가천대학교
Gachon University

학생 여러분 반갑습니다.

다른 친구들이 입장할 때까지 조금 기다려 주십시오.

곧 모바일 프로그래밍 수업을 시작합니다.

음소거( )가 되었는지 확인 바랍니다.

모바일 프로그래밍
화목(1,2교시)/ 화목(3,4교시)
정윤현 (AI/소프트웨어학부)

Mobile Programming

Android Programming

Chap 8-2. SQLite (Database)

Prof. Younhyun Jung

Email) younhyun.jung@gachon.ac.kr

가천대학교 소프트웨어학과
Gachon University

Android Data Storage

- Android provides several options to save persistent application data.
  - The solution you choose depends on your specific needs.

- Data storage options :
  - **Shared Preferences :**
    - Good for private small datasets in key-value pairs.
  - **Internal Storage :**
    - Store private data on the device's memory.
  - **External Storage:**
    - Store public data on the shared external storage.
  - **SQLite Databases:**
    - **Store structured data in a database.**
  - **+ Network Connection:**
    - Store data on the web with your own network server.

Introduction

- If you want to store the test results of all the students in a school,
- How to manage the sets of data in Android Apps?
- It is much more efficient to use a database to represent them because you can use **database querying** to retrieve the results of specific students.
- Generally, for saving relational data, using a database is much more efficient.

↳ 구조화된 데이터 효율적

Overview

- Rapid and efficient data storage and retrieval are essential
- **Android** provides a lightweight relational database for each application using **SQLite**.
- Your applications can take advantage of the managed relational database engine to store data securely and efficiently.

SQLite

- SQLite
  - The most widely deployed SQL database engine in the world
  - A well-regarded **relational database management system** (RDBMS)
    - occupies a small amount of disk storage and memory, so it's a perfect choice for creating databases on many mobile operating systems such as Android, iOS.
    - SQLite has a reputation of being extremely reliable and is the database system of choice for many consumer electronic devices,
      - including several MP3 players, the iPhone, and the iPod Touch

↳ 굉장히 제한적인 환경에서도 / 신뢰성 ↑

Documentation on SQLITE available at
http://www.sqlite.org/sqlite.html

Two SQLite Libraries

1 SQLiteDatabase

- Exposes methods to manage a SQLite database.
  - to create, delete, execute SQL commands, and perform other common database management tasks.
- Package : android.database.sqlite.SQLiteDatabase

2 SQLiteOpenHelper

관리를 자동으로 해줌!

- A helper class to manage database creation and version management.
- an abstract class used to implement the best practice pattern for creating, opening, and upgrading databases.

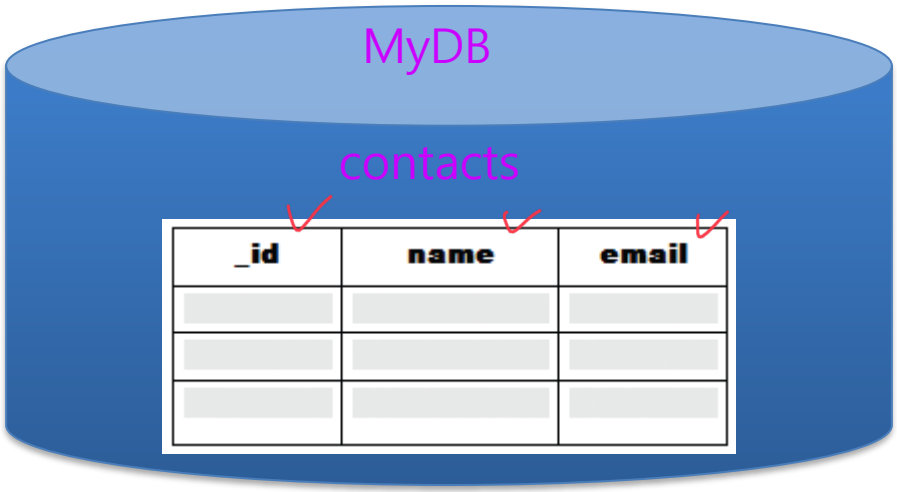
상속받아서 처리.

