ITMD 455/555 *Intelligent Device Applications* Lab 6

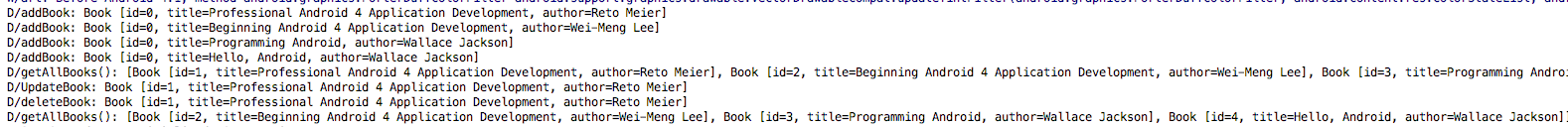
#### SQLite Books Database- 50 points

**Introduction**. This lab will have you run through stand operations of creating, reading, updating, and deleting from a database. This concept is known as CRUD which stands for create, read, update and delete.

**Objective:** For this lab you will create a local database called BookDB, with table name Books, which will

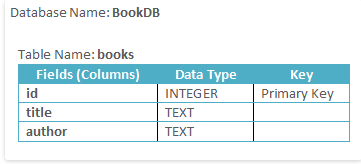
use various methods to interact with the database to thus read, update and delete from the database, etc.

All activities on the dbase will be logged and thus viewed in LogCat. A sample output shows as follows:



Study carefully the code that will follow and the Log file to observe your actions using the basic functions of a database. Run your program with differing approaches and see the reactions in **LogCat** over and over again, that’s the way to learn the “back-end” or database logic that’s produced. Add any relevant comments as well when your adding any new code into this lab app. Future lecture/labs will continue with dbase logic as that is a major chunk in any data persistent based app. Your final project no doubt will include the need for databases, **so again learn things well**! Test/run, test/run, test/run! Look carefully over all the methods related to CRUD functions especially, your “helper” class files, etc., as they especially add in important functionality to your app!

For this lab we will begin creating a database table programmatically with three columns looking like this schema:



**STEP 1 Creating a New Android Project**

Create a new project called BookReviews.

Create your first blank activity and call it MainActivity. Use the default layout name.

**STEP 2 Create a class in your package called Book.**

This class merely will provide the necessary “getters/setters” from a derived book object you will

eventually create. The class will be a springboard to help hold the table data, thus you will see

the relevant data members created for *interacting* with the three columns of the table and

various getter/setter methods to *interact* with the Book class that allows for setting data and getting data to and from the class.

Enter code in as follows:

**public** **class** Book {

**private** **int** id;

**private** String title;

**private** String author;

**public** Book(){}

**public** Book(String title, String author) {

**super**();

**this**.title = title;

**this**.author = author;

}

//getters & setters

**public** **int** getId() { **return** id; }

**public** **void** setId(**int** id) { **this**.id = id; }

**public** String getTitle() { **return** title; }

**public** **void** setTitle(String title) { **this**.title = title; }

**public** String getAuthor() { **return** author; }

**public** **void** setAuthor(String author) { **this**.author = author;}

@Override

**public** String toString() {

**return** "Book [id=" + id + ", title=" + title + ", author=" + author

+ "]";

}

}

**STEP 3 Create a new class called SqlHelper that *extends* the functionality of**

**SQLiteOpenHelper (**a helper class to manage a database creation and version management**).**

The recommended method in creating a new SQLite database is to create a subclass of **SQLiteOpenHelper** and override the **onCreate()** method, in which you can execute a SQLite command to create tables in the database.

The functionality of the class is thus:

Create a new class **SqlHelper** that extends **SQLiteOpenHelper**.

Have the **SqlHelper constructor** definitely call the **super** class constructor.

Override **onCreate()** method to create the table(s).

Override **onUpgrade()** to drop old tables (as an option) and create new ones and/or do

whatever actions needed.

