ITSMAP F13 Lesson 8 part 2

SQLite + Database Adapter

Jesper Rosholm Tørresø

SQLite RDBMS

- Zero-Configuration
- Single Database File
- Compact (275Kb)
- Android default Database engine is Lite. SQLite is a lightweight transactional database engine
- Occupies a small amount of disk storage and memory,
- Perfect choice for creating relational databases on many mobile operating systems such as Android, iOS.
- SQL statements compiled into virtual machine code

http://www.sqlite.org/different.html

Things to consider when dealing with SQLite:

- **Data type integrity is not maintained** in SQLite, you can put a value of a certain data type in a column of another data type (put string in an integer and vice versa).
- **Referential integrity is NOT default maintained** in SQLite, for **FOREIGN KEY** constraints or JOIN statements se links below.
- **SQLite Full Unicode support is optional** and not installed by default.

  http://www.sqlite.org/foreignkeys.html
  http://alvinalexander.com/android/sqlite-foreign-keys-example
  http://panierter-pinguin.de/blog/?p=138
  http://zetcode.com/db/sqlite/joins/

SQLite types:

- NULL, INTEGER, REAL, TEXT and BLOB
- Type affinity
  - In order to maximize compatibility between SQLite and other database engines, SQLite supports the concept of "type affinity" on columns. The type affinity of a column is the recommended type for data stored in that column. The important idea here is that the type is recommended, not required. Any column can still store any type of data. It is just that some columns, given the choice, will prefer to use one storage class over another. The preferred storage class for a column is called its "affinity".
- Primary key ID – auto increment recommended! (Ascending or descending)

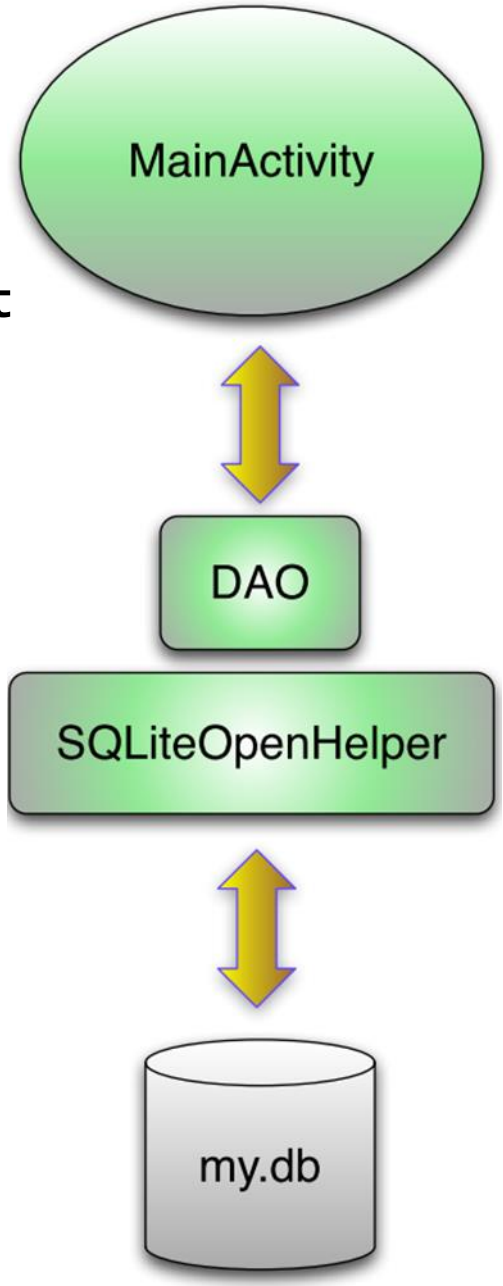
Android App and SQLite

- When a structured access to data collections is needed or if an existing database (i.e. file like database.db) follows the App.
- To ways to set up a database
  - Importing an existing database (file)
    - Supported by tools that generates adapters, db file and if necessary a ContentProvider.
  - Create database inside the App
    - Creation of db file coded by you together with adapters etc