  - You create a subclass implementing onCreate(SQLiteDatabase), onUpgrade(SQLiteDatabase, int, int) and optionally onOpen(SQLiteDatabase)
- Package: android.database.sqlite.SQLiteOpenHelper

Example: Using SQLiteOpenHelper

Example:

- Create a **database** named MyDB containing one **table** named contacts. This **table** will have three columns:
  _id, name, and email



Example: Using SQLiteOpenHelper

```
public class YourDatabaseHelper extends SQLiteOpenHelper {
        static final String KEY_ROWID = "_id";
        static final String KEY_NAME = "name";
        static final String KEY_EMAIL = "email";

        static final String DATABASE_NAME ="MyDB";
        static final String DATABASE_TABLE = "contacts";
        static final String DATABASE_VERSION = 1;
        // SQL Statement to create a new database
        static final String DATABASE_CREATE =
            "create table " +  DATABASE_TABLE + " ("
             + KEY_ROWID + " integer primary key autoincrement, "
            + "name text not null, email text not null);";

        public YourDatabaseHelper(Context context) {
                super(context, DATABASE_NAME, null,
                                DATABASE_VERSION);
        }
        …
}
```

Your necessary info

create table query.

Specify DB name in the constructor

onCreate(SQLiteDatabase db)

2

```
@Override
public void onCreate(SQLiteDatabase db) {
   // TODO Auto-generated method stub
     try {
            db.execSQL(DATABASE_CREATE);
     } catch (SQLException e) {
            e.printStackTrace();
     }
}
```

The onCreate() method creates a new database **if the required database is not present**

현재 DB 없을 때 실행됨!

- Called when no database exists in disk and the helper class needs to create a new one.
- The first time you try to access a database (e.g., using **getWritableDatabase()**), SQLiteOpenHelper will notice it doesn't exist and call the onCreate( ) method to create it.
- In this example, we override onCreate() that and run a CREATE TABLE SQL statement.

onUpdate(SQLiteDatabase db)

3

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion,  int newVersion) {
   db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
   onCreate(db);
}
```

- Called when there is a database version mismatch meaning that the version of the database on disk needs to be upgraded to the current version.
- In this example, we just *simply drops the existing table and replaces it with the new definition*, but you could do something smarter here if you like.

Pattern: Using SQLiteOpenHelper

1. 0. You create a subclass of ***SQLiteOpenHelper***
   - Constructor
     - Call super() to initialize underlying database
2. Override onCreate()
   - We can create tables
3. (optional) can also override onUpgrade()
   - make a modification to the tables structures

- **Execute** operations on underlying SQLite database
  - * Use execSQL() to execute SQL Statements
  - * Call `getReableDatabase()` / `getWritableDatabase()` to **get read/write <u>an instance</u>** of ***<u>SQLiteDatabase</u>***
    - Can then call query(), insert(), delete(), update()

직접 Query 날리거나

함수를 사용!

```
SQLiteDatabase db = YourDBHelper.getWritableDatabase();
// then, db.insert( … ); …
```

Another Example

```
public class DataBaseHelper extends SQLiteOpenHelper {

        private static final String  DATABASE_NAME = "MyDB";
        static final String DATABASE_TABLE = "contacts";
        static final int DATABASE_VERSION = 1;
        static final String DATABASE_CREATE =
                "create table contacts (_id integer primary key autoincrement, "
                + "name text not null, email text not null);";

        static final String TAG = "DataBaseHelper";

        public DataBaseHelper(Context context) {
                super(context, DATABASE_NAME, null, DATABASE_VERSION);;
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
                try {
                        db.execSQL(DATABASE_CREATE);
                } catch (SQLException e) {
                        e.printStackTrace();
                }
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                Log.w(TAG, "Upgrading database from version " + oldVersion + " to
"
                + newVersion + ", which will destroy all old data");
                db.execSQL("DROP TABLE IF EXISTS contacts");
                onCreate(db);
        }
}
```

1

2

3

ContentValues AND Cursors

- **ContentValues**
  - used to insert new rows into tables.
  - Each *ContentValues object* represents a single table row as a map of column names to values.
  - Example:

```
ContentValues initialValues = new ContentValues();
initialValues.put(KEY_NAME, name);
initialValues.put(KEY_EMAIL, email);
db.insert(DATABASE_TABLE, null, initialValues);
```

Default : null → values에 값이 없는 경우 행 추가 안 함

