 만든 후에 어떤 타입의 값들을 넣더라도 모두 저장이 가능합니다.  [http://cfs13.tistory.com/image/12/tistory/2009/01/19/19/19/49745398a39f2](http://cfs13.tistory.com/original/12/tistory/2009/01/19/19/19/49745398a39f2)  아래와 같이 각 컬럼의 값들에 따라 데이터타입이 결정됩니다.  [http://cfs12.tistory.com/image/9/tistory/2009/01/20/14/22/49755f836f166](http://cfs12.tistory.com/original/9/tistory/2009/01/20/14/22/49755f836f166)  한 컬럼에는 기본으로 1GB까지 저장이 가능하며 컴파일시에 SQLITE\_MAX\_LENGTH 값을 설정하여 2GB까지 가능합니다. 하지만 이전에 간단히 테스트를 해보았는데 OS X에서는 3MB 정도까지만 가능했습니다. 아마 애플에서 제한크기를 낮추어 설치하지 않았나 짐작을 하고 있습니다. 리눅스에서는 몇 백MB까지 인서트가 가능했습니다.  **2. 기본 명령어** **1) 시작** sqlite는 파일을 기반으로 한 컴팩트한 DB입니다. 대부분의 DB들이 파일을 기반으로 하지만 sqlite3는 데이터베이스 오픈시 인자로 파일을 받습니다.  > sqlite3 [file-name]  파일이 존재할 경우에는 기존 db파일을 열고 존재하지 않을 경우에는 새로 생성합니다. 쉘에서는 쿼리와 구별하여 명령어에 '.'이 접두사로 사용됩니다.   **2) 종료** .quit 또는 .exit는 sqlite3를 종료하는 명령어입니다.  sqlite3>.quit sqlite3>.exit  **3) 도움말**  sqlite3>.help  sqlite3 쉘에서 사용가능한 명령어들을 보여 줍니다.  **4) 테이블 목록**  sqlite3>.tables  현재 데이터베이스에 등록된 테이블들의 목록을 확인할 수 있습니다.  **5) 스키마 정보**  sqlite3>.schema [table-name]  지정된 테이블의 스키마를 확인할 수 있습니다. 테이블명을 입력하지 않으면 모든 테이블의 스키마가 출력됩니다. "ALTER TABLE"도 있는데 add와 rename만 가능한 것 같습니다.   **6) 쿼리결과를 파일로 저장**  sqlite3>.mode insert sqlite3>.output db.sql sqlite3>select \* from mytable; sqlite3>.quit  mode는 출력될 타입을 지정하며 csv, column, html, insert, line, list, tabs, tcl등으로 설정할 수 있습니다. output은 저장될 파일을 지정합니다. 위와 같이 실행 후 종료하면 mytable의 데이터들이 insert로된 sql문으로 db.sql 파일로 저장되어 있습니다.  **7) 테이블 변경**  sqlite3>alter table [table\_name] rename to [new\_table\_name]; sqlite3>alter table [table\_name] add [new\_column\_name];  alter는 테이블명의 변경과 테이블 컬럼의 추가만 가능한 것 같습니다.  **8) vacuum**  sqlite3>vacuum  데이터베이스의 테이블의 로우를 재정렬하고 delete, drop으로 인한 빈공간을 제거하는 최적화 작업을 수행합니다. insert, delete, drop등의 명령을 자주 수행하는 DB들은 정기적으로 vacuum을 실행해 주는 것이 좋습니다.  **3. 자주 사용되는 C 함수** **1) Open & Close**  int sqlite3\_open(const char \*filename, sqlite3 \*\*ppDb); int sqlite3\_close(sqlite3 \*pDb);  sqlite3 데이터베이스 파일을 열고 닫는 함수 입니다.  **사용 예)**  sqlite3 \*db; NSString\* path = @"./mydb.sqlite3";  if (sqlite3\_open([path UTF8String], &db) != SQLITE\_OK)     NSLog(@"Fail to open sqlite3: %s", sqlite3\_errmsg(db));     sqlite3\_close(db);          return NULL; }  **2) 쿼리 실행**  nt sqlite3\_prepare\_v2(sqlite3 \*db, const char \*zSql, int nByte, sqlite3\_stmt \*\*ppStmt, const char \*\*pzTail); int sqlite3\_step(sqlite3\_stmt\* stmt); int sqlite3\_finalize(sqlite3\_stmt \*pStmt);  쿼리는 prepare -> step -> finalize의 단계로 실행되고 종료됩니다. prepare는 쿼리실행에 앞서 쿼리를 컴파일하여 바이트코드로 변경합니다. step은 prepare에서 준비된 코드를 실행하며 실행될때 마다 다음 데이터를 가지고 옵니다. finalize는 prepare와 쌍으로 할당된 sqlite3\_stmt의 메모리를 해제합니다.  **사용 예)**  if (sqlite3\_prepare\_v2(prevDB, "SELECT \* FROM cross\_temp", -1, &statement, NULL) == SQLITE\_OK) {     while (sqlite3\_step(statement) == SQLITE\_ROW) {         int pid = sqlite3\_column\_int(statement, 0);         NSLog(@"pid %d", pid);     } } sqlite3\_finalize(statement);  **3) 결과값**  int sqlite3\_column\_int(sqlite3\_stmt\*, int iCol); const unsigned char \*sqlite3\_column\_text(sqlite3\_stmt\*, int iCol);  스텝에서 각 컬럼의 값들은 column\_ 함수로 가져올 수 있습니다. 위는 int와 char\*의 예이며 이외에도 각 타입에 맞는 함수들이 제공됩니다.  **4) 쿼리 실행 - exec**  int sqlite3\_exec(sqlite3\*, const char \*sql, int (\*callback)(void\*,int,char\*\*,char\*\*), void \*, char \*\*errmsg);  exec은 위의 prepare, step, finalize를 동시에 실행하여 편리하게 사용할 수 있습니다. select와 같은 쿼리에서 각 단계의 데이터들은 콜백함수를 구현하여 처리할 수 있습니다.   **사용 예)**  if (sqlite3\_exec(db, "SELECT \* FROM item\_info limit 10", setData, NULL, &error)) {     NSLog(@"EXEC ERROR: %s", error);     sqlite3\_free(error); }  콜백함수의 첫번째 인자는 sqlite3\_exec의 4번째 인자에서 넘긴 사용자 데이터가 넘어 옵니다. cols는 전체 컬럼의 갯수이며 value, name에 각각 값과 컬럼명이 넘어 옵니다.  int setData(void\* data,int cols, char\*\* value, char\*\* name) {      for (int i = 0; i < cols; i++) {         NSLog(@"%s, %s", value[i], name[i]);     }     return 0; }  \* **아이폰에서 쓰기 가능한 DB**  어플리케이션은 실행시에 번들 내부의 파일에는 쓰기 권한이 없습니다. 그렇기 때문에 번들 내부에 있는 sqlite3 파일은 insert, update, delete와 같은 DB를 변경하는 쿼리가 실행되지 않습니다.   이를 해결하기 위해서는 Documents 디렉토리로 DB파일을 복사하고 이곳의 파일로 열어야 쓰기가 가능합니다. 애플의 아이폰 개발자 센터에 있는 [SQLiteBooks](http://developer.apple.com/iphone/library/samplecode/SQLiteBooks/index.html) 샘플에서 AppDelegate.m의 createEditableCopyOfDatabaseIfNeeded 메소드를 확인하시면 구현에 대해서 잘 나와있습니다. 그리고 아이폰에서 sqlite3 구현에 대한 간단한 내용은 이전 [아이폰 SQLite3 샘플](http://www.cocoadev.co.kr/203)이란 포스팅에서 확인하실 수 있습니다. [아이폰 SQLite3 샘플](http://www.cocoadev.co.kr/203) SQLite는 경량화된 DB로 맥과 아이폰/아이팟 터치에 기본적으로 내장되어 편리하게 사용할 수 있습니다. 자세한 내용은 [SQLite 공식 홈페이지](http://www.sqlite.org/)와 아래의 문서들을 참조하시면 도움이 되실 것입니다.   * [SQLite C Interface - Functions](http://www.sqlite.org/c3ref/funclist.html) * [SQLite In 5 Minutes Or Less](http://www.sqlite.org/quickstart.html) * [Wikipedia - SQLite](http://en.wikipedia.org/wiki/Sqlite)   이와함께 애플의 [iPhoneDev Center](http://developer.apple.com/iphone/)에 SQLite Book List란 샘플을 보시면, 아이폰 SDK에서 사용하는 방법이 잘 나와있습니다. SQLite의 개발자인 [Richard Hipp](http://www.hwaci.com/drh/)이 구글 테크토크에서 직접 SQLite에 대해서 설명하는 '[An Introducion to SQLite](http://video.google.com/videoplay?docid=-5160435487953918649)'란 동영상도 참고하면 좋습니다.  [http://cfs14.tistory.com/image/22/tistory/2008/11/25/14/17/492b8a8268f39](http://cfs14.tistory.com/original/22/tistory/2008/11/25/14/17/492b8a8268f39)  아래는 제가 SQLite를 테스트 해보기 위해 만들어 본 간단한 샘플코드입니다. 아이폰에서 사용자로 부터 입력을 받은 후에 SQLite DB에 저장하는 간단한 샘플입니다. DB를 오픈하는 부분과 SELECT, INSERT하는 부분만 참고하시면 쉽게 사용하실 수 있습니다.  **\* SQLiteTestAppDelegate.h**  #import <UIKit/UIKit.h> #import <sqlite3.h>  @interface SQLiteTestAppDelegate : NSObject <UIApplicationDelegate> {     UIWindow \*window;     sqlite3 \*db;     NSMutableArray \*dataList;      IBOutlet UITextField \*newString;     IBOutlet UITableView \*dataTable; }  @property (nonatomic, retain) IBOutlet UIWindow \*window;  - (void)updateDataList;  - (IBAction)addRow:(id)sender;      @end  **SQLiteTestAppDelegate.m**  #import "SQLiteTestAppDelegate.h"  @implementation SQLiteTestAppDelegate  @synthesize window;  - (void)applicationDidFinishLaunching:(UIApplication \*)application {          // Override point for customization after application launch     [window makeKeyAndVisible];          /\* 어플리케이션 패스를 구한다. \*/          NSArray \*paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);     NSString \*documentsDirectory = [paths objectAtIndex:0];     NSString \*path = [documentsDirectory stringByAppendingPathComponent:@"mydata.db"];              /\* 데이터베이스를 오픈한다 \*/     if(sqlite3\_open([path UTF8String], &db) == SQLITE\_OK) {         char \*error = NULL;         const char\* query = "SELECT count(\*) from mytable";                  /\* mytable을 쿼리해보고 오류가 있으면 mytable을 생성한다. \*/         if (sqlite3\_exec(db, query, NULL, 0, &error) != SQLITE\_OK) {             sqlite3\_free(error);                          /\* 테이블 생성 \*/             if (sqlite3\_exec(db, "CREATE TABLE mytable ('name' CHAR(16))", NULL, 0, &error) != SQLITE\_OK) {                 NSLog(@"TABLE CREATE ERROR: %s", error);                 sqlite3\_free(error);             }             }     } else {         /\* DB 오픈 에러 \*/         sqlite3\_close(db);         db = NULL;                  NSLog(@"DB OPEN ERROR: '%s'", sqlite3\_errmsg(db));         }          dataList = [[NSMutableArray alloc] initWithCapacity:100];          [self updateDataList]; }  - (void)applicationWillTerminate:(UIApplication \*)application {     if (db) {         sqlite3\_close(db);     } }  - (void)dealloc {     [dataList release];          [window release];     [super dealloc]; }  /\*\* 현재 DB에 있는 데이터를 dataList에 등록 \*/ - (void)updateDataList {          /\* 이전 데이터를 모두 삭제 \*/     [dataList removeAllObjects];          const char \*query = "SELECT name FROM mytable";     sqlite3\_stmt \*statement;          if (sqlite3\_prepare\_v2(db, query, -1, &statement, NULL) == SQLITE\_OK) {         while (sqlite3\_step(statement) == SQLITE\_ROW) {                          /\* dataList에 쿼리결과 등록 \*/             NSString\* str = [[NSString alloc] initWithUTF8String:(char \*)sqlite3\_column\_text(statement, 0)];             [dataList addObject:str];             [str release];         }     }     sqlite3\_finalize(statement);      [dataTable reloadData]; }   #pragma mark IBACTION  - (IBAction)addRow:(id)sender {      char \*error = NULL;          /\* 사용자가 입력한 값을 DB에 추가한다 \*/     NSString \*query = [NSString stringWithFormat:@"INSERT INTO mytable VALUES ('%@')", [newString text]];     sqlite3\_exec(db, [query UTF8String], NULL, 0, &error);          [self updateDataList]; }  #pragma mark TextField Delegate method  - (BOOL)textFieldShouldReturn:(UITextField \*)textField {          [textField resignFirstResponder];     return YES; }  #pragma mark TableView Delegate method  - (NSInteger)numberOfSectionsInTableView:(UITableView \*)tableView {     return 1; } - (NSString \*)tableView:(UITableView \*)tableView titleForHeaderInSection:(NSInteger)section {     return @"Table Items"; } - (NSInteger)tableView:(UITableView \*)tableView numberOfRowsInSection:(NSInteger)section {     return [dataList count]; } - (UITableViewCell \*)tableView:(UITableView \*)tableView cellForRowAtIndexPath:(NSIndexPath \*)indexPath {          static NSString \*cellIdentifier = @"Cell";          UITableViewCell \*cell = [tableView dequeueReusableCellWithIdentifier:cellIdentifier];     if (cell == nil) {         cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero reuseIdentifier:cellIdentifier] autorelease];     }     cell.text = [dataList objectAtIndex:indexPath.row];          return cell; } @end   |  | | --- | | **sqlite 도움말(help) 명령어 (command)** http://blogimgs.naver.net/imgs/nblog/spc.gif |   .backup ?DB? FILE // 데이터 베이스를 백업한다.  .bail ON|OFF .databases // 데이터베이스 정보 보기 // mysql> show databases; .dumb ?TABLE? ... // 덤프를 만들어 준다. .echo ON|OFF // 실행된 명령문 보기 .exit //끝내기 // mysql>exit .explain ?ON|OFF?  .header(s) ON|OFF // 컬럼명 표시/비표시 .help // 도움말 보기 // mysql>help  .import FILE TABLE .indices ?TABLE? // 사용가능한 인덱스 보기 .load FILE ?ENTRY? .log FILE|off // 로그 파일 설정 .mode MODE ?TABLE? // 출력 결과 형식  [scv | column | html | insert | line | list | tabs | tcl] .nullvalue STRING // null 값을 어떻게 표시 할지 설정 .output FILENAME // 결과 출력을 지정된 파일 이름의 파일 형태로 한다. .output stdout // 결과 출력을 화면에 한다. .prompt MAIN CONTINUE // prompt 앞의 표시 설정 ex).prompt "mysql> " "> " .quit // 끝내기 // mysql>quit .read FILENAME // 파일을 읽어서 명령어를 실행. .restore ?DB? FILE // 데이터베이스 복구 .schema ?TABLE? // 테이블 정보 보기   mysql>show create table [TABLE\_NAME]  mysql>desc [TABLE\_NAME] .separator STRING // 컬럼 구분자 설정 .show // 설정 정보 보기 .stats ON|OFF // sqlite 상태 표시/비표시 .tables ?TABLE? // 테이블 보기 // mysql>show tables; .timeout MS     MS=milli second // 타임아웃 시간 설정 .width NUM1 NUm2 ..//결과의 컬럼의 넓이 설정[.mode column]일때만 적용 됨. .timer ON|OFF // 쿼리 실행시간 표시/비표시   * 시작 >sqlite3 [file-name] * 종료 >sqlite3>.quit // sqlite3>.exit * 도움말 >sqlite3>.help * 테이블 목록 확인 >sqlite3>.tables * 스키마 정보 >sqlite3>.schema [table-name] * 쿼리 결과를 파일로 저장  >sqlite3>.mode insert >sqlite3>.output db.sql >sqlite3>select \* from my table; * 테이블 변경 sqlite3>alter table [table\_name] rename to [new\_table\_name]; sqlite3>alter table [table\_name] add [new\_column\_name]; * Vacuum >sqlite3>vacuum DB의 테이블 row를 재정렬하고 delete, drop으로 인한 빈공간을 제거하는 최적화 작업을 수행합니다. insert, delete, drop 등 의 명령을 수행하는 DB들은 정기적으로 vacuum을 실행해 주는 것이 좋습니다. |

