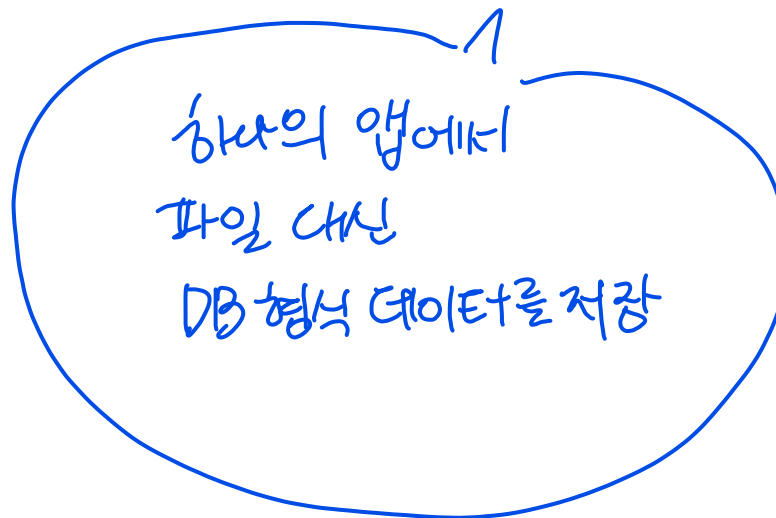


자료 보관 (앱이 종료된 시점 이후 다시 앱 실행시켰을 때도
자료가 영구적으로 유지될 수 있게)

- Shared Preference
- 파일 (가장 기본적으로 모든 프로그램이 공통적으로 사용함)

데이터베이스 활용

- SQLite → 앱 자체에서
앱 전용 데이터를 만든다.
↓
Local



목차


SQLite

DB 사용 절차

DB 준비 - SQLiteOpenHelper 클래스

DB 사용 - SQLiteDatabase/Cursor 클래스

원시적
방법



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

→ 앱 유지되는 동안!

→ 앱 삭제시 데이터 날라감!

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



SQLiteOpenHelper
상속 클래스 작성

```
class MyDBHelper : SQLiteOpenHelper (...){  
    MyDBHelper() // 생성자  
    onCreate()   // 테이블 생성  
    onUpgrade()  // DB 수정이 필요할 경우  
}
```

필수!

helper 객체 생성

```
val helper = MyDBHelper (...)
```

helper를 사용하여
SQLiteDatabase 객체 획득

```
val writableDB = helper.writableDatabase  
// insert, update, delete  
or  
val readableDB = helper.readableDatabase  
// select
```

SQLiteDatabase 객체를
사용하여 **Query** 수행

• 메소드 사용 방식 → 권장

writableDB. insert()	// 자료 저장 시
writableDB. update()	// 자료 수정 시
writableDB. delete()	// 자료 삭제 시
readableDB. query()	// 자료 검색 시

필요에
따라 선택

helper 객체(및 관련
객체) **Close** 수행

• SQL 문 직접 작성 방식

writableDB. execSQL()	// insert, update, delete 사용 시
readableDB. rawQuery()	// select 사용 시

```
helper.close()  
// Cursor 를 사용하였을 경우 Cursor.close() ← select 사용 시
```

• DB 사용이
필요한 곳마다
해당 코드 작성

SQLiteOpenHelper

☞ 데이터베이스를 편리하게 사용할 수 있도록 도와주는 클래스 → 상속하여 사용

- ◆ 데이터베이스 저장 파일 생성
- ◆ 테이블 생성
- ◆ 테이블 업그레이드, 기본 샘플 데이터 추가 등
- ◆ SQLiteDatabase 객체 제공

☞ 필수 재정의 메소드

- ◆ 생성자: 사용할 DB 파일 명 및 DB 버전을 지정
- ◆ onCreate()
 - 사용할 테이블을 SQL을 사용하여 생성
 - 샘플이 필요할 경우 테이블 생성 후 샘플 추가 문장 작성
- ◆ onUpgrade()
 - 테이블 구조를 변경해야 할 필요가 있을 때 사용. 특별히 재정의하지 않아도 무방

