

Android Studio



West University of Timisoara, Romania
Computer Science Department
IE3, Fall 2015
Dr. Liviu Octavian Mafteiu-Scai

Working with Databases

Databases - an overview

Database *Tables* provide the basic level of data structure in a database. A database can contain multiple tables and each table is designed to hold information of a specific type.

Tables

	Table 1				Table 2		
Record 1	Field 1	Field 2	Field 3	Record 1	Field 1	Field 2	Field 3
Record 2	Field 1	Field 2	Field 3	Record 2	Field 1	Field 2	Field 3
Record 3	Field 1	Field 2	Field 3	Record 3	Field 1	Field 2	Field 3
Record 4	Field 1	Field 2	Field 3				

It is helpful to view a database table as being similar to a spreadsheet:

- a column represents a data field in the corresponding table.
 - each new record that is saved to a table is stored in a row.
- Rows are also sometimes referred to as records or entries*

Each *table* has a name that must be unique within that particular database. A table name, once assigned to a table in one database, may only be re-used within the context of a different database.

Database Schema define the characteristics of the data stored in a database table. More, are also used to define the structure of entire databases and the relationship between the various tables contained in a database.

Primary Key: a database table must contain one or more columns that can be used to identify each row in the table uniquely.

SQL ⇔ Structured Query Language

a language designed to enable the reading and writing of database data

What is SQLite ?

SQLite is an embedded, relational database management system (RDBMS).



embedded

- ⇔ *SQLite* is provided in the form of a library that is linked into applications
- ⇔ NO standalone database server
- ⇔ All database operations are handled internally using functions from SQLite library.

SQLite is written in C programming language !

Android SQLite Java Classes

SQLite is written in the C, Android apps are written in Java

SO

to bridge this “language gap”, the Android SDK includes a set of classes that provide a Java layer on top of the *SQLite* database management system.

1. Cursor class include methods:

close() – Releases all resources used by the cursor and closes it.

getCount() – Returns the number of rows contained within the result set.

moveToFirst() – Moves to the first row within the result set.

moveToLast() – Moves to the last row in the result set.

moveToNext() – Moves to the next row in the result set.

move() – Moves by a specified offset from the current position in the result set.

get<type>() – Returns the value of the specified <type> contained at the specified column index of the row at the current cursor position (variations consist of *getString()*, *getInt()*, *getShort()*, *getFloat()* and *getDouble()*).

Android SQLite Java Classes

SQLiteDatabase class methods:

insert() – Inserts a new row into a database table.

delete() – Deletes rows from a database table.

query() – Performs a specified database query and returns matching results via a Cursor object.

execSQL() – Executes a single SQL statement that does not return result data.

rawQuery() – Executes an SQL query statement and returns matching results in the form of a Cursor object.

SQLiteOpenHelper class methods:

onCreate() – Called when the database is created for the first time.

onUpgrade() – Called in the event that the application code contains a more recent database version number reference.

getWritableDatabase() – Opens or creates a database for reading and writing. Returns a reference to the database in the form of a SQLiteDatabase object.

getReadableDatabase() – Creates or opens a database for reading only. Returns a reference to the database in the form of a SQLiteDatabase object.

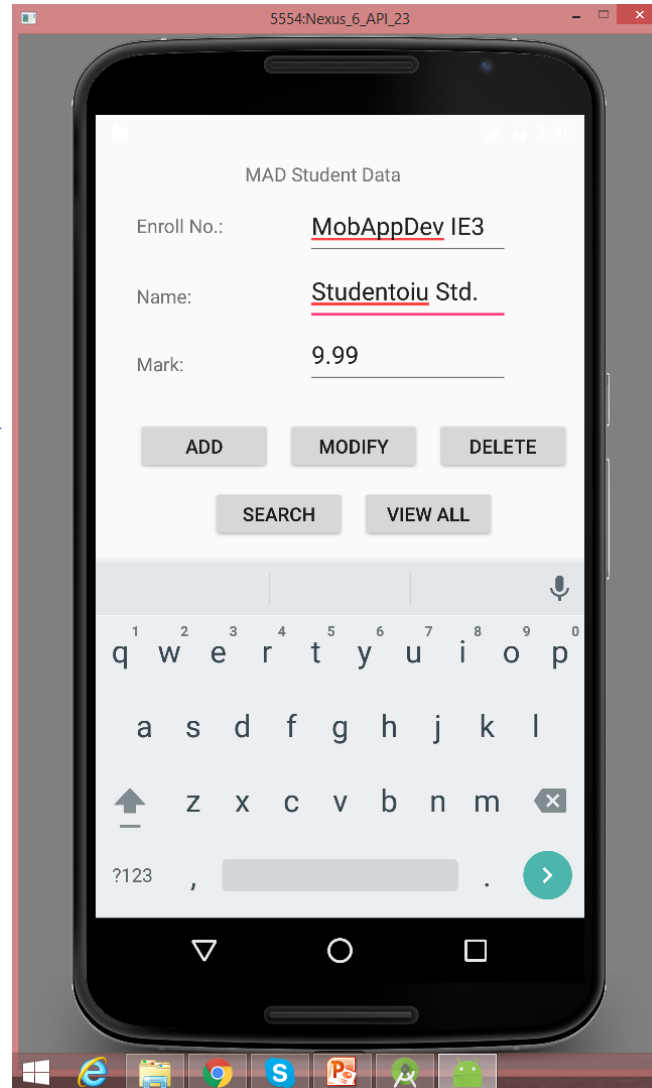
close() – Closes the database.

