# SQLite and Android

There are many ways to handle data on mobile devices. Unstructured data such as a jpeg file or html page is best stored in files. Structured or relational data such as your list of contacts is best stored and managed using relational databases. Android supports the very simple and very cool SQLite.

SQLite is not directly comparable to client/server SQL database engines such as MySQL, Oracle, PostgreSQL, or SQL Server since SQLite is trying to solve a different problem.

Client/server SQL database engines strive to implement a shared repository of enterprise data. They emphasize scalability, concurrency, centralization, and control. SQLite strives to provide local data storage for individual applications and devices. SQLite emphasizes economy, efficiency, reliability, independence, and simplicity.

SQLite does not compete with client/server databases.

Because an SQLite database requires no administration, it works well in devices that must operate without expert human support. SQLite is a good fit for use in cellphones, set-top boxes, televisions, game consoles, cameras, watches, kitchen appliances, thermostats, automobiles, machine tools, airplanes, remote sensors, drones, medical devices, and robots: the "internet of things".

*SQLite* is an Open Source database. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. The database requires limited memory at runtime (approx. 250 KByte).

The main package is android.database.sqlite that contains the classes to manage your own databases

SQLite supports the following data types:

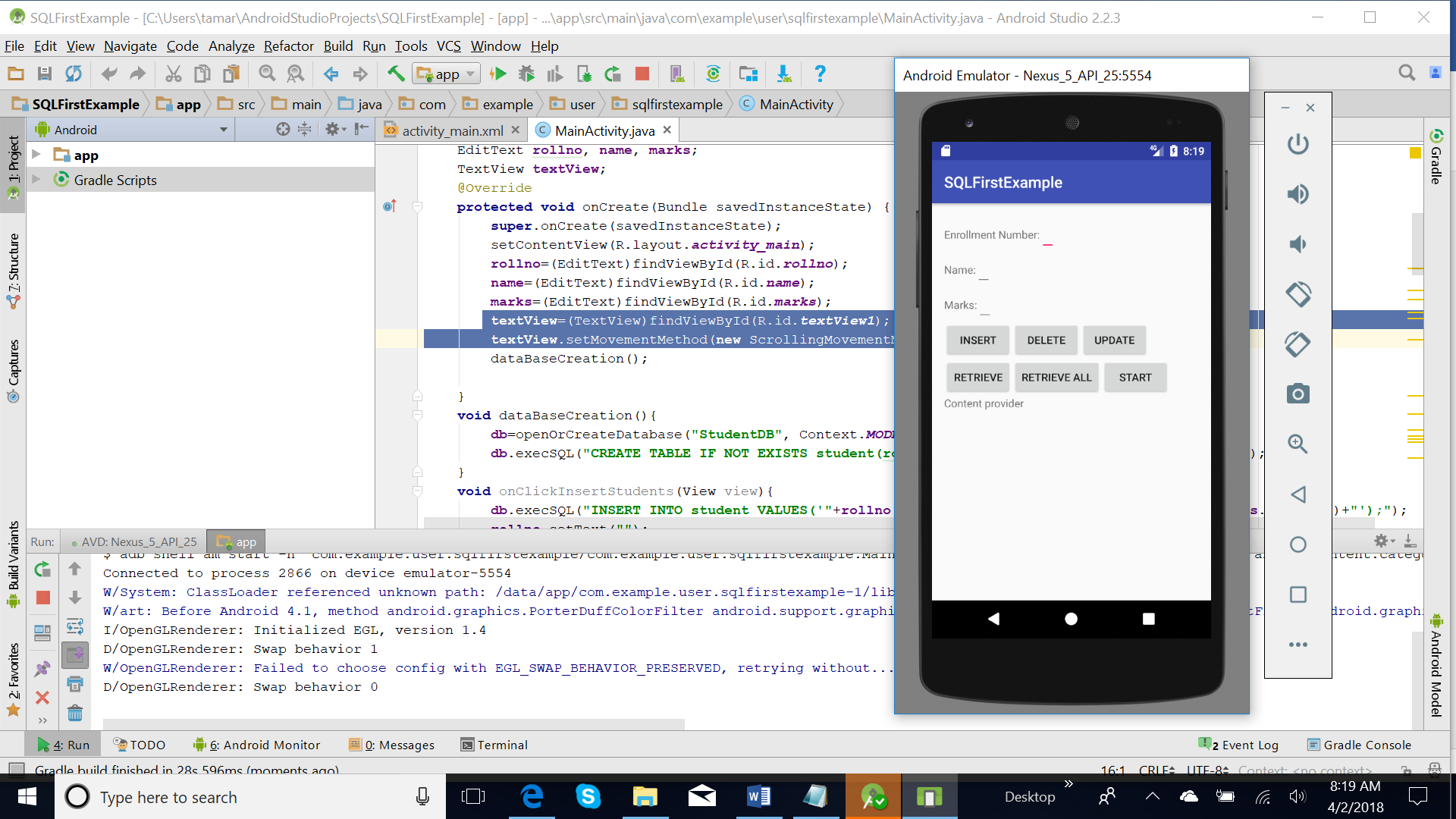
* TEXT(similar to String in Java)
* INTEGER(similar to long in Java)
* REAL (similar to double in Java). All other types must be converted into one of these fields before getting saved in the database. SQLite itself does not validate if the types written to the columns are actually of the defined type. This means you can write an integer into a string column and vice versa.[[1]](#footnote-1)