Let’s begin by coding the class and we’ll go over the relevant coding syntax and logic thereafter. Include the following code into your helper class:

Here is the starter code:

**public** **class** SqlHelper **extends** SQLiteOpenHelper {

// Database Version

**private** **static** **final** **int** *DATABASE\_VERSION* = 1;

// Database Name

**private** **static** **final** String *DATABASE\_NAME* = "BookDB";

// Books table name

**private** **static** **final** String *TABLE\_BOOKS* = "books";

// Books Table Columns names

**private** **static** **final** String *KEY\_ID* = "id";

**private** **static** **final** String *KEY\_TITLE* = "title";

**private** **static** **final** String *KEY\_AUTHOR* = "author";

**public** SqlHelper(Context context) {

**super**(context, *DATABASE\_NAME*, **null**, *DATABASE\_VERSION*);

}

@Override

**public** **void** onCreate(SQLiteDatabase db) {

// SQL statement to create book table

String CREATE\_BOOK\_TABLE = "CREATE TABLE books ( " +

"id INTEGER PRIMARY KEY AUTOINCREMENT, " +

"title TEXT, "+

"author TEXT )";

// create books table

db.execSQL(CREATE\_BOOK\_TABLE);

}

@Override

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

// Drop older books table if existed

db.execSQL("DROP TABLE IF EXISTS books");

// create fresh books table

**this**.onCreate(db);

}

/\*CRUD operations (create "add", read "get", update, delete) \*/

**public** **void** addBook(Book book){

Log.*d*("addBook", book.toString());

// 1. get reference to writable DB

SQLiteDatabase db = **this**.getWritableDatabase();

// 2. create ContentValues to add key "column"/value

ContentValues values = **new** ContentValues();

values.put(*KEY\_TITLE*, book.getTitle()); // get title

values.put(*KEY\_AUTHOR*, book.getAuthor()); // get author

// 3. insert

db.insert(*TABLE\_BOOKS*, // table

**null**, //nullColumnHack

values); // key/value -> keys = column names/values

// 4. Close dbase

db.close();

}

// Get All Books

**public** List<Book> getAllBooks() {

List<Book> books = **new** LinkedList<Book>();

// 1. build the query

String query = "SELECT \* FROM " + *TABLE\_BOOKS*;

// 2. get reference to writable DB

SQLiteDatabase db = **this**.getWritableDatabase();

Cursor cursor = db.rawQuery(query, **null**);

// 3. go over each row, build book and add it to list

Book book = **null**;

**if** (cursor.moveToFirst()) {

**do** {

book = **new** Book();

book.setId(Integer.*parseInt*(cursor.getString(0)));

book.setTitle(cursor.getString(1));

book.setAuthor(cursor.getString(2));

// Add book to books

books.add(book);

} **while** (cursor.moveToNext());

}

Log.*d*("getAllBooks()", books.toString());

**return** books; // return books

}

// Updating single book

**public** **int** updateBook(Book book) {

// 1. get reference to writable DB

SQLiteDatabase db = **this**.getWritableDatabase();

// 2. create ContentValues to add key "column"/value

ContentValues values = **new** ContentValues();

values.put("title", book.getTitle()); // get title

values.put("author", book.getAuthor()); // get author

// 3. updating row

**int** i = db.update(*TABLE\_BOOKS*, //table

values, // column/value

*KEY\_ID*+" = ?", // selections

**new** String[] { String.*valueOf*(book.getId()) }); //selection args

// 4. close dbase

db.close();

Log.*d*("UpdateBook", book.toString());

**return** i;

}

// Deleting single book

**public** **void** deleteBook(Book book) {

// 1. get reference to writable DB

SQLiteDatabase db = **this**.getWritableDatabase();

// 2. delete

db.delete(*TABLE\_BOOKS*,

*KEY\_ID*+" = ?",

**new** String[] { String.*valueOf*(book.getId()) });

// 3. close

db.close();

Log.*d*("deleteBook", book.toString());

}

}

Here are your needed import statements:

**import** java.util.LinkedList;

**import** java.util.List;

**import** android.content.ContentValues;

**import** android.content.Context;

**import** android.database.Cursor;

**import** android.database.sqlite.SQLiteDatabase;

**import** android.database.sqlite.SQLiteOpenHelper;

**import** android.util.Log;

Notice that you have been provided with many functions to perform your CRUD operations. Functions are the way to go to assist interaction with your MainActivity which can merely call these functions to add, delete, view or update the database via function calls. Data parameters can be passed in as well to allow for either the handling of information into the database (ex. manually insert records as this lab demonstrates) or eventually creating the UI to allow for a user to insert data into the database for example, which again can be passed into the appropriate function in your helper class, namely the addBook method in this case. Notice also that *many* of your functions take in a **book** object that is passed into the parameter list. The object itself can then be utilized in the implementation of any your CRUD methods. For example the addBook, deleteBook and updateBook methods, all take in a Book object, which serves to *hold* the data from the database source!

Note a **VERY** important line at the very top of your member variable declaration area namely this assignment statement:

// Database Version

**private** **static** **final** **int** *DATABASE\_VERSION* = 1;

This is an interesting declaration and it does a few very important things. For one, it AUTOMATICALLY calls the **onUpgrade** method that is needed in the class which drops a table if it detects a new version or implementation of the database that is the current version. Actually if you check the parameter list of that function you’ll notice that it tracks or detects a change in version (old version vs. new version) and thus if you CHANGE the version number in your declaration (shown above) and run your app, a new version kicks in. That’s okay if you do this and you may have too if things fail on you (SQLException errors, etc.) or you just want to create a new INSTANCE of the dbase. You can toggle between versions if you like provided the prior version ran smoothly otherwise you may be dinged with an error! Further note, as you may run your app many times successfully, you will be adding new records each time (4 records at a time really per the code you’ll soon see added in via your MainActivity file) unless you comment out that code of course. That’s okay too, but note if your Log shows many records from many runs, your records will exemplify a differing **id** (i.e., incremented with a different **id** as your create statement set’s the **id** column to autoincrement) which is perfectly fine. Note further though that you want to have exact **id**’s to update or delete from later so beware of this.

Study over the CRUD functions and the resident query logic implemented for each CRUD function big time as that language is the de facto standard for executing SQLite queries. Use it and manipulate it as well as deemed worthy.

**STEP 4 Open up your MainActivity.java file and add in the following:**

**First add the following code into your Activity:**

**public** **class** MainActivity **extends** AppCompatActivity {

@Override

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

SqlHelper db = **new** SqlHelper(**this**);

/\*\* CRUD Operations \*\*/

// add Books

db.addBook(**new** Book("Android Studio Development Essentials", "Neil Smyth"));

db.addBook(**new** Book("Beginning Android Application Development", "Wei-Meng Lee"));

db.addBook(**new** Book("Programming Android", "Wallace Jackson"));

db.addBook(**new** Book("Hello, Android", "Wallace Jackson"));

// get all books

List<Book> list = db.getAllBooks();

// update one book

**int** j = db.updateBook(list.get(0));

// delete one book

db.deleteBook(list.get(0));

// get all books

db.getAllBooks();

}

}

**Next add the following import in:**

**import** java.util.List;

Notice how your MainActivity is really the driver to trigger your CRUD methods. A **List** is first created to gather the table data into the list itself via the **SqlHelper db** object which is instantiated to work the CRUD operations off the database.

As the List now holds the database data it becomes a display via the calling of the **getAllBooks()**method.

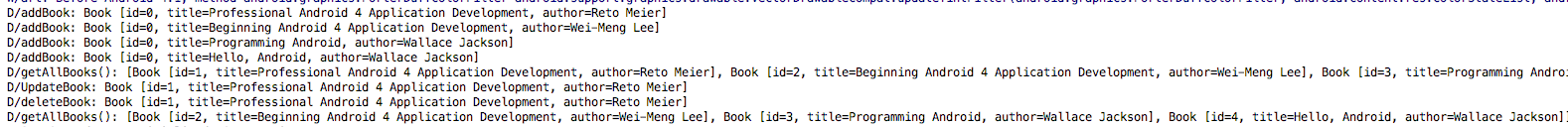
Further note the calling to update a book from the database as well as the deleting of a book from the database. We will discuss that logic in the next step.