SQLiteOpenHelper 상속 클래스 작성

• FoodDBHelper

• SQLiteOpenHelper 상속
- context, DB명, cursorFactory, DB ver.

```

11 class FoodDBHelper(context: Context?) : SQLiteOpenHelper(context, DB_NAME, null, 1) {
12     val TAG = "FoodDBHelper"
13
14     companion object {
15         const val DB_NAME = "food_db"
16         const val TABLE_NAME = "food_table"
17         const val COL_FOOD = "food"
18         const val COL_COUNTRY = "country"
19     }
20
21     override fun onCreate(db: SQLiteDatabase?) {
22         val CREATE_TABLE =
23             "CREATE TABLE ${TABLE_NAME} (${BaseColumns._ID} INTEGER PRIMARY KEY AUTOINCREMENT, " +
24             "${COL_FOOD} TEXT, ${COL_COUNTRY} TEXT )"
25         Log.d(TAG, CREATE_TABLE)
26         db?.execSQL(CREATE_TABLE)
27     }
28
29     override fun onUpgrade(db: SQLiteDatabase?, oldVer: Int, newVer: Int) {
30         val DROP_TABLE = "DROP TABLE IF EXISTS ${TABLE_NAME}"
31         db?.execSQL(DROP_TABLE)
32         onCreate(db)
33     }
34 }

```

• DB 및 테이블, 테이블 컬럼명 등을 companion object 로 생성하여 보관 → 해당 항목명의 사용 일관성을 위해

• 최초로 readableDatabase 또는 writableDatabase 사용 시 호출
• DB의 테이블 생성 및 샘플 데이터 추가

• BaseColumns 인터페이스의 **_ID** 속성(_id) 지정
→ 다른 안드로이드 요소에서 필요한 경우가 있음

• 새로운 버전의 DB를 사용할 필요가 있을 때 사용

Helper 객체 생성

DB 사용 부분에서 객체 생성

Helper 객체 생성 시의 동작

- ◆ 생성자에 의해 DB파일이 안드로이드 내부저장소에 생성
- ◆ 실기기는 보안 문제로 폴더 외부 접근이 안되므로 에뮬레이터 상에서만 확인 가능
- ◆ [Device File Explorer] 사용
 - 에뮬레이터 선택 후 [File Explorer] 선택
- data/data/**PACKAGE_NAME**/databases/**DATABASE_NAME**
- 예) data/data/**mobile.example.dbtest**/databases/**food_db**

• DB 파일은 컴퓨터에 다운받을 수 있으며, 다운받은 파일은 SQLiteBrowser 프로그램을 사용하여 내용 확인 가능

SQLiteDatabase

☞ Helper 클래스에 의해 관리되는 데이터베이스 클래스

◆ DB 작업 Query 를 수행

CRUD

☞ Helper 클래스를 사용하여 획득

◆ 읽기 전용 (select)

val myDB : SQLiteDatabase = myDBHelper.*readableDatabase*

◆ 읽기/쓰기 겸용 (insert/update/delete) lock 걸어야 함!

val myDB : SQLiteDatabase = myDBHelper.*writableDatabase*

☞ SQL을 직접 사용하거나 관련 메소드를 사용하여
Query 수행

☞ 모든 작업 수행 후 Helper 객체를 통해 반드시 **close()**
하여 종료

SQLiteDatabase Query 1

AUTOINCREMENT
자동으로 값이 들어감

동덕여자대학교

food_table

<u>_id</u>	food	<u>country</u> nation
1	된장찌개	한국

Query 전용 함수 사용 or SQL 사용

- 전용함수 사용은 Content Provider 사용과 동일

데이터 삽입 - insert() or execSQL()

- ContentValues 객체 생성 후 입력할 데이터 값 설정 후 insert() 사용 → 반환값이 있으므로 삽입 결과 확인 가능
- execSQL()를 사용하여 SQL 직접 작성 후 수행

•전용함수 사용방식

```
// helper 는 FoodDBHelper 객체
val db = helper.writableDatabase
val newRow = ContentValues()
newRow.put("food", "된장찌개")
newRow.put("country", "한국")
db.insert("food_table", null, newRow)
helper.close()
```

•SQL 직접 사용방식

```
db.execSQL("INSERT INTO food_table " +
    " VALUES (NULL, '된장찌개', '한국')")
```

SQL 명령어를 직접 작성

테이블 이름

문자열 표시 시
' ' 사용 주의!!!