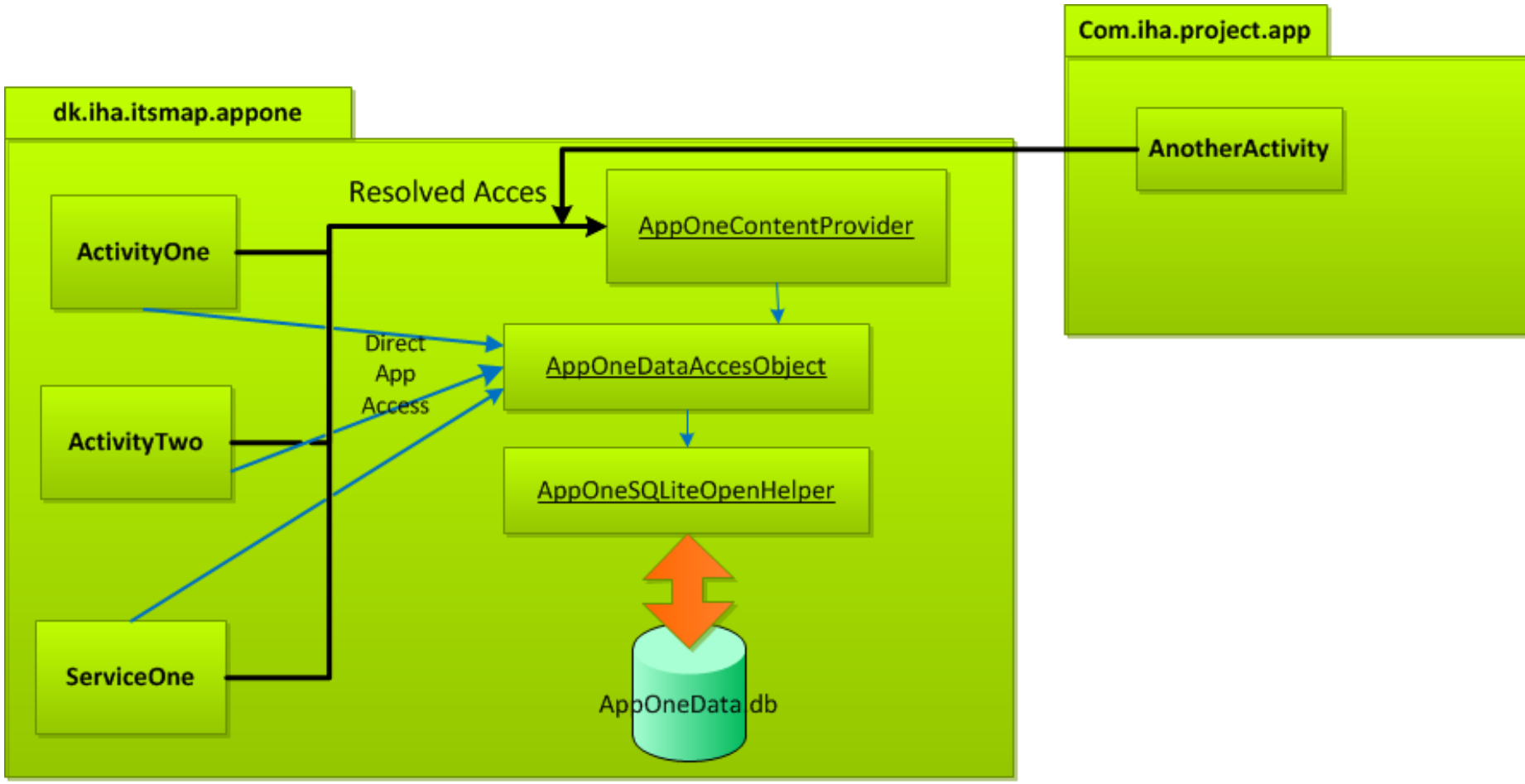
Architecture SQLite

Classes to deal with

- **`dk.iha.itsmap.appone.AppOneDataAccessObject`**
  - Your own adapter for CRUD operations on the database
- **`android.database.sqlite.SQLiteOpenHelper`**
  - Your own helper Class for creating, opening and upgrading the database (Abstract class, you implement the App specific part)
- **`android.database.sqlite.SQLiteDatabase`**
  - SQLite Database driver
- **`android.database.Cursor;`**
  - A Cursor implementation that exposes results from a query on a SQLiteDatabase. (AKA a result set) .