Goal:

An Android app for managing a sample student database (EnrollNumbe, StudentName and StudentMarks). The user's operations will be : *Add, Modify, Delete, Search* and *ViewAll* records.

The user interface layout
(based on an *AbsoluteLayout* style)

Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.
! it's a little deprecated !



content_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myLayout"
    android:stretchColumns="0"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:text="@string/title"
        android:layout_x="110dp"
        android:layout_y="10dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView android:text="@string/enroll_no"
        android:layout_x="30dp"
        android:layout_y="48dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText android:id="@+id/editEnrollno"
        android:layout_x="155dp"
        android:layout_y="33dp"
        android:layout_width="150dp"
        android:layout_height="50dp"/>
    <TextView android:text="@string/name"
        android:layout_x="30dp"
        android:layout_y="100dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <EditText android:id="@+id/editName"
        android:inputType="text"
        android:layout_x="155dp"
        android:layout_y="81dp"
        android:layout_width="150dp"
        android:layout_height="50dp"/>
    <TextView android:text="@string/marks"
        android:layout_x="30dp"
        android:layout_y="150dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

```
<EditText android:id="@+id/editMarks"
    android:inputType="numberDecimal"
    android:layout_x="155dp"
    android:layout_y="128dp"
    android:layout_width="150dp"
    android:layout_height="50dp"/>
    <Button android:id="@+id/btnAdd"
        android:text="@string/add"
        android:layout_x="30dp"
        android:layout_y="200dp"
        android:layout_width="100dp"
        android:layout_height="40dp"/>
    <Button android:id="@+id/btnDelete"
        android:text="@string/delete"
        android:layout_x="250dp"
        android:layout_y="200dp"
        android:layout_width="100dp"
        android:layout_height="40dp"/>
    <Button android:id="@+id/btnModify"
        android:text="@string/modify"
        android:layout_x="140dp"
        android:layout_y="200dp"
        android:layout_width="100dp"
        android:layout_height="40dp"/>
    <Button android:id="@+id/btnSearch"
        android:text="@string/view"
        android:layout_x="85dp"
        android:layout_y="250dp"
        android:layout_width="100dp"
        android:layout_height="40dp"/>
    <Button android:id="@+id/btnViewAll"
        android:text="@string/view_all"
        android:layout_x="195dp"
        android:layout_y="250dp"
        android:layout_width="100dp"
        android:layout_height="40dp"/>
```

```
</AbsoluteLayout>
```

strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MAD Student Database App</string>
    <string name="title">MAD Student Data</string>
    <string name="enroll_no">Enroll No.: </string>
    <string name="name">Name: </string>
    <string name="marks">Mark: </string>
    <string name="add">Add</string>
    <string name="delete">Delete</string>
    <string name="modify">Modify</string>
    <string name="view">Search</string>
    <string name="view_all">View All</string>
</resources>
```


MainActivity.java (1)

```
package com.example.mafteiu_scai.mystudents;
```

```
import android.app.Activity;
import android.app.AlertDialog.Builder;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import com.example.mafteiu_scai.mystudents.R;
```

import SQLite classes.

Called when the activity is first created.