(테이블, null, 새로운 값)

- 수행 완료 후 helper.close() 호출

SQLiteDatabase Query 2

데이터 수정 – update() or execSQL()

- ◆ ContentValues 객체에 변경할 값 및 조건 설정 후 update() 사용
- ◆ execSQL() 메소드를 사용하여 SQL 직접 작성 후 수행

•전용함수 사용방식

```
val db = helper.getWritableDatabase
```

```
val updateRow = ContentValues()
```

```
updateRow.put("country", "한국")
```

```
val whereClause = "food=?" // 수정할 row 검색 조건 - 여러 개일 경우 A=? and B=? ...
```

```
val whereArgs = arrayOf("된장찌개") // 검색조건의 ? 와 결합할 값 조건이 여러 개일 수도 ...
```

```
db.update("food_table", updateRow, whereClause, whereArgs)
```

```
helper.close()
```

food_table

_id	food	nation
1	된장찌개	한국

•SQL 직접 사용방식

```
db.execSQL("UPDATE food_table " +
```

```
" SET country='한국' WHERE food='된장찌개'")
```

↓
바꿀것

↓
조건

SQLiteDatabase Query 3

데이터 삭제 – delete() or execSQL()

- ◆ 삭제대상 검색 조건 지정 후 delete() 수행
- ◆ execSQL 메소드를 사용하여 SQL 직접 작성 후 수행

• 전용함수 사용방식

```
val db = helper.writableDatabase
val whereClause = "food=?"
val whereArgs = arrayOf("된장찌개")
db.delete("food_table", whereClause, whereArgs)
helper.close()
```

food_table

_id	food	nation
1	된장찌개	한국

• SQL 직접 사용방식

```
db.execSQL("DELETE FROM food_table WHERE food='된장찌개'")
```

SQLiteDatabase Query 4

데이터 검색 – query() or.rawQuery()

- ◆ 조건문 지정 후 query() 수행
- ◆ rawQuery 메소드를 사용하여 SQL 직접 수행
- ◆ Cursor(데이터 반환 집합에 대한 레퍼런스) 값을 반환
- ◆ 사용 후 Helper 및 Cursor 는 반드시 close() 적용

•전용함수 사용방식

```
val db = helper.readableDatabase
val columns = arrayOf("_id", "food", "country") // 검색할 컬럼명, null 일 경우 모든 컬럼
val selection = "food=?" // 검색 조건
val selectionArgs = arrayOf("된장찌개") // 검색 조건 ? 에 결합할 값
val cursor = db.query(
    "food_table", columns, selection, selectionArgs,
    null, null, null, null
)
```

•읽기전용 사용 가능

groupBy, having, orderBy, limit

→ null

*

•SQL 직접 사용방식

```
val cursor = db.rawQuery("SELECT _id, food, country FROM" +
    " food_table WHERE food='된장찌개'", null)
```

?

배열

SQLiteDatabase Query 5

Cursor

- ◆ select 문에 의해 반환한 레코드의 집합을 가리킴
- ◆ Cursor.moveToNext() : 다음 레코드가 있으면 true, 없으면 false
- ◆ Cursor.getType(column_index)를 사용하여 값을 읽어옴

```
// DB 검색 결과를 DTO 에 저장하여 List에 보관하고자 할 경우
// val foodList = arrayListOf<FoodDto>()
```

```
with(cursor) { this: Cursor!
```

• with(T) { } : T 객체가 this 역할

```
while (moveToNext()) {
```

```
    val id = getInt( getColumnIndex("_id"))
```

```
    val food = getString( getColumnIndex("food"))
```

```
    val country = getString ( getColumnIndex("country"))
```

```
    Log.d(TAG, "$id - $food ( $country )")
```

```
    // 결과를 DTO 객체에 저장후 List에 보관
```

```
    // val dto = FoodDto(id, food, country)
```

```
    // foodList.add(dto)
```

```
}
```

```
}
```

• Cursor.getColumnIndex()
- 컬럼명으로 컬럼의 순서(index)를 알아냄
Cursor.getColumnIndex ("food") == 1