DB Arch. + ContentProvider




```
private static String DB_PATH = "/data/data/dk.iha.itsmap.appone/databases/";
```

Android Framework

Skeleton for DAO and SQLiteOpenHelper 1

http://kurser.iha.dk/eit/jem1/Basic/Android/Professional%20Android%202%20Code%20Listings/Chapter_7.txt

```
package dk.iha.itsmap.themapp;


//** Listing 7-1: Skeleton code for a standard database adapter implementation
import android.content.Context;
import android.database.*;
import android.database.sqlite.*;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.util.Log;

public class MyDAO {
private static final String DATABASE_NAME = "myDatabase.db";
private static final String DATABASE_TABLE = "mainTable";
private static final int DATABASE_VERSION = 1;
```

Skeleton for DAO and SQLiteOpenHelper 2

http://kurser.iha.dk/eit/jem1/Basic/Android/Professional%20Android%202%20Code%20Listings/Chapter_7.txt

```
// The index (key) column name for use in where clauses.
public static final String KEY_ID="_id";

// The name and column index of each column in your database.
public static final String KEY_NAME="name";
public static final int NAME_COLUMN = 1;
// TODO: Create public field for each column in your table.

// SQL Statement to create a new database.
private static final String DATABASE_CREATE = "create table " +
 DATABASE_TABLE + " (" + KEY_ID +
 " integer primary key autoincrement, " +
 KEY_NAME + " text not null);";
```

Skeleton for DAO and SQLiteOpenHelper 3

http://kurser.iha.dk/eit/jem1/Basic/Android/Professional%20Android%202%20Code%20Listings/Chapter_7.txt

```
// Variable to hold the database instance
private SQLiteDatabase db;
// Context of the application using the database.
private final Context context;
// Database open/upgrade helper
private myDbHelper dbHelper;

public MyDAO(Context _context) {
 context = _context;
 dbHelper = new myDbHelper(context, DATABASE_NAME, null,
DATABASE_VERSION);
}

public MyDAO open() throws SQLException {
```

Skeleton for DAO and SQLiteOpenHelper 4

http://kurser.iha.dk/eit/jem1/Basic/Android/Professional%20Android%202%20Code%20Listings/Chapter_7.txt

```
 db = dbHelper.getWritableDatabase();
 return this;
}

public void close() {
   db.close();
}

public int insertEntry(MyObject _myObject) {
 // TODO: Create a new ContentValues to represent my row
 // and insert it into the database.
 return index;
}
```

Skeleton for DAO and SQLiteOpenHelper 5

http://kurser.iha.dk/eit/jem1/Basic/Android/Professional%20Android%202%20Code%20Listings/Chapter_7.txt

```
public boolean removeEntry(long _rowIndex) {
 return db.delete(DATABASE_TABLE, KEY_ID + "=" + _rowIndex,
null) > 0;
}

public Cursor getAllEntries () {
 return db.query(DATABASE_TABLE, new String[] {KEY_ID,
        KEY_NAME}, null, null, null, null, null);
}

public MyObject getEntry(long _rowIndex) {
 // TODO: Return a cursor to a row from the database and
 // use the values to populate an instance of MyObject
 return objectInstance;
}
```

Skeleton for DAO and SQLiteOpenHelper 6

http://kurser.iha.dk/eit/jem1/Basic/Android/Professional%20Android%202%20Code%20Listings/Chapter_7.txt

```
public boolean updateEntry(long _rowIndex, MyObject _myObject) {
 // TODO: Create a new ContentValues based on the new object
 // and use it to update a row in the database.
 return true;
}
```