Access to an SQLite database involves accessing the file system. This can be slow. Therefore, it is recommended to perform database operations asynchronously.

Example: Let see through an example all the steps that we need te create a simple database in SQLite and the basic operations like insert, update delete, and retrieve related to a table.

This example application (it is taken from Internet partially) creates a simple Student Management System, which allows a user to add, delete, modify and view student details. The application accepts a student's roll number, name and marks and adds these details to a student table. For simplicity, all the fields are of VARCHAR data type, which represents a string type.

We need to provide the user interface



For this interface we need the .xml file

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:id="@+id/activity\_main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:paddingBottom="@dimen/activity\_vertical\_margin"  
 android:paddingLeft="@dimen/activity\_horizontal\_margin"  
 android:paddingRight="@dimen/activity\_horizontal\_margin"  
 android:paddingTop="@dimen/activity\_vertical\_margin"  
 tools:context="com.example.user.sqlfirstexample.MainActivity"**>  
 <**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:orientation="vertical"** >  
 <**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/edit1Layout"  
 android:orientation="horizontal"  
 android:weightSum="2"**>  
 <**TextView  
 android:id="@+id/rollnoLabel"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Enrollment Number:"** />  
  
 <**EditText  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/rollno"  
 android:layout\_alignRight="@+id/rollnoLabel"  
 android:layout\_alignEnd="@+id/rollnoLabel"** />  
 </**LinearLayout**>  
 <**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/edit2Layout"  
 android:orientation="horizontal"  
 android:weightSum="2"**>  
 <**TextView  
 android:id="@+id/nameLabel"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Name:"** />  
 <**EditText  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/name"  
 android:layout\_alignRight="@+id/nameLabel"  
 android:layout\_alignEnd="@+id/nameLabel"** />  
 </**LinearLayout**>  
 <**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/edit3Layout"  
 android:orientation="horizontal"  
 android:weightSum="2"**>  
 <**TextView  
 android:id="@+id/marksLabel"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Marks:"** />  
 <**EditText  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/marks"  
 android:layout\_alignRight="@+id/marksLabel"  
 android:layout\_alignEnd="@+id/marksLabel"** />  
 </**LinearLayout**>  
 <**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/firstLayout"  
 android:orientation="horizontal"  
 android:weightSum="2"**>  
 <**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Insert"  
 android:id="@+id/button1"  
 android:layout\_below="@id/marks"  
 android:onClick="onClickInsertStudents"**/>  
 <**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Delete"  
 android:id="@+id/button2"  
 android:onClick="onClickDeleteStudents"**/>  
 <**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Update"  
 android:id="@+id/button3"  
 android:onClick="onClickUpdateStudents"**/>  
  
