

## Lab07 - Dùng SQLite giao tiếp với ListView

### **Giới thiệu SQLite.**

Trong Android, SQLite là một cơ sở dữ liệu quan hệ nhẹ (lightweight relational database) được tích hợp sẵn vào hệ thống. Nó cung cấp một cách để lưu trữ và truy xuất dữ liệu trong ứng dụng Android. SQLite được sử dụng phổ biến trong việc lưu trữ dữ liệu cục bộ như cài đặt ứng dụng, dữ liệu người dùng, và các loại dữ liệu khác mà ứng dụng cần lưu trữ và truy xuất trong quá trình hoạt động. SQLite được thiết kế để đơn giản, nhẹ nhàng và phản hồi tốt, làm cho nó trở thành lựa chọn phổ biến cho việc phát triển ứng dụng Android.

### **Các bước cơ bản khi làm việc với SQLite.**

Để sử dụng SQLite trong ứng dụng Android, bạn cần thực hiện các bước sau:

- **Thêm Dependency:** Thêm thư viện SQLite vào dự án Android của bạn. Thông thường, bạn sẽ sử dụng thư viện được cung cấp sẵn trong Android SDK.
- **Tạo hoặc Mở Cơ sở dữ liệu:** Trước tiên, bạn cần tạo hoặc mở cơ sở dữ liệu SQLite. Điều này thường được thực hiện trong lớp SQLiteOpenHelper.
- **Định nghĩa Cấu trúc Dữ liệu:** Xác định cấu trúc của cơ sở dữ liệu, bao gồm các bảng và các cột trong bảng. Bạn cần phải tạo các bảng và định nghĩa các cột cho dữ liệu của bạn.
- **Thực hiện Truy vấn:** Sử dụng SQL để thực hiện các truy vấn để thêm, sửa đổi, xóa hoặc truy vấn dữ liệu từ cơ sở dữ liệu.
- **Xử lý Kết quả:** Xử lý kết quả trả về từ các truy vấn SQLite. Điều này bao gồm việc đọc dữ liệu từ các bảng và xử lý chúng theo cách mà ứng dụng của bạn cần.
- **Đóng Cơ sở dữ liệu:** Khi bạn đã hoàn thành việc sử dụng cơ sở dữ liệu, hãy đảm bảo rằng bạn đóng nó để giải phóng tài nguyên và tránh lỗi.

- **Xử lý Ngoại lệ:** Đảm bảo xử lý các ngoại lệ liên quan đến cơ sở dữ liệu một cách chính xác, chẳng hạn như việc xử lý các trường hợp cơ sở dữ liệu không tồn tại hoặc không mở được.
- **Kiểm tra Quyền truy cập:** Trong Android, bạn cần xác định quyền truy cập vào cơ sở dữ liệu trong tệp Manifest của ứng dụng của bạn.

**Các lớp cơ bản khi làm việc với SQLite trong Android bao gồm:**

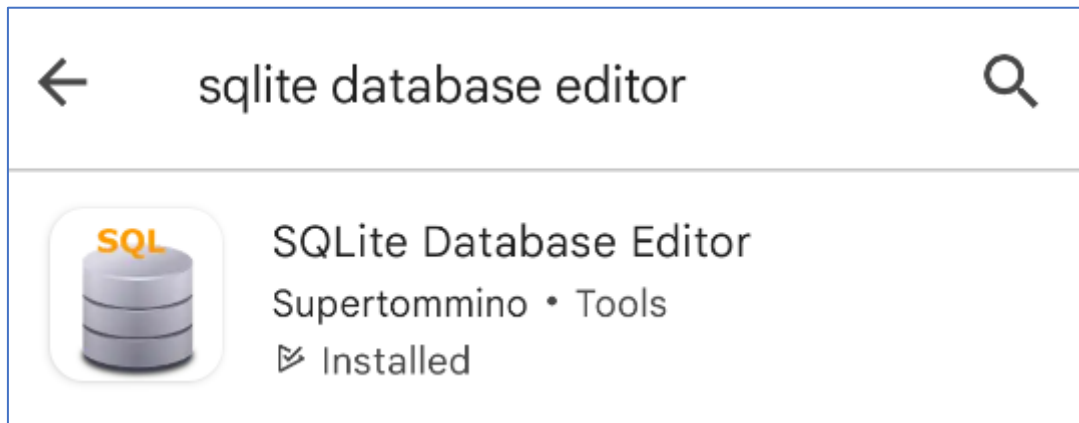
- **SQLiteOpenHelper:** Là một lớp trợ giúp (helper class) để quản lý cơ sở dữ liệu SQLite. Lớp này cung cấp các phương thức để tạo và cập nhật cơ sở dữ liệu, cũng như cung cấp một đối tượng SQLiteDatabase để thao tác với cơ sở dữ liệu.
- **SQLiteDatabase:** Là lớp để thực hiện các hoạt động trên cơ sở dữ liệu SQLite, bao gồm việc thực thi các truy vấn SQL, thêm, sửa đổi hoặc xóa dữ liệu. Lớp này cung cấp các phương thức để thực hiện các hoạt động này và quản lý giao tiếp với cơ sở dữ liệu.
- **Cursor:** Là một đối tượng để duyệt qua các dòng kết quả của một truy vấn SQLite. Cursor cho phép bạn lấy dữ liệu từ cơ sở dữ liệu bằng cách di chuyển qua các dòng kết quả và truy cập vào các giá trị trong mỗi dòng.
- **ContentValues:** Là một lớp được sử dụng để đại diện cho một tập hợp các giá trị để thêm hoặc cập nhật vào cơ sở dữ liệu. ContentValues được sử dụng khi bạn muốn thêm hoặc cập nhật dữ liệu trong cơ sở dữ liệu bằng cách sử dụng phương thức insert() hoặc update() của lớp SQLiteDatabase.

Các lớp này là những phần cơ bản và quan trọng khi làm việc với SQLite trong Android. Bằng cách sử dụng chúng một cách hiệu quả, bạn có thể thực hiện các thao tác cơ bản như tạo, mở, đóng cơ sở dữ liệu, thực thi truy vấn, và quản lý dữ liệu trong ứng dụng của mình.

## Chuẩn bị:

Dùng SQLite Database Editor để kiểm tra xem cơ sở dữ liệu có tạo thành công hay không.

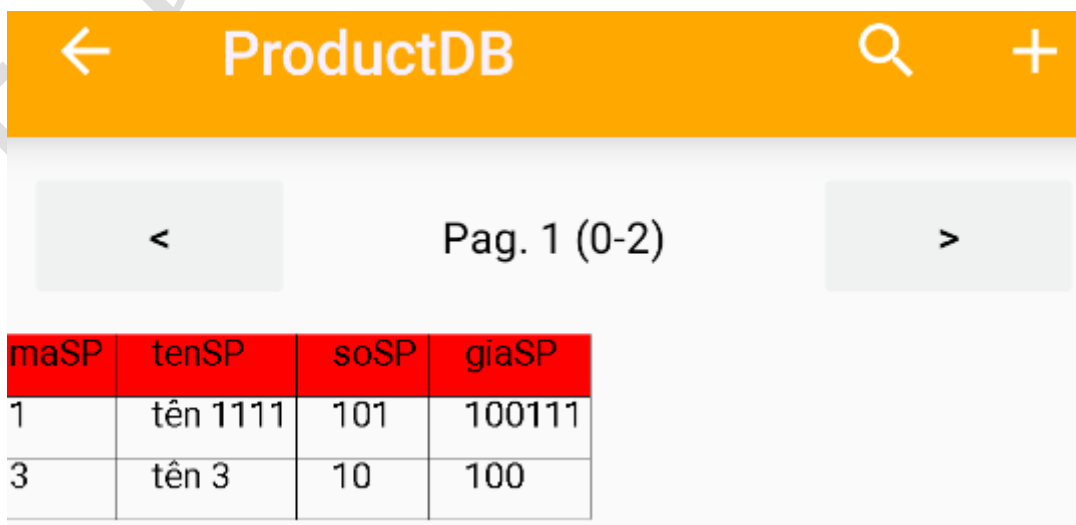
- Mở **CH Play** trên điện thoại, tìm và cài đặt SQLite Database Editor.



Sau khi cài đặt xong, thực thi ứng dụng có sử dụng SQLite, mở ứng dụng SQLite Database Editor, tìm trong mục **App List**