Skeleton for DAO and SQLiteOpenHelper 7

http://kurser.iha.dk/eit/jem1/Basic/Android/Professional%20Android%202%20Code%20Listings/Chapter_7.txt

```
private static class myDbHelper extends SQLiteOpenHelper {

 public myDbHelper(Context context, String name,
                   CursorFactory factory, int version) {
   super(context, name, factory, version);
 }

// Called when no database exists in disk and the helper class needs
 // to create a new one.
 @Override
 public void onCreate(SQLiteDatabase _db) {
   _db.execSQL(DATABASE_CREATE);
 }

 // 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.
```

Skeleton for DAO and SQLiteOpenHelper 8

http://kurser.iha.dk/eit/jem1/Basic/Android/Professional%20Android%202%20Code%20Listings/Chapter_7.txt

```
 @Override
 public void onUpgrade(SQLiteDatabase _db, int _oldVersion, int _newVersion)
{
   // Log the version upgrade.
   Log.w("TaskDBAdapter", "Upgrading from version " +
                          _oldVersion + " to " +
                          _newVersion + ", which will destroy all old
data");

   // Upgrade the existing database to conform to the new version. Multiple
   // previous versions can be handled by comparing _oldVersion and
_newVersion
   // values.

   // The simplest case is to drop the old table and create a new one.
   _db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);
   // Create a new one.
   onCreate(_db);
 }
}
}
```

mydatabase.query(...) 1

parameters for a SELECT statement

public Cursor query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)

table	The table name to compile the query against.
columns	A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
selection	A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.
selectionArgs	You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.
groupBy	A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.
having	A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.
orderBy	How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.

mydatabase.query(...) 2

parameters for a SELECT statement

Cursor android.database.sqlite.SQLiteDatabase.query(boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

distinct	true if you want each row to be unique, false otherwise.
table	The table name to compile the query against.
columns	A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
selection	A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.
selectionArgs	You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.
groupBy	A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.
having	A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.
orderBy	How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.
limit	Limits the number of rows returned by the query, formatted as LIMIT clause. Passing null denotes no LIMIT clause

Use query()

```
// Return all rows for columns one and three, no duplicates
String[] result_columns = new String[] {KEY_ID, KEY_COL1,
KEY_COL3};

Cursor allRows = db.query(true, DATABASE_TABLE, result_columns,
                                    null, null, null, null, null,
null);

// Return all columns for rows where column 3 equals a set value
// and the rows are ordered by column 5.
String where = KEY_COL3 + "=" + requiredValue;
String order = KEY_COL5;
Cursor myResult = db.query(DATABASE_TABLE, null, where,
                                     null, null, null, order);
```

Use query() and Cursor

```
int GOLD_HOARDED_COLUMN = 2;

Cursor myGold = db.query("GoldHoards", null, null, null, null, null,
null);

float totalHoard = 0f;

// Make sure there is at least one row.
if (myGold.moveToFirst()) {
  // Iterate over each cursor.
  do {
    float hoard = myGold.getFloat(GOLD_HOARDED_COLUMN);
    totalHoard += hoard;
  } while(myGold.moveToNext());
}

Float averageHoard = totalHoard / myGold.getCount();
```

Links

- JDBC in Android:
  - http://code.google.com/p/sqldroid/
- Working with SQLite databases
  - http://sqliteman.com/
- Using existing db files:
  - http://www.reigndesign.com/blog/using-your-own-sqlite-database-in-android-applications/

Lesson 8 Exercise 2

- Create an application to add customers and show customer data: name and address
- Implement insert, delete(id) and find(id)
- Extend application to keep track of orders from customers
- Make a table for orders and link orders to customers. An order can only have ONE customer, but a customers can have more orders
- What happens when you use TEXT type data in a INTEGER FIELD ?

http://www.vogella.com/articles/AndroidSQLite/article.html . A tutorial