```
//static final String KEY_NAME = "name";
//static final String KEY_EMAIL = "email";
//static final String DATABASE_TABLE = "contacts";
```

Content Values AND Cursors

- **Cursors**
  - Database queries are returned as Cursor objects.
    - Note: returned data is "a result set" (not a single value)
  - Rather than extracting and returning a copy of the result values, Cursors are "pointers" to the result set within the underlying data.
  - Cursors provide a managed way of controlling your position (row) in the result set of a database.
    - Cursor class includes a number of navigation functions
      - E.g., moveToFirst(), moveToNext(), moveToPrevious(), …

```
if (c.moveToFirst()) {
        do {
                        // do something for c;
         } while (c.moveToNext());
}
```

Cont.

```
// Specify the result column projection. Return the minimum set
// of columns required to satisfy your requirements.
String[] result_columns = new String[] { KEY_ID, KEY_NAME, KEY_EMAIL };
// Specify the where clause that will limit our results.
String where = KEY_ACCESSIBLE_COLUMN + "=" + 1;
// Replace these with valid SQL statements as necessary.
String whereArgs[] = null;
String groupBy = null;
String having = null;
String order = null;
SQLiteDatabase db = myDBOpenHelper.getWritableDatabase();
Cursor cursor = db.query(HoardDBOpenHelper.DATABASE_TABLE,
                         result_columns, where,
                         whereArgs, groupBy, having, order);
```

- table : 쿼리를 수행할 테이블 이름입니다.
- columns : 자료를 받아올 필드입니다. null을 입력하면 모든 필드를 반환합니다.

[extra value]

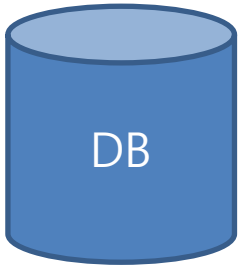
- selection : SQL의 "where" 구문에 해당하는 조건과 값을 입력합니다. 조건이 많은 경우, 조건의 값들을 ?로 대체한 후 selectionArgs에서 값을 지정해 줄 수 있습니다.
- selectionArgs : selection에서 ?로 지정했던 조건의 값들을 입력합니다.
- groupBy : SQL의 "group by" 구문에 해당합니다.
- having : groupBy를 지정했을 경우, 조건을 넣어줍니다.
- orderBy : 결과값의 정렬 방식을 지정합니다. null을 입력하면 주요 키를 기준으로 하여 오름차순 정렬을 수행합니다.

Cont.

- Example

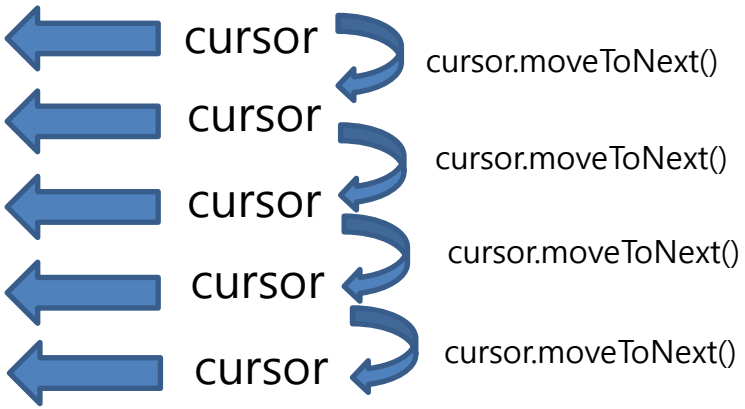
```
SQLiteDatabase sqlitedb;
Cursor cursor = sqlitedb.query("customers", null, null, null, null, null, null);
```

select * from customers;



"customers" table

Radu	Male	SMS	e-mail
Winnie	Female	SMS	
David	Male		
Ayaz	Male	SMS	e-mail
Brandi	Female		e-mail