```
public class MainActivity extends Activity implements OnClickListener {
    EditText editEnrollno, editName, editMarks;
    Button btnAdd, btnDelete, btnModify, btnSearch, btnViewAll;
    SQLiteDatabase db;
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.content_main);
    editEnrollno = (EditText) findViewById(R.id.editEnrollno);
    editName = (EditText) findViewById(R.id.editName);
    editMarks = (EditText) findViewById(R.id.editMarks);
    btnAdd = (Button) findViewById(R.id.btnAdd);
    btnDelete = (Button) findViewById(R.id.btnDelete);
    btnModify = (Button) findViewById(R.id.btnModify);
    btnSearch = (Button) findViewById(R.id.btnSearch);
    btnViewAll = (Button) findViewById(R.id.btnViewAll);
    btnAdd.setOnClickListener(this);
    btnDelete.setOnClickListener(this);
    btnModify.setOnClickListener(this);
    btnSearch.setOnClickListener(this);
    btnViewAll.setOnClickListener(this);
```

```
db = openOrCreateDatabase("MADStudentsDB", Context.MODE_PRIVATE, null);
db.execSQL("CREATE TABLE IF NOT EXISTS student(enrollno VARCHAR,name VARCHAR,marks VARCHAR);");
```

```
}
```

Enroll No.:	MobAppDev IE3
Name:	Studentoiu Std.
Mark:	9.99

ADD	MODIFY	DELETE
SEARCH		VIEW ALL

Called when when a button is pressed.

Path to database file to open and/or create

the structure of table student

MainActivity.java (2)

```
public void onClick(View view) {
```

```
    if (view == btnAdd) {  
        if (editEnrollno.getText().toString().trim().length() == 0 ||  
            editName.getText().toString().trim().length() == 0 ||  
            editMarks.getText().toString().trim().length() == 0) {  
            showMessage("Error", "Please enter all student data");  
            return;  
        }  
        db.execSQL("INSERT INTO student VALUES('" + editEnrollno.getText() + "','" + editName.getText() +  
            "','" + editMarks.getText() + "')");  
        showMessage("Success", "Record added");  
        clearText();  
    }
```

add students to the database under certain conditions

```
    if (view == btnDelete) {  
        if (editEnrollno.getText().toString().trim().length() == 0) {  
            showMessage("Error", "Please enter Student Enroll Number");  
            return;  
        }  
        Cursor c = db.rawQuery("SELECT * FROM student WHERE enrollno='" + editEnrollno.getText() + "'", null);  
        if (c.moveToFirst()) {  
            db.execSQL("DELETE FROM student WHERE enrollno='" + editEnrollno.getText() + "'");  
            showMessage("Success", "Record Deleted");  
        } else {  
            showMessage("Error", "Invalid Enroll Number");  
        }  
        clearText();  
    }
```

delete a student from table, based in enroll number

The rawQuery method has two parameters:

-String query: The select statement

-String[] selection args: The arguments if a WHERE clause is included in the select statement

MainActivity.java (3)

```

if (view == btnModify) {
    if (editEnrollNo.getText().toString().trim().length() == 0) {
        showMessage("Error", "Please enter Enroll Number");
        return;
    }
    Cursor c = db.rawQuery("SELECT * FROM student WHERE enrollNo='" + editEnrollNo.getText() + "'", null);
    if (c.moveToFirst()) {
        db.execSQL("UPDATE student SET name='" + editName.getText() + "',marks='" + editMarks.getText() +
            "' WHERE enrollNo='" + editEnrollNo.getText() + "'");
        showMessage("Success", "Student Data Modified");
    } else {
        showMessage("Error", "Invalid Enroll Number");
    }
    clearText();
}

if (view == btnSearch) {
    if (editEnrollNo.getText().toString().trim().length() == 0) {
        showMessage("Error", "Please enter Student enroll number");
        return;
    }
    Cursor c = db.rawQuery("SELECT * FROM student WHERE enrollNo='" + editEnrollNo.getText() + "'", null);
    if (c.moveToFirst()) {
        editName.setText(c.getString(1));
        editMarks.setText(c.getString(2));
    } else {
        showMessage("Error", "Invalid Enroll Number");
        clearText();
    }
}
}

```

modify students's data

Search a record (student), based on enroll number

MainActivity.java (4)

```
if (view == btnViewAll) {
    Cursor c = db.rawQuery("SELECT * FROM student", null);
    if (c.getCount() == 0) {
        showMessage("Error", "No students found in database");
        return;
    }
    StringBuffer buffer = new StringBuffer();
    while (c.moveToNext()) {
        buffer.append("Enroll Number: " + c.getString(0) + "\n");
        buffer.append("Name: " + c.getString(1) + "\n");
        buffer.append("Mark: " + c.getString(2) + "\n\n");
    }
    showMessage("MAD Student Data", buffer.toString());
}
```

← View all records (students) of database