 </**LinearLayout**>  
 <**LinearLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/secondLayout"  
 android:orientation="horizontal"  
 android:weightSum="2"**>  
 <**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Retrieve"  
 android:id="@+id/button4"  
 android:onClick="onClickRetrieveStudent"**/>  
 <**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Retrieve All"  
 android:id="@+id/button5"  
 android:onClick="onClickRetrieveStudents"**/>  
 <**Button  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Start"  
 android:id="@+id/button6"  
 android:onClick="onClickDeleteDatabase"**/>  
  
 </**LinearLayout**>  
 <**TextView  
 android:id="@+id/textView1"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:text="Content provider"  
 android:layout\_alignParentBottom="true"  
 android:layout\_centerHorizontal="true"  
 android:lines="10"  
 android:scrollbars="vertical"  
 android:scrollbarStyle="insideOverlay"  
 android:fadeScrollbars="true"  
 android:fadingEdge="vertical"** />  
 </**LinearLayout**>  
</**RelativeLayout**>

The steps we need to follow in MainActivity are

1. Link with user interface controls

**rollno**=(EditText)findViewById(R.id.***rollno***);  
 **name**=(EditText)findViewById(R.id.***name***);  
 **marks**=(EditText)findViewById(R.id.***marks***);  
 **textView**=(TextView)findViewById(R.id.***textView1***);

1. Creation of a database and a table that has three VARCHAR columns

**db**=openOrCreateDatabase(**"StudentDB"**, Context.***MODE\_PRIVATE***, **null**);  
**db**.execSQL(**"CREATE TABLE IF NOT EXISTS student**

**(rollno VARCHAR,name VARCHAR,marks VARCHAR);"**);

DataBase name is StudentDB and MODE\_PRIVATE means that the created file can only be accessed by the calling application (or all applications sharing the same user ID). The table student is created too.

1. The method onClickInsertStudents, that is invoked when the button “Insert” is clicked, inserts data into table

**void** onClickInsertStudents(View view){  
 **db**.execSQL(**"INSERT INTO student VALUES**

**('"**+**rollno**.getText()+**"',**

**'"**+**name**.getText()+**"',**

**'"**+**marks**.getText()+**"');"**);  
 **rollno**.setText(**""**);  
 **name**.setText(**""**);  
 **marks**.setText(**""**);  
 }

After insertion we clean the edit fields for the next entry.

1. Delete a record of the table for a value of the student enrollment is another basic operation for the database.

**void** onClickDeleteStudents(View view){  
 **db**.execSQL(**"DELETE FROM student WHERE**

**rollno='"**+**rollno**.getText()+**"'"**);  
 }

This method is invoked when we click the button “Delete”;

1. A table record can be retrieved based on a query using the student enrollment:

**void** onClickRetrieveStudent(View view) {  
 String query=

**"SELECT \* FROM student WHERE rollno='"**+**rollno**.getText()+**"'"**;  
 Cursor c=**db**.rawQuery(query, **null**);  
 **if**(c.moveToFirst())  
 {  
 **name**.setText(c.getString(1));  
 **marks**.setText(c.getString(2));  
 }  
 }

This method is invoked when we click the button “Retrieve”;

The query on the database returns a cursor, object of the class Cursor which includes the data asked by the query. As you see from the snippet there is control over navigation of the cursor -- see *cursor.moveToFirst()* in the code. This method provides a managed way to move to the first element in the cursor -- recall there could be many rows returned and cursor.moveToFirst() moves the position of the cursor to the first row. There are a number of navigation methods but here are some commonly used ones:

* move(int offset) - move the cursor by a relative amount, forward or backward, from the current position.
* moveToFirst() - move the cursor to the first row
* moveToLast() - move the cursor to the last row.
* moveToNext() - move the cursor to the next row.
* moveToPosition(int position) - move the cursor to an absolute position.
* moveToPrevious() - move the cursor to the previous row.