SQLite 설치  
- 경로 : <http://sqlite.org/download.html> - 여기 가면 윈도우용, 리눅스용이 있습니다.  
- 제경우 리눅스용 sqlite-2.8.6 과 sqlite-devel-2.8.6 RPM 을 다운 받아  
설치했습니다.. 환경설정파일 같은 것도 없고. 데몬을 띄우는 것도 필요 없습니다.  
그냥 rpm 으로 깔면 끝입니다.  
- 아래는 간단한 사용례 입니다.

$ sqlite test\_db  
SQLite version 2.8.6  
Enter ".help" for instructions  
sqlite>  
쉘에서 이렇게만 하면 현재 디렉토리에 test\_db 라는 파일이 하나 만들어집니다.  
이 파일(DB)을 php에서 사용하려면 당연히 웹서버가 읽고 쓸수 있는 권한이 있어야  
합니다.

sqlite> create table test\_tb ( id int PRIMARY KEY, name varchar(30));  
테이블 test\_tb 를 만듭니다.  
sqlite> .tables //테이블목록 보기  
test\_tb  
sqlite> .schema test\_tb // 테이블의 구조를 봅니다.  
create table test\_tb ( id int PRIMARY KEY, name varchar(30));  
sqlite> insert into test\_tb values ( 1,'권기성');  
// AUTO\_INCREMENT 기능하는 INSERT  
sqlite> insert into test\_tb values((SELECT max(id) FROM test\_tb)+1,'권기성2');  
sqlite> select \* from test\_tb;  
1|권기성  
2|권기성2