**STEP 5 Run your code.**

Run your code and check your logcat output to see if your app has added records to the

database as well as deleted a record and an updated record from the dbase as well.

Your LogCat should display something similar to the following snapshot. (Note the clean run, in other words assuming the table data fresh with id’s 1-4.)



**Notice carefully what is going on with the display:**

1. **Four times addBook was called from the MainActivity to render 4 records into the books database.**
2. **getAllBooks was called to display the 4 records that were entered in.**
3. **UpdateBook was called to update something...? Well if you notice from the logcat display, it really didn’t update technically anything that didn’t already exist. NOTICE the id number associated with the update query (id=1). Why do you think this is so? Trace your code and you will hopefully see why!!!**
4. **deletebook was called to delete id=1 from the database which you will see from the final call in your MainActivity was successful!**

**MODIFY YOUR CODE IN THE APPROPRIATE AREA TO INCLUDE YOUR NAME IN THE LOGCAT FILE TO APPEAR BEFORE THE INFORMATION OF THE DATABASE TRANSACTION LISTED ABOVE APPEARS.**

**RERUN YOUR APP AND SNAPSHOT YOUR LOGCAT FILE SHOWING YOUR NAME AND THE RELEVANT TRANSACTIONAL INFORMATION THAT IS SHOWN EXACTLY ABOVE.**

**STEP 6 To finish, modify your code as follows:**

For this last step you will modify your app in a twofold manner. For one you will update truly a particular record and two, you will show a count of records as well.

1. Update your 4th record, namely the author’s name, to Ben Jackson. You see there was a mistake in the code purposely when record 4 was added in so you can learn to update a part of a record! ☺ To update your record adjust your coding in two areas:

In your **SqlHelper.java** file adjust your **updateBook** method to take in the following parameters:

**public** **int** updateBook(Book book, String newTitle, String newAuthor)

Notice that the function now takes in a **newTitle** and **newAuthor** as additional parameter values. This will serve as data that can be used to truly *update* a particular record when the function is called from your **MainActivity** file.

Of course your method needs to do something with the data passed in, so adjust some of the implementation, namely your **Content Values** to reflect the variables now passed in.

Note that part of your implementation already gets/sets the book id being passed via the book object passed which serves as part of your selection args provided. The id # serves as a filter to point to a particular record that reflects a given text title you want to update for our example when the app is run. You will see next that you will need to pass in the appropriate id # to correctly update any given record’s value(s).

Next in your MainActivity.java file adjust your call to the **updateBook** to correctly call the function with the new parameter values to be passed**.** Pass the appropriate values of **Hello, Android** and **Ben Jackson** to correctly update the author’s name for the **4th**

**record**, whose textbook is appropriately called **Hello, Android.**

No need to adjust any logcat information here for display purposes as the information will still trigger fine showing the new results.

RERUN YOUR APPLICATION AND **SNAPSHOT** YOUR **LOGCAT** FILE SHOWING THE UPDATED INFORMATION.

2. Show a count of the database records at the end of your LogCat file. Include the following method stub into an appropriate area in your app for the retrieval of the record count:

**public** **int** getIds()

{

String selectQuery = "SELECT id FROM books";

SQLiteDatabase database = **this**.getReadableDatabase();

Cursor c = database.rawQuery(selectQuery, **null**);

c.moveToFirst();

**int** total = c.getCount();

**return** total;

}

Include the call to getIds this as your last method call in your MainActivity along with a Log.d issuance to display the total record count along with appropriate labeling. ☺

RERUN YOUR APPLICATION AND **SNAPSHOT** YOUR **LOGCAT** FILE SHOWING *ALL* YOUR TRANSACTIONAL INFORMATION INCLUDING YOUR TOTAL RECORD COUNT.

Include in Word, all your snapshots. Also include your source files and xml files, etc. zipped up to BB for credit.