0 1 food_table 2

_id	food	nation
1	된장찌개	한국

cursor →

↓ select 결과

1	된장찌개	한국
---	------	----

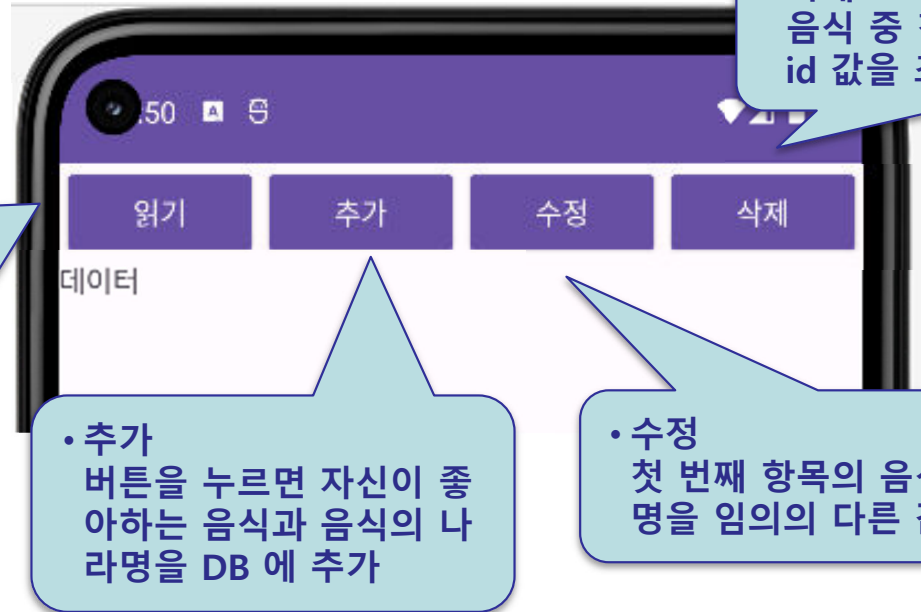
```
cursor.close()
helper.close()
```

• 사용 완료 후 cursor, helper
순으로 close()호출

다음 DB 사용 앱을 완성하시오.

◆ Exam01.zip 사용

- 읽기
Database 에 저장하고 있는 모든 레코드를 Log 창에 출력
- 샘플 데이터 → DBHelper 의 onCreate() 부분에서 미리 추가할 것
예)
불고기 (한국)
비빔밥 (한국)
휘귀 (중국)
딤섬 (중국)
스시 (일본)
오코노미야키 (일본)

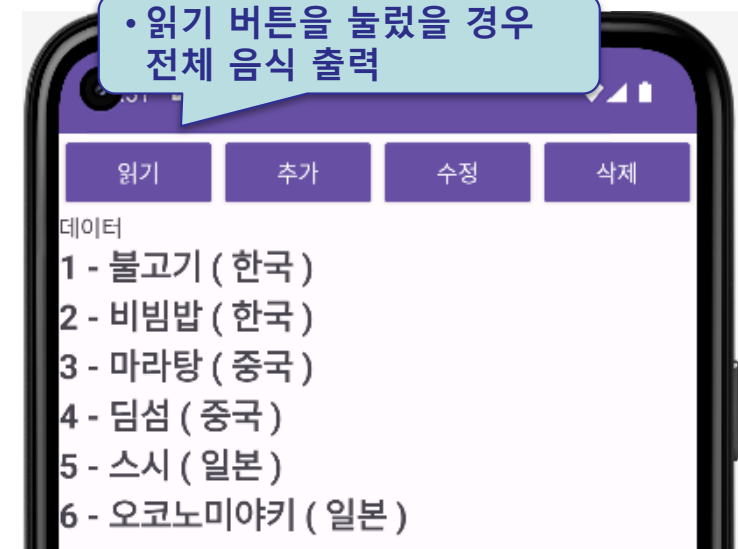


- 삭제
음식 중 첫 번째 항목을 id 값을 조건으로 삭제

- 추가
버튼을 누르면 자신이 좋아하는 음식과 음식의 나라명을 DB 에 추가

- 수정
첫 번째 항목의 음식명과 나라명을 임의의 다른 값으로 변경

- 읽기 버튼을 눌렀을 경우
전체 음식 출력



◆ 추가/수정/삭제 시 코드로 작성한 값 사용

실습 2

실습 1의 앱을 수정하여 다음과 같이 구성하시오.

- ◆ 추가, 수정, 삭제 를 누를 경우 새로운 Activity 를 띄워 값을 입력 받은 후 DB 작업을 수행
- ◆ 각 Activity 에서 DB 작업 후 MainActivity 에서 읽기를 눌러 확인

