

SQLite 를 사용한 Database 처리

2022년 여름 계절학기
산학 협력 프로젝트

목차

SQLite

Swift 에서의 SQLite 사용

DB의 사용

- ◆ 사용준비
- ◆ 테이블 생성
- ◆ 데이터 추가/검색/수정/삭제
- ◆ 테이블 제거
- ◆ 종료 작업

SQLite

클라이언트 어플리케이션에 주로 사용하는 경량 내장형 DBMS

- ◆ 관계형 데이터베이스
- ◆ <http://www.sqlite.org>
- ◆ 안드로이드, iOS, 그리고 웹 브라우저 등에서 사용 → 안드로이드의 경우 프레임워크에 기본 내장

기기에 자료를 영구로 저장해야 할 경우 적용

- ◆ 휴대폰 내부에 파일로 DB가 만들어짐
- ◆ 라이브러리 형태로 호출하여 사용 (클래스 import)



Swift에서의 SQLite 사용

데이터 관리

- ◆애플 기본: UserDefaults, CoreData, 파일 처리
- ◆외부 소스 활용: SQLite, Realm, Firebase

iOS 의 SQLite DB 사용

- ◆SQLite3 패키지 사용 – Swift4 버전에서 주로 사용
- ◆접두어 sqlite3_ 를 갖는 함수를 주로 사용

사용 절차

- ◆DB Open 작업
- ◆Table 생성 (생성하지 않았을 경우)
- ◆CRUD 연산: Create, Read, Update, Delete
- ◆DB close 작업



각 버튼 및 **action** 함수 추가

- ◆ DB 열기: btnOpenDatabase
- ◆ 테이블 생성: btnCreateTable
- ◆ 샘플 데이터 추가: btnInsert
- ◆ 전체 데이터 확인: btnSelectAll
- ◆ 특정 데이터 수정: btnUpdate
 - id 1 인 항목 이름을 수정
- ◆ 특정 데이터 삭제: btnDelete
 - id 1 인 항목을 삭제
- ◆ 테이블 삭제: btnDropTable
 - 테이블 삭제
- ◆ DB 닫기: btnCloseDatabase

DB 사용 준비

📁 관련 패키지 import

```
import SQLite3
```

📁 필요 상수 및 변수 선언

```
let DB_NAME = "my_db.sqlite"
let TABLE_NAME = "my_table"
let COL_ID = "id"
let COL_NAME = "name"

var db: OpaquePointer? = nil // 데이터베이스를 가리키는 포인터
```

📁 DB 파일 준비

◆ DB 파일 열기 직전 수행

```
let dbFile = try! FileManager.default.url(for: .documentDirectory,
                                           in: .userDomainMask,
                                           appropriateFor: nil,
                                           create: false)

    .appendingPathComponent(DB_NAME)
```

DB 열기

DB 파일 열기

- ◆ DB 파일이 없을 경우 생성 후 열기 수행

```
if sqlite3_open(dbFile.path, &db) == SQLITE_OK {  
    print("Successfully Opened")  
    print(dbFile)  
} else {  
    print("Unable to open DB")  
}
```

DB close

- ◆ DB 사용 완료 후 close

```
sqlite3_close(db)
```

DB 테이블 생성

데이터를 저장할 테이블 생성

my_table

id (Integer)	name (text)
-----------------	----------------

```
let createTableString = """
    CREATE TABLE IF NOT EXISTS \$(TABLE_NAME) ( \$(COL_ID) INTEGER PRIMARY KEY AUTOINCREMENT,
        \$(COL_NAME) TEXT);
    """

var createTableStmt: OpaquePointer?

print ("TABLE SQL: \$(createTableString)")

if sqlite3_prepare_v2(db, createTableString, -1, &createTableStmt, nil) == SQLITE_OK {
    if sqlite3_step(createTableStmt) == SQLITE_DONE {
        print("Successfully created.")
    }
    sqlite3_finalize(createTableStmt)
} else {
    let error = String(cString: sqlite3_errmsg(db)!)
    print("Table Error: \$(error)")
}
```


테이블에 데이터 추가

insert 문을 사용한 데이터 추가

◆ prepared statement 사용

```
var insertStmt: OpaquePointer?

if sqlite3_prepare_v2(db, "insert into \((TABLE_NAME) values (null, ?)", -1, &insertStmt, nil) == SQLITE_OK {

    let SQLITE_TRANSIENT = unsafeBitCast(-1, to: sqlite3_destructor_type.self)

    if sqlite3_bind_text(insertStmt, 1, "test1", -1, SQLITE_TRANSIENT) != SQLITE_OK{
        let errmsg = String(cString: sqlite3_errmsg(db)!)
        print("Text Binding Failure: \(errmsg)")
        return
    }

    if sqlite3_step(insertStmt) == SQLITE_DONE {
        print ("Successfully inserted.")
    } else {
        print ("insert error")
    }

    sqlite3_finalize(insertStmt)

} else {
    print ("Insert statment is not prepared.")
}
```