sqlite> create index name\_idx on test\_tb (name) ; //인덱스 만들기

sqlite> begin ; //트랜잭션도 가능  
sqlite> insert into test\_tb values ( 3,'김기성');  
sqlite> select \* from test\_tb;  
1|권기성  
2|권기성2  
3|김기성  
sqlite> rollback; //롤백  
sqlite> select \* from test\_tb;  
1|권기성  
2|권기성2

sqlite> .output backup.txt //다음줄 select 의 결과가 backup.txt 파일로 저장됨  
sqlite> select \* from test\_tb ;  
sqlite> .exit //종료

// 쉘에서 test\_db 를 dump 받아서 압축함 ( mysqldump 와 비슷 )  
$ echo '.dump' | sqlite test\_db | gzip -c > test\_db.dump.gz //백업하기  
$ zcat test\_db.dump.gz | sqlite test\_db //복원하기

3. PHP 에서 모듈설치  
- PHP 5 베타 : --with-sqlite 옵션을 주고 컴파일합니다.  
(참고로 php 5 에서는 mysql 을 사용하려면 반드시 --with-mysql 을 줘야합니다.)  
컴파일후 pearDB 를 사용하려면 이 경우도  
# pear install SQLite 명령어로 sqlite 패키지를 설치해야할겁니다.(아마)

- PHP 4.3.3 : 제경우 4.3.3 에서 테스트 했습니다. 최소 가능 버전이 얼마인지는 모르겠습니다.  
이경우 php 의 최초 컴파일시 --with-sqlite 라는 옵션자체가 없습니다.  
그러므로 일반적으로 php 컴파일 해서 설치한 후 나중에  
PEAR 패키지로 따로 설치를 해야합니다. 리눅스 쉘에서 루트 권한으로

# pear install SQLite

라고 하면 패키지를 자동으로 다운 받아서 소스 풀고 한참 컴파일 합니다.  
결과적으로 {PHP설치위치}/extensions/no-debug-non-zts-20020429 와 같은 디렉토리에  
sqlite.so 라는 so 파일이 하나생깁니다.  
이파일을 {PHP설치위치}/extensions/ 로 복사한후  
php.ini 파일에서

extension=sqlite.so

와 같이 동적 모듈을 사용하도록 수정합니다.  
조심할 점은 php.ini 에서 extension\_dir 이 제대로 설정되거나 아니면 주석처리되어  
디폴트 위치를 찾도록 해야합니다.

이렇게 한후 웹서버를 재시작하고, phpinfo() 로 봤을때 sqlite 항목이 나오면 제대로  
설치가 된겁니다.

- PEAR DB  
위에서 pear install SQLite 명령시 pear 패키지로 자동 설치하며  
{PHP설치위치}/php/DB/sqlite.php 라는 파일로 저장됩니다.  
이 파일을 통해서 sqlite 도 PEAR DB 에서 간편하게 사용가능합니다.

4. PHP 소스  
- 간단한 사용법은 아래와 같습니다. DB 연결부분은 쉽게 이해가 가실겁니다.  
- 실패하기 제일 쉬운 이유는 DB 인 test\_db 라는 파일의 경로가 제대로 설정되어야하고  
웹서버가 읽고 쓸수 있는 권한이 주어져야합니다.

<?php  
// PEAR DB 연결  
require\_once 'DB.php';  
require\_once 'DB/sqlite.php';

// DSN 정의 - DB 파일의 경로와 퍼미션 주의  
$dsn = array (  
'phptype' => "sqlite", 'database' => "./test\_db",  
'mode' => 0644  
);  
$db = &new DB\_sqlite();  
$db->connect($dsn, array('persistent'=> true) );  
$table = 'test\_tb'; // 테이블명

//입력  
$result = $db->query("INSERT INTO $table VALUES (9,'아무개');");

printf("affectedRows:\t\t%s\n<br>", $db->affectedRows() );  
printf("getLastInsertId:\t%s\n<br>", $db->getLastInsertId() );  
$result = $db->query("SELECT \* FROM $table;" );  
while($row = $db->fetchrow($result)){  
print\_r($row ); echo "<br>";  
}  
$db->disconnect();  
?>

5. 기타  
- <http://www.sqlite.org> 와 그곳에 있는 링크들의 정보를 참조하세요..  
  
- sqlite 관리툴은 sf.net 에서 sqlite 라는 검색어로 찾아보면 몇개 있습니다.  
<http://sourceforge.net/projects/sqlite-admin/> 이것은 phpMyadmin 비슷한 php 관리툴인데  
단점은 php5 가 필요합니다.  
<http://sourceforge.net/projects/sqlitebrowser/> 이것은 Qt 로된 리눅스용 GUI 입니다.

- 위의 사용예에서 보인 pear DB 를 사용하지 않고 ,그냥 php 의 sqlite 함수를 이용하려면  
<http://kr.php.net/manual/kr/ref.sqlite.php> 이곳을 참조하세요.