```
while(cursor.moveToNext())
…
String name = cursor.getString(0);
String gender = cursor.getString(1);
…
```

Cont.

- **The Cursor class** includes a number of navigation functions, including, but not limited to, the following:
  - **moveToFirst** — Moves the cursor to the first row in the query result
  - **moveToNext** — Moves the cursor to the next row
  - **moveToPrevious** — Moves the cursor to the previous row
  - **getCount** — Returns the number of rows in the result set
  - **getColumnIndexOrThrow** — Returns the zero-based index for the column with the specified name (throwing an exception if no column exists with that name)
  - **getColumnName** — Returns the name of the specified column index
  - **getColumnNames** — Returns a string array of all the column names in the current Cursor
  - **moveToPosition** — Moves the cursor to the specified row
  - **getPosition** — Returns the current cursor position

etc...(update/delete)

- Managing Database
  - Update items from Database

whereCluase가 whereArgs인 곳을 values로 업데이트해라

```
int update (String table, ContentValues values, String whereClause, String[] whereArgs)
```

Parameters	
table	String: the table to update in
values	ContentValues: a map from column names to new column values. null is a valid value that will be translated to NULL.
whereClause	String: the optional WHERE clause to apply when updating. Passing null will update all rows.
whereArgs	String: You may include ?s in the where clause, which will be replaced by the values from whereArgs. The values will be bound as Strings.

whereCluase가 whereArgs인 곳을 delete 해라

  - Delete items from Database

```
int delete (String table, String whereClause, String[] whereArgs)
```

"name = ?"
and name = ?"
{"kelly", "Toag"}

Parameters	
table	String: the table to delete from
whereClause	String: the optional WHERE clause to apply when deleting. Passing null will delete all rows.
whereArgs	String: You may include ?s in the where clause, which will be replaced by the values from whereArgs. The values will be bound as Strings.

Exercise

- Make an MainActivity.java file

```
public class MainActivity extends AppCompatActivity {

    SQLiteDatabase db;
    MySQLiteOpenHelper helper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        helper = new MySQLiteOpenHelper(MainActivity.this, "person.db", null, 1);
        insert("유저1", 18, "홍길동");
        insert("유저2", 28, "홍길동2");
        insert("유저3", 28, "홍길동3");

        update("유저1", 58);
        delete("유저2");
        select();
    }

    public void insert(String name, int age, String address) {
        db = helper.getWritableDatabase();
        ContentValues values = new ContentValues();

        values.put("name", name);
        values.put("age", age);
        values.put("address", address);
        db.insert("student", null, values);
    }
```

```
package org.androidtown.listview.mydatabase;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.util.Log;
```


[출처] Android – SQLite 활용 예제1 | 작성자 장스

Exercise

- Make an MainActivity.java file

```
    public void update (String name, int age) {
        db = helper.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put("age", age);
        db.update("student", values, "name=?", new String[]{name});
    }

    public void delete (String name) {
        db = helper.getWritableDatabase();
        db.delete("student", "name=?", new String[]{name});
        Log.i("db1", name + "정상적으로 삭제 되었습니다.");
    }
    public void select() {

        db = helper.getReadableDatabase();
        Cursor c = db.query("student", null, null, null, null, null, null);
        /* query (String table, String[] columns, String selection, String[]
         * selectionArgs, String groupBy, String having, String orderBy)
         */
        while (c.moveToNext()) {

            int _id = c.getInt(c.getColumnIndex("_id"));
            String name = c.getString(c.getColumnIndex("name"));
            int age = c.getInt(c.getColumnIndex("age"));
            String address = c.getString(c.getColumnIndex("address"));
            Log.i("db1", "id: " + _id + ", name : " + name + ", age : " + age
                    + ", address : " + address);
        }
    }
}
```

select * from student.

[출처] Android – SQLite 활용 예제1 | 작성자 장스

Exercise

- Make an MySQLiteOpenHelper

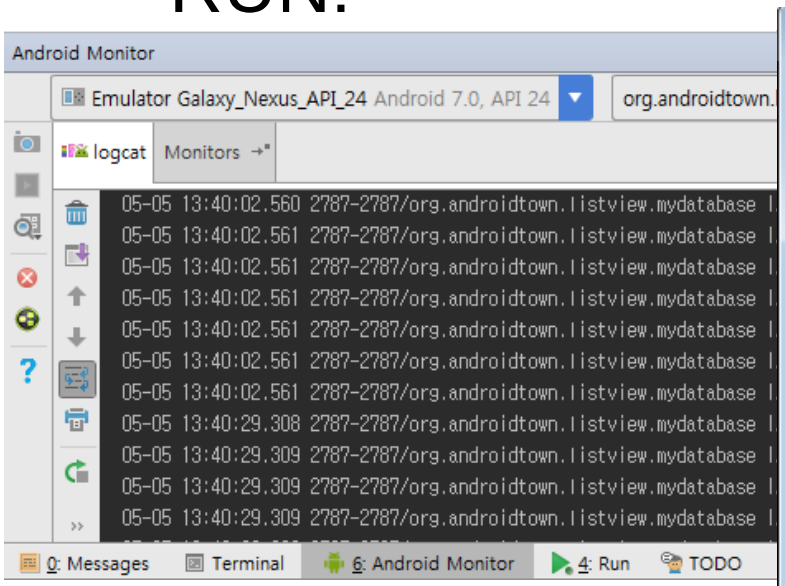
```
public class MySQLiteOpenHelper extends SQLiteOpenHelper {