Other important methods very useful for interacting with cursor objects include:

* getPosition() - returns the current position of the cursor in the row set.
* getCount() - returns the numbers of rows in the cursor.
* getColumnNames() - returns a string array holding the names of all of the columns in the result set in the order in which they were listed in the result.
* getColumnIndexOrThrow(String columnName) - returns the zero-based index for the given column name, or throws IllegalArgumentException if the column doesn't exist.
* getColumnCount() - return total number of columns

And finally important control methods on the object:

* close() - closes the Cursor, releasing all of its resources and making it completely invalid.

1. After we retrieve a record based on enrollment number we can modify it.

**void** onClickUpdateStudents(View view){  
 **db**.execSQL(**"UPDATE student SET**

**name='"**+**name**.getText()+**"',**

**marks='"**+**marks**.getText()+**"' WHERE**

**rollno='"**+**rollno**.getText()+**"'"**);  
 }

1. If we need to display all the records we use the method onClickRetrieveStudents. This method is invoked when we click the button “Retrieve All”;

**void** onClickRetrieveStudents(View view) {  
 Cursor c = db.rawQuery("SELECT \* FROM student", null);  
 if (c.getCount() == 0) {  
 textView.setText("Error, No records found");  
 return;  
 }  
 StringBuffer buffer = new StringBuffer();  
 while (c.moveToNext()) {  
 buffer.append("Rollno: " + c.getString(0) + "\n");  
 buffer.append("Name: " + c.getString(1) + "\n");  
 buffer.append("Marks: " + c.getString(2) + "\n\n");  
 }  
 textView.setText(buffer.toString());\*/

}

1. If we need to delete all the database and create a new one we use the method.

**void** onClickDeleteDatabase(View view){  
 **this**.deleteDatabase(**"StudentDB"**);  
 }

This method is invoked when we click the button “Start”;

If one of operations takes time we can use a AsyncTask class, for example in this example we can put the retrieve of all the records in doInBackground method.

**class** DatabaseAsync **extends** AsyncTask<String, Void, String> {  
 String **result**=**""**;  
 @Override  
 **protected** String doInBackground(String... param) {  
 **try**{  
 Cursor c = **db**.rawQuery(**param[0]**, **null**);  
 **if** (c.getCount() == 0) {  
 **result**=**"Error, No records found"**;  
 **return result**;  
 }  
 StringBuffer buffer = **new** StringBuffer();  
 **while** (c.moveToNext()) {  
 buffer.append(**"Rollno: "** + c.getString(0) + **"\n"**);  
 buffer.append(**"Name: "** + c.getString(1) + **"\n"**);  
 buffer.append(**"Marks: "** + c.getString(2) + **"\n\n"**);  
 }  
 **return**(buffer.toString());  
  
 } **catch** (Exception e) {  
 Log.*d*(**"error "**, e.getMessage().toString());  
 }  
 Log.*d*(**"query3 "**,**result**);  
 **return result**;  
  
 }  
  
 @Override  
 **protected void** onPostExecute(String result) {  
 Log.*d*(**"query4 "**,result);  
 **textView**.setText(result);  
  
 }  
  
  
 }  
}

Let organize our program differently creating some layers.

* **app layer** -- just deals with the UI
* **data storage layer** -- application specific and deals with creating the database and operations on it but these are still database independent.
* **SQLIte layer** -- this deals with all the plumbing and database specific operations through a **db helper**

1. First organize our data in a class: MyData. We are adding an additional field here that will correspond to a Table key in database.