```
}
public void showMessage(String title, String message) {
    Builder builder = new Builder(this);
    builder.setCancelable(true);
    builder.setTitle(title);
    builder.setMessage(message);
    builder.show();
}
```

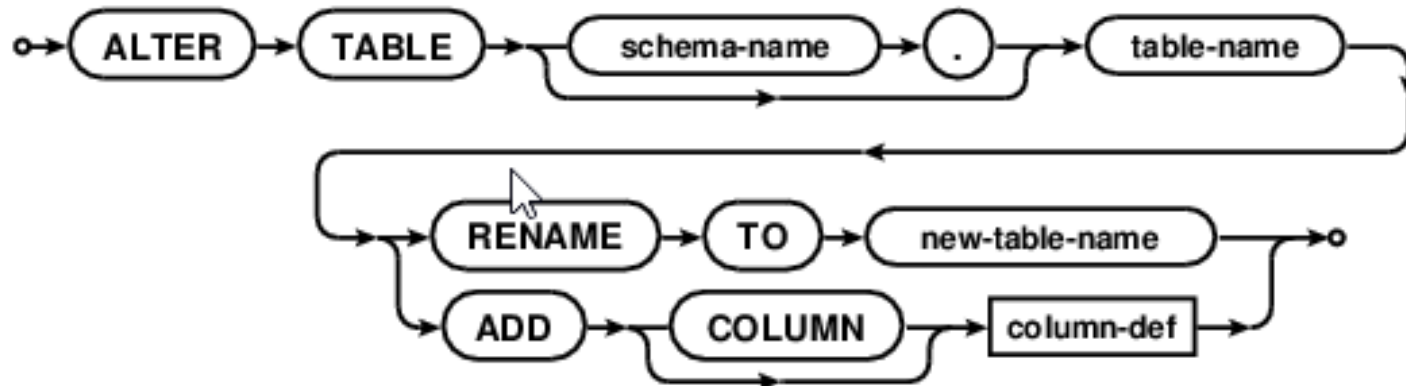
← Display error/warning messages for user

```
public void clearText() {
    editEnrollNo.setText("");
    editName.setText("");
    editMarks.setText("");
    editEnrollNo.requestFocus();
}
}
```

← Clear the **TextEdit** forms

Warning

SQLite supports a limited subset of ALTER TABLE. The ALTER TABLE command in SQLite allows the user to rename a table or to add a new column to an existing table.



Homework: add buttons and methods for *Next* and *Prev.* records in table.

Indications:

```
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    DisplayRecord(cursor);

}

public void NextRecord (View view){
    if (cursor.moveToNext())
    {
        DisplayRecord(cursor);
    }

}

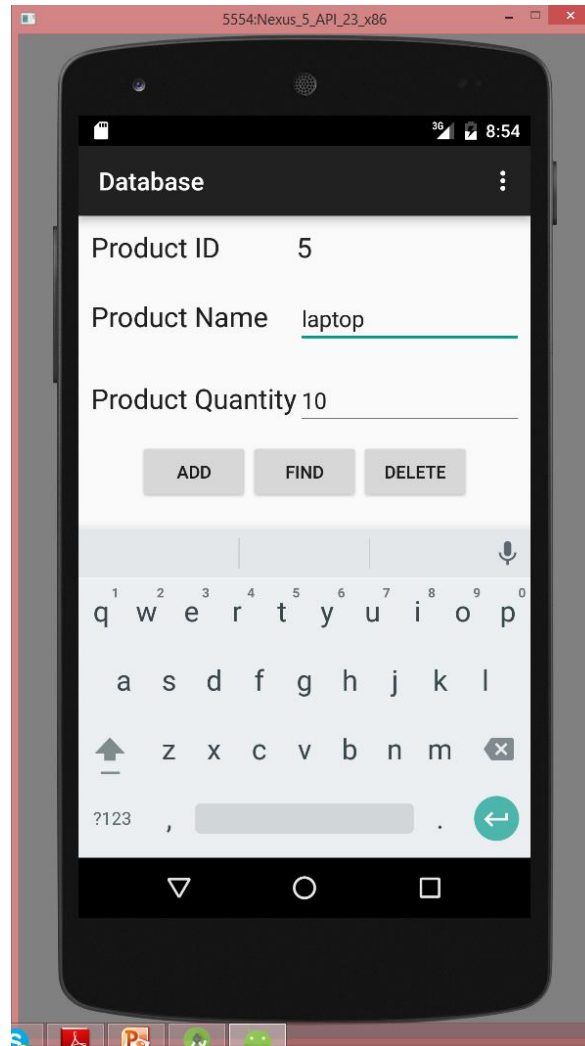
public void PreviousRecord (View view){
    if (cursor.moveToPrevious())
    {
        DisplayRecord(cursor);
    }
}