    public MySQLiteOpenHelper(Context context, String name,
                              SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
        // TODO Auto-generated constructor stub

    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // TODO Auto-generated method stub
        String sql = "create table student (" +
                "_id integer, " +
                "name text, " +
                "age integer, " +
                "address text);";
        db.execSQL(sql);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

        String sql = "drop table if exists student";
        db.execSQL(sql);
        onCreate(db);
    }
}
```

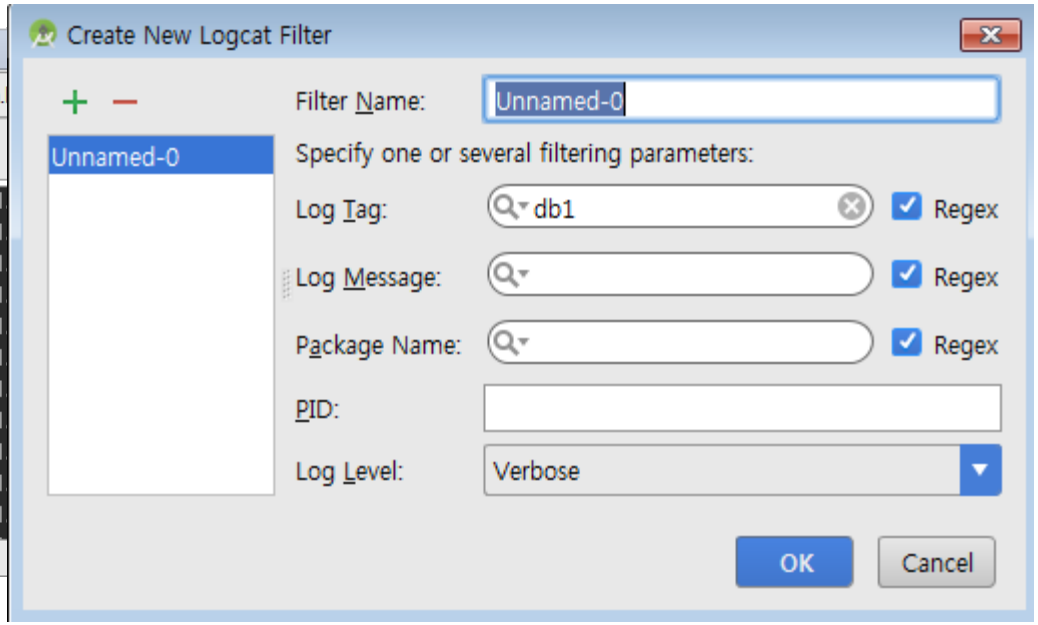
```
package org.androidtown.listview.mydatabase;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
```


[출처] Android – SQLite 활용 예제1 | 작성자 장스

Exercise

- RUN!






```
05-05 13:40:02.560 2787-2787/org.androidtown.listview.mydatabase I/db1: 유저2정상적으로 삭제 되었습니다.
05-05 13:40:02.561 2787-2787/org.androidtown.listview.mydatabase I/db1: id: 1, name : 유저1, age : 58, address : 홍길동
05-05 13:40:02.561 2787-2787/org.androidtown.listview.mydatabase I/db1: id, :3 name : 유저3, age : 28, address : 홍길동3
```