**public class** MyData {  
 **private long id**;  
 **private** String **rollno**;  
 **private** String **name**;  
 **private** String **marks**;  
  
 **public long** getId() {  
 **return id**;  
 }  
  
 **public void** setId(**long** id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getRollno() {  
 **return rollno**;  
 }  
  
 **public void** setRollno(String rollno) {  
 **this**.**rollno** =rollno;  
 }  
  
 **public** String getName() {  
 **return name**;  
 }  
  
 **public void** setName(String name) {  
 **this**.**name** =name;  
 }  
 **public** String getMarks() {  
 **return marks**;  
 }  
  
 **public void** setMarks(String marks) {  
 **this**.**marks** = marks;  
 }  
  
}

1. We can use a helper class SQLiteDatabase ,to manage database creation and version management.

You create a subclass implementing onCreate(SQLiteDatabase), onUpgrade(SQLiteDatabase, int, int) and optionally onOpen(SQLiteDatabase), and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state.

So, the syntaxes of the creation of a database, or table, and the syntaxes of the database operations are simplified using methods of SQLiteDatabase class.

Therefore the first thing that we do is creation of an extension of the class SQLiteDatabase.

**import** android.content.Context;  
**import** android.database.sqlite.SQLiteDatabase;  
**import** android.database.sqlite.SQLiteOpenHelper;  
**import** android.util.Log;  
  
**public class** MySQLiteHelper **extends** SQLiteOpenHelper {  
  
 **public static final** String ***TABLE\_DATA*** = **"Student"**;  
 **public static final** String ***COLUMN\_ID*** = **"\_id"**;  
 **public static final** String ***COLUMN\_DATA1*** = **"rollno"**;  
 **public static final** String ***COLUMN\_DATA2*** = **"name"**;  
 **public static final** String ***COLUMN\_DATA3*** = **"marks"**;  
  
 **private static final** String ***DATABASE\_NAME*** = **"StudentDB"**;  
 **private static final int *DATABASE\_VERSION*** = 1;  
  
 *// Database creation sql statement* **private static final** String ***DATABASE\_CREATE*** = **"create table "** + ***TABLE\_DATA*** + **"( "** + ***COLUMN\_ID*** + **" integer primary key autoincrement, "**

+ ***COLUMN\_DATA1***+ **" varchar,"**

+ ***COLUMN\_DATA2***+ **" varchar,"**

+ ***COLUMN\_DATA3***+ **" varchar);"**;  
  
 **public** MySQLiteHelper(Context context) {  
 **super**(context, ***DATABASE\_NAME***, **null**, ***DATABASE\_VERSION***);  
  
 }  
  
 **void** DeleteDatabase(Context context){  
 context.deleteDatabase(***DATABASE\_NAME***);  
 }  
  
 @Override  
 **public void** onCreate(SQLiteDatabase database) {  
  
 database.execSQL(***DATABASE\_CREATE***);  
 }  
  
 @Override  
 **public void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion)

{  
 Log.*w*(MySQLiteHelper.**class**.getName(),  
 **"Upgrading database from version "** + oldVersion + **" to "** + newVersion + **", which will destroy all old data"**);  
 db.execSQL(**"DROP TABLE IF EXISTS "** + ***TABLE\_DATA***);  
 onCreate(db);  
 }  
  
}

Where we are using constants to make the work easier to be used in different cases. This is the **SQLIte layer**

1. In a separate class, in our case MyDataSource class, we develop all the methods to operate with the database. This is the **data storage layer.**

**import** android.content.ContentValues;  
**import** android.content.Context;  
**import** android.database.Cursor;  
**import** android.database.SQLException;  
**import** android.database.sqlite.SQLiteDatabase;  
  
  
**public class** MyDataSource {  
 *// Database fields* **private** SQLiteDatabase **database**;  
 **private** MySQLiteHelper **dbHelper**;  
  
 **private** String[] **allColumns** = { MySQLiteHelper.***COLUMN\_ID***,  
 MySQLiteHelper.***COLUMN\_DATA1***, MySQLiteHelper.***COLUMN\_DATA2***,MySQLiteHelper.***COLUMN\_DATA3***};  
  