public void DisplayRecord(Cursor c) {
    EditText nameTxt = (EditText)findViewById(R.id.Name);
    EditText phoneTxt = (EditText)findViewById(R.id.Phone);
    EditText emailTxt = (EditText)findViewById(R.id.Email);
    nameTxt.setText(c.getString(1));
    phoneTxt.setText(c.getString(2));
    emailTxt.setText(c.getString(3));
}
```

**A new database example
more complicated but no more complex**

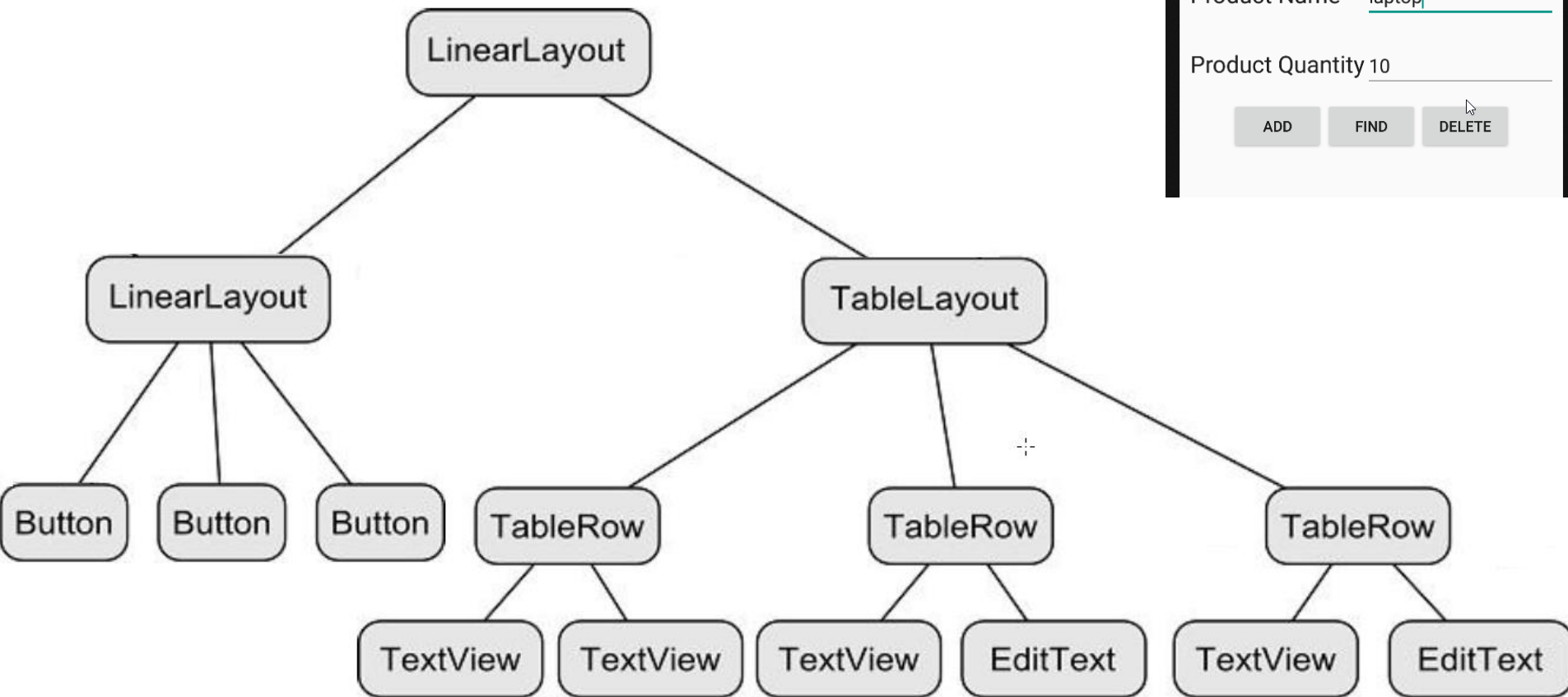
Goal:

An Android app for managing a products database (ID, Name and Quantity). The user's operations will be : *Add*, *Find* and *Delete* records.



← The user interface layout

The hierarchical tree structure of the application:



Database

Product ID 5

Product Name laptop

Product Quantity 10

ADD

FIND

DELETE

Creating the Database Project -steps-

1. Start Android Studio and create a new project, entering *Database* into the Application name field and *myname.com* as the Company Domain setting before clicking on the *Next* button.
2. Enable the *Phone and Tablet* option on the form factors screen.
3. Set the minimum SDK setting to API 8: Android 2.2 (Froyo).
4. Create a blank activity named *DatabaseActivity* with corresponding layout and menu resource files named *activity_database* and *menu_database* respectively.

The database schema for the *products* table is:

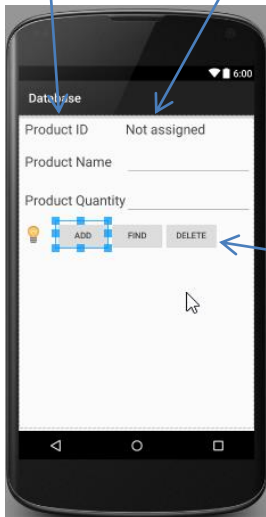
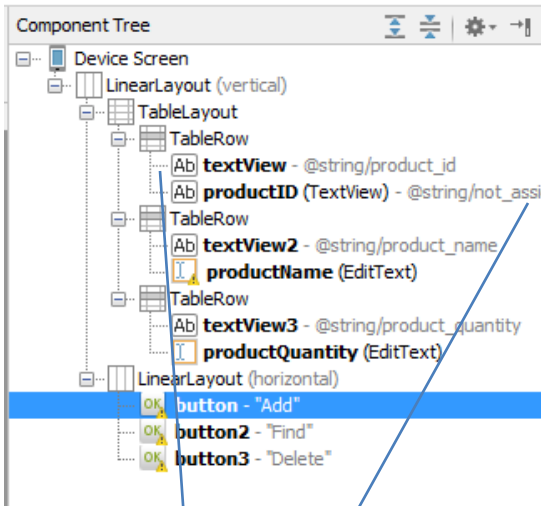
Column	Data Type
productid	Integer / Primary Key/ Auto Increment
productname	Text
productquantity	Integer

- The name of the database will be *productID.db* which contains a single table named *products*.
- Each record in the database table will contain a unique *product ID*, a *product description* and the *quantity of that product item* currently in stock, corresponding to column names of “*productid*”, “*productname*” and “*productquantity*” respectively.
- The *productid* column will act as the primary key and will be automatically assigned and incremented by the database management system.

Creating the Layout

Under the previous tree structure, using *Design* option to edit *activity_database.xml*.

The final form must be:



```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
```

```
<TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal">
```

```
<TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_margin="10dp">
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="@string/product_id"
    android:id="@+id/textView" />
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="@string/not_assigned"
    android:id="@+id/productID" />
```

```
<TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_margin="10dp">
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="@string/product_name"
    android:id="@+id/textView2" />
```

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/productName"
    android:layout_column="1" />
```

```
</TableRow>
```

```
<TableRow
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_margin="10dp">
```

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="@string/product_quantity"
    android:id="@+id/textView3" />
```

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="numberDecimal"
    android:ems="10"
    android:id="@+id/productQuantity" />
```

```
</TableRow>
</TableLayout>
```

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:layout_gravity="center_horizontal">
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Add"
    android:id="@+id/button"
    android:onClick="newProduct" />
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Find"
    android:id="@+id/button2"
    android:onClick="lookupProduct" />
```

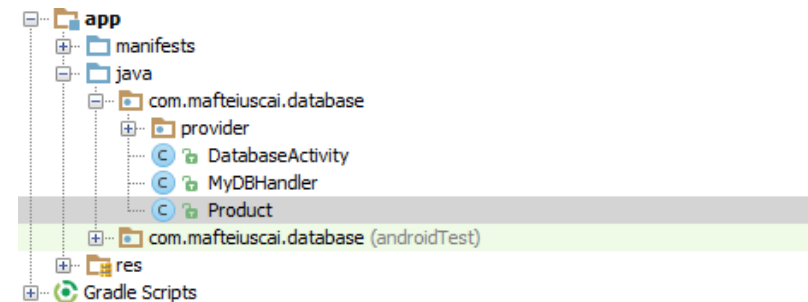
```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Delete"
    android:id="@+id/button3"
    android:onClick="removeProduct" />
```

```
</LinearLayout>
</LinearLayout>
```

Creating the Data Model

The database handler will be a subclass of SQLiteOpenHelper and will provide an abstract layer between the underlying SQLite database and the activity class. This subclass is for adding, removing and querying database entries.

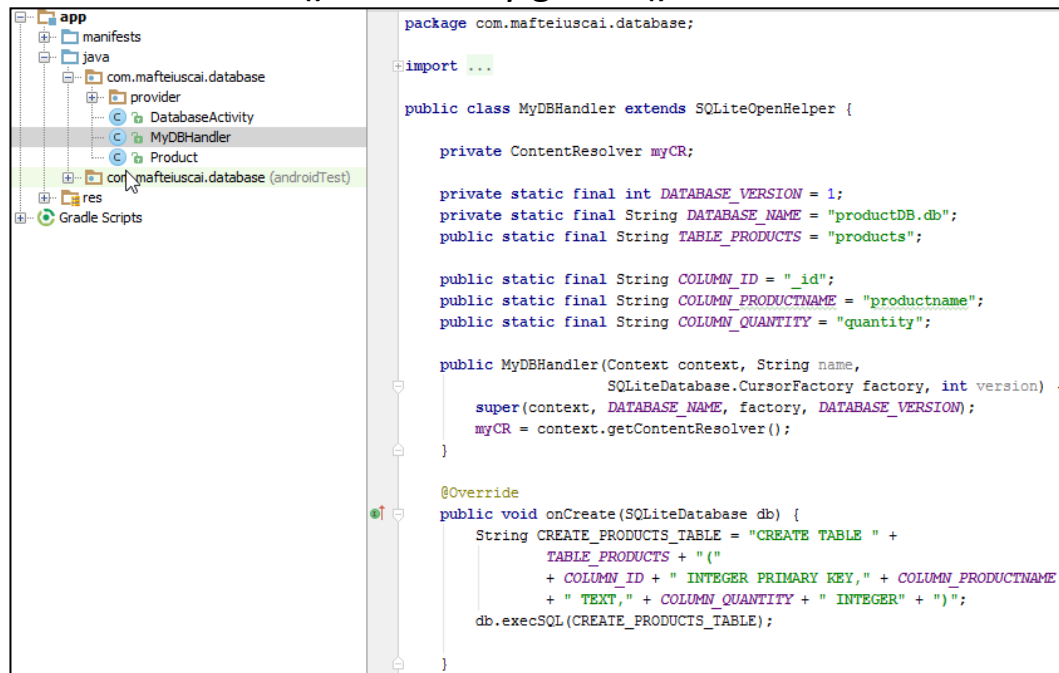
For this, create a new class in project, named *Product* (*Product.java* file):



```
public class Product {  
    private int _id;  
    private String _productname;  
    private int _quantity;  
  
    public Product() {  
    }  
  
    public Product(int id, String productname, int quantity) {  
        this._id = id;  
        this._productname = productname;  
        this._quantity = quantity;  
    }  
  
    public Product(String productname, int quantity) {  
        this._productname = productname;  
        this._quantity = quantity;  
    }  
  
    public void setID(int id) { this._id = id; }  
  
    public int getID() { return this._id; }  
  
    public void setProductName(String productname) { this._productname = productname; }  
  
    public String getProductName() { return this._productname; }  
  
    public void setQuantity(int quantity) { this._quantity = quantity; }  
  
    public int getQuantity() { return this._quantity; }  
}
```

Implementing the Data Handler

The data handler will be implemented by subclassing from the Android SQLiteOpenHelper class with *onCreate()* and *onUpgrade()* methods. So, add a new class named *MyDBHandler*:



The screenshot shows the Android Studio interface. On the left, the 'Project' view displays the file structure: 'app' contains 'manifests', 'java', and 'res'. The 'java' folder contains 'com.mafteiuscai.database', which includes 'DatabaseActivity', 'MyDBHandler', and 'Product'. The 'com.mafteiuscai.database' package is selected. The main editor shows the code for 'MyDBHandler.java'.

```
package com.mafteiuscai.database;

import ...

public class MyDBHandler extends SQLiteOpenHelper {

    private ContentResolver myCR;

    private static final int DATABASE_VERSION = 1;
    private static final String DATABASE_NAME = "productDB.db";
    public static final String TABLE_PRODUCTS = "products";

    public static final String COLUMN_ID = "_id";
    public static final String COLUMN_PRODUCTNAME = "productname";
    public static final String COLUMN_QUANTITY = "quantity";

    public MyDBHandler(Context context, String name,
        SQLiteDatabase.CursorFactory factory, int version) {
        super(context, DATABASE_NAME, factory, DATABASE_VERSION);
        myCR = context.getContentResolver();
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_PRODUCTS_TABLE = "CREATE TABLE " +
            TABLE_PRODUCTS + "(" +
                COLUMN_ID + " INTEGER PRIMARY KEY," + COLUMN_PRODUCTNAME
                + " TEXT," + COLUMN_QUANTITY + " INTEGER" + ")";
        db.execSQL(CREATE_PRODUCTS_TABLE);
    }
}
```

```
@Override
public void onCreate(SQLiteDatabase db) {
    String CREATE_PRODUCTS_TABLE = "CREATE TABLE " +
        TABLE_PRODUCTS + "(" +
            COLUMN_ID + " INTEGER PRIMARY KEY," + COLUMN_PRODUCTNAME
            + " TEXT," + COLUMN_QUANTITY + " INTEGER" + ")";
    db.execSQL(CREATE_PRODUCTS_TABLE);
}

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

public void addProduct(Product product) {

    ContentValues values = new ContentValues();
    values.put(COLUMN_PRODUCTNAME, product.getProductName());
    values.put(COLUMN_QUANTITY, product.getQuantity());

    myCR.insert(MyContentProvider.CONTENT_URI, values);
}
```

```
public Product findProduct(String productname) {
    String[] projection = {COLUMN_ID,
        COLUMN_PRODUCTNAME, COLUMN_QUANTITY};

    String selection = "productname = \"" + productname + "\"";

    Cursor cursor = myCR.query(MyContentProvider.CONTENT_URI,
        projection, selection, null,
        null);

    Product product = new Product();

    if (cursor.moveToFirst()) {
        cursor.moveToFirst();
        product.setID(Integer.parseInt(cursor.getString(0)));
        product.setProductName(cursor.getString(1));
        product.setQuantity(Integer.parseInt(cursor.getString(2)));
        cursor.close();
    } else {
        product = null;
    }
    return product;
}
```

```
public boolean deleteProduct(String productname) {

    boolean result = false;

    String selection = "productname = \"" + productname + "\"";

    int rowsDeleted = myCR.delete(MyContentProvider.CONTENT_URI,
        selection, null);

    if (rowsDeleted > 0)
        result = true;

    return result;
}
```

Implementing the Activity Event Methods

In *DatabaseActivity.java* source file implement the code to identify the views in the user interface and to implement the three “on Click” target methods:

```
package com.mafteiuscai.database;

import ...

public class DatabaseActivity extends ActionBarActivity {

    TextView idView;
    EditText productBox;
    EditText quantityBox;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_database);
        idView = (TextView) findViewById(R.id.productID);
        productBox = (EditText) findViewById(R.id.productName);
        quantityBox =
            (EditText) findViewById(R.id.productQuantity);
    }

    public void newProduct (View view) {
        MyDBHandler dbHandler = new MyDBHandler(this, null, null, 1);

        int quantity =
            Integer.parseInt(quantityBox.getText().toString());

        Product product =
            new Product(productBox.getText().toString(), quantity);

        dbHandler.addProduct(product);
        productBox.setText("");
        quantityBox.setText("");
    }
}
```

```
public void lookupProduct (View view) {
    MyDBHandler dbHandler = new MyDBHandler(this, null, null, 1);

    Product product =
        dbHandler.findProduct(productBox.getText().toString());

    if (product != null) {
        idView.setText(String.valueOf(product.getID()));

        quantityBox.setText(String.valueOf(product.getQuantity()));
    } else {
        idView.setText("No Match Found");
    }
}

public void removeProduct (View view) {
    MyDBHandler dbHandler = new MyDBHandler(this, null,
        null, 1);

    boolean result = dbHandler.deleteProduct(
        productBox.getText().toString());

    if (result)
    {
        idView.setText("Record Deleted");
        productBox.setText("");
        quantityBox.setText("");
    }
    else
        idView.setText("No Match Found");
}
}
```

Ta-Ta for now!