OpaquePointer 를 반복 사용
하고자 할 때에는
sqlite3_reset(insertStmt) 와
같이 리셋 후 재사용

테이블 데이터 검색

📄 전체 또는 일부 검색

```
let sql = "select * from \$(TABLE_NAME) where \$(COL_NAME) = \$(name)"

var queryStmt: OpaquePointer?

if sqlite3_prepare(db, sql, -1, &queryStmt, nil) != SQLITE_OK {
    let errmsg = String(cString: sqlite3_errmsg(db)!)
    print("Reading Error : \$(errmsg)")
    return
}

while(sqlite3_step(queryStmt) == SQLITE_ROW) {
    let id = sqlite3_column_int(queryStmt, 0)
    let name = String(cString: sqlite3_column_text(queryStmt, 1))
    print("id: \$(id) name: \$(name)")
}

sqlite3_finalize(queryStmt)
```

테이블 데이터 수정

전체 또는 특정 항목의 데이터 수정

```
let query = "update \$(TABLE_NAME) set \$(COL_NAME) = ? where \$(COL_ID) = ?"

var updateStmt: OpaquePointer?

if sqlite3_prepare(db, query, -1, &updateStmt, nil) != SQLITE_OK {
    let errmsg = String(cString: sqlite3_errmsg(db)!)
    print("error preparing update: \(errmsg)")
    return
}

let SQLITE_TRANSIENT = unsafeBitCast(-1, to: sqlite3_destructor_type.self)

// prepared statement 매개변수 연결(binding)
if sqlite3_bind_text(updateStmt, 1, "my_id", -1, SQLITE_TRANSIENT) != SQLITE_OK {
    let errmsg = String(cString: sqlite3_errmsg(db)!)
    print("Text Binding Failure: \(errmsg)")
    return
}

if sqlite3_bind_int(updateStmt, 2, 1) != SQLITE_OK {
    let errmsg = String(cString: sqlite3_errmsg(db)!)
    print("Integer Binding Failure: \(errmsg)")
    return
}

if sqlite3_step(updateStmt) != SQLITE_DONE {
    let errmsg = String(cString: sqlite3_errmsg(db)!)
    print("Update Failure: \(errmsg)")
    return
}

sqlite3_finalize(updateStmt)
```

테이블 데이터 삭제

전체 또는 특정 데이터 삭제

```
let query = "delete from \((TABLE_NAME) where \((COL_ID) = ?"
var deleteStmt: OpaquePointer?

if sqlite3_prepare(db, query, -1, &deleteStmt, nil) != SQLITE_OK{
    let errmsg = String(cString: sqlite3_errmsg(db)!)
    print("error preparing stmt: \(errmsg)")
    return
}

bindParam(deleteStmt!, no: 1, param: id)

if sqlite3_step(deleteStmt) != SQLITE_DONE {
    let errmsg = String(cString: sqlite3_errmsg(db)!)
    print("Delete Failure: \(errmsg)")
    return
}

sqlite3_finalize(deleteStmt)
```



```
func bindTextParams(_ stmt: OpaquePointer, no: Int, param: String) {  
    let SQLITE_TRANSIENT = unsafeBitCast(-1, to: sqlite3_destructor_type.self)  
  
    if sqlite3_bind_text(stmt, Int32(no), param, -1, SQLITE_TRANSIENT) != SQLITE_OK{  
        let errmsg = String(cString: sqlite3_errmsg(db))  
        print("Text Binding Failure: \(errmsg)")  
        return  
    }  
}  
  
func bindIntParams(_ stmt: OpaquePointer, no: Int, param: Int) {  
    if sqlite3_bind_int(stmt, Int32(no), Int32(param)) != SQLITE_OK{  
        let errmsg = String(cString: sqlite3_errmsg(db))  
        print("Integer Binding Failure: \(errmsg)")  
        return  
    }  
}
```

테이블 삭제 및 데이터베이스 닫기

테이블 삭제

```
if sqlite3_exec(db, "drop table if exists \$(TABLE_NAME)", nil, nil, nil) != SQLITE_OK {
    let errmsg = String(cString: sqlite3_errmsg(db)!)
    print("Drop Error: \$(errmsg)")
    return
}
```

데이터베이스 닫기

```
if sqlite3_close(db) != SQLITE_OK {
    let errmsg = String(cString: sqlite3_errmsg(db)!)
    print("Database Close Error: \$(errmsg)")
    return
}
```

고려 사항

📖 **DB**의 명령 각각을 함수로 만들어 재사용할 수 있도록 구성

📖 **DB** 기능을 담당하는 별도의 클래스를 작성하여 **DB** 기능을 갖도록 구성 → **DBManager**

📖 데이터를 담는 **DTO** 클래스 사용

```
class NameDto {
    var id: Int
    var name: String?

    init(id: Int, name: String?) {
        self.id = id
        self.name = name
    }
}
```

 각 **DB** 기능을 함수로 분리하여 작성해보기

 **12**장의 테이블 뷰 예제에 **DB** 를 적용하여 보기

1. 테이블 뷰의 항목을 DB에서 가져와 보여주기
2. ADD 를 할 때 DB 에 추가하기
3. 삭제하기 수행 시 DB에서 삭제하기