 **public** MyDataSource(Context context) {  
 **dbHelper** = **new** MySQLiteHelper(context);  
 }  
  
  
 **public void** open() **throws** SQLException {  
 **database** = **dbHelper**.getWritableDatabase();  
 }  
  
 **void** InsertStudent(String value1, String value2, String value3){  
 ContentValues values = **new** ContentValues();  
 values.put(MySQLiteHelper.***COLUMN\_DATA1***, value1);  
 values.put(MySQLiteHelper.***COLUMN\_DATA2***,value2);  
 values.put(MySQLiteHelper.***COLUMN\_DATA3***,value3);  
 **long** insertId = **database**.insert(MySQLiteHelper.***TABLE\_DATA***, **null**,values);  
 }  
 **public void** deleteStudent(**long** id1) {  
 **long** id = id1;  
 **database**.delete(MySQLiteHelper.***TABLE\_DATA***, MySQLiteHelper.***COLUMN\_ID*** + **" = "** + id, **null**);  
 }  
 **public void** updateStudent(**long** id1,

String value1,String value2, String value3) {  
 **long** id = id1;  
 ContentValues values = **new** ContentValues();  
 values.put(MySQLiteHelper.***COLUMN\_DATA1***, value1);  
 values.put(MySQLiteHelper.***COLUMN\_DATA2***,value2);  
 values.put(MySQLiteHelper.***COLUMN\_DATA3***,value3);  
  
 **database**.update(MySQLiteHelper.***TABLE\_DATA***,

values,MySQLiteHelper.***COLUMN\_ID***+**"="**+id1, **null**);  
 }  
 **public** MyData retrieveStudent(**long** id1) {  
 **long** id = id1;  
 Cursor cursor = **database**.query(MySQLiteHelper.***TABLE\_DATA***,  
 **allColumns**, **null**, **null**,  
 **null**, **null**, **null**);  
 MyData newData=**null**;  
 cursor.moveToFirst();  
 **while** (!cursor.isAfterLast()) {  
 newData = cursorToData(cursor);  
 cursor.moveToNext();  
 }  
 *// make sure to close the cursor* cursor.close();  
 **return** newData;  
 }  
 **public** String retrieveStudents() {  
 Cursor cursor = **database**.query(MySQLiteHelper.***TABLE\_DATA***,  
 **allColumns**, **null**, **null**,  
 **null**, **null**, **null**);  
 MyData newData=**null**;  
 String result=**""**;  
 cursor.moveToFirst();  
 **while** (!cursor.isAfterLast()) {  
 newData = cursorToData(cursor);  
 result+=newData.getId()+**" "**+newData.getRollno()+**"**

**"**+newData.getName()+**" "**+newData.getMarks()+**"\n"**;  
 cursor.moveToNext();  
 }  
 *// make sure to close the cursor* cursor.close();  
 **return** result;  
 }  
  
 **private** MyData cursorToData(Cursor cursor) {  
 MyData mydata = **new** MyData();  
 mydata.setId(cursor.getLong(0));  
 mydata.setRollno(cursor.getString(1));  
 mydata.setName(cursor.getString(2));  
 mydata.setMarks(cursor.getString(3));  
 **return** mydata;  
 }  
  
 **public void** close() {  
 **dbHelper**.close();  
 }  
  
 **public void** deleteDataBase(Context context) {  
 **dbHelper**.DeleteDatabase(context);  
 }  
}

1. We complete the MainActivity, to link it with UI and the methods that are invoked when we click the buttons.

**import** android.app.Activity;  
**import** android.os.Bundle;  
**import** android.text.method.ScrollingMovementMethod;  
**import** android.view.View;  
**import** android.widget.EditText;  
**import** android.widget.TextView;  
  
**public class** MainActivity **extends** Activity {  
 **private** MyDataSource **datasource**;  
 EditText **rollno**, **name**, **marks**,**searchField**;  
 TextView **textView**;  
  
 @Override  
 **public void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
 **searchField**=(EditText)findViewById(R.id.***searchField***);  
 **rollno**=(EditText)findViewById(R.id.***rollno***);  
 **name**=(EditText)findViewById(R.id.***name***);  
 **marks**=(EditText)findViewById(R.id.***marks***);  
 **textView**=(TextView)findViewById(R.id.***textView1***);  
 **textView**.setMovementMethod(**new** ScrollingMovementMethod());  
  
 dataBaseCreation();  
  
 }  
  
 **void** dataBaseCreation(){  
 **datasource** = **new** MyDataSource(**this**);  
 **datasource**.open();  
 }  
  
 **void** onClickInsertStudents(View view){  
 **datasource**.InsertStudent(**rollno**.getText().toString(),

**name**.getText().toString(),**marks**.getText().toString());  
 **rollno**.setText(**""**);  
 **name**.setText(**""**);  
 **marks**.setText(**""**);  
 **rollno**.setFocusable(**true**);  
 }  
 **void** onClickDeleteStudents(View view){  
 **datasource**.deleteStudent(

Long.*parseLong*(**searchField**.getText().toString()));  
 **name**.setText(**""**);  
 **marks**.setText(**""**);  
 **rollno**.setFocusable(**true**);  
 }  
 **void** onClickUpdateStudents(View view){  
 **datasource**.updateStudent(

Long.*parseLong*(**searchField**.getText().toString()),  
 **rollno**.getText().toString(),  
 **name**.getText().toString(),  
 **marks**.getText().toString());  
 **name**.setText(**""**);  
 **marks**.setText(**""**);  
 **rollno**.setFocusable(**true**);  
 }  
 **void** onClickRetrieveStudent(View view) {  
 **if**( **searchField**.getText().toString().length()>0) {  
 MyData mydata = **datasource**.retrieveStudent(

Long.*parseLong*(**searchField**.getText().toString()));  
 **rollno**.setText(mydata.getRollno());  
 **name**.setText(mydata.getName());  
 **marks**.setText(mydata.getMarks());  
 }  
 }  
 **void** onClickRetrieveStudents(View view) {  
 String result=**datasource**.retrieveStudents();  
 **textView**.setText(result);  
 }  
  
  
 **void** onClickDeleteDatabase(View view){  
 **datasource**.deleteDataBase(**this**);  
 dataBaseCreation();  
 }  
 **void** onClickQuit(View view){  
 **datasource**.close();  
  
  
 }  
 @Override  
 **protected void** onResume() {  
 **datasource**.open();  
 **super**.onResume();  
 }  
  
 @Override  
 **protected void** onPause() {  
 **datasource**.close();  
 **super**.onPause();  
 }  
  
}

## Assignment

The goal of the assignment is to store the data the come from accelerometer in a Database.

1. You need to create a table using the query:

**CREATE TABLE IF NOT EXISTS sensordata"** +  
 **"(ID integer primary key autoincrement, value1 VARCHAR, value2 VARCHAR, value3 VARCHAR);**

to create a table for the sensor data

1. You can code just three methods (when you click a button)

**void** onClickInsert(View view){  
. . .  
  
}  
**void** onClickDelete(View view){  
 **. . .**  
}  
  
**void** onClickRetrieveAll(View view) {  
. . .

}

1. From onClickInsert method call an intent, that will bring in MainActivity the sensor data,
2. Use the AsyncTask from onActivityResult method.
3. The communication between two intents related to data, let be by strings. You can compose the data in sensor Intent in one string and decompose this string in MainActivity in some floats. For this maybe you need to use the Java syntax:

datastring.split(delimiter)where the delimiter can be “-“ or any other.

**References:**

<https://www.codeproject.com/Articles/783073/A-Simple-Android-SQLite-Example>

<http://www.vogella.com/tutorials/AndroidSQLite/article.html#sqlite-and-android>

<https://www.tutorialspoint.com/android/android_sqlite_database.htm>

1. http://www.tutorialspoint.com/sqlite/sqlite\_data\_types.htm [↑](#footnote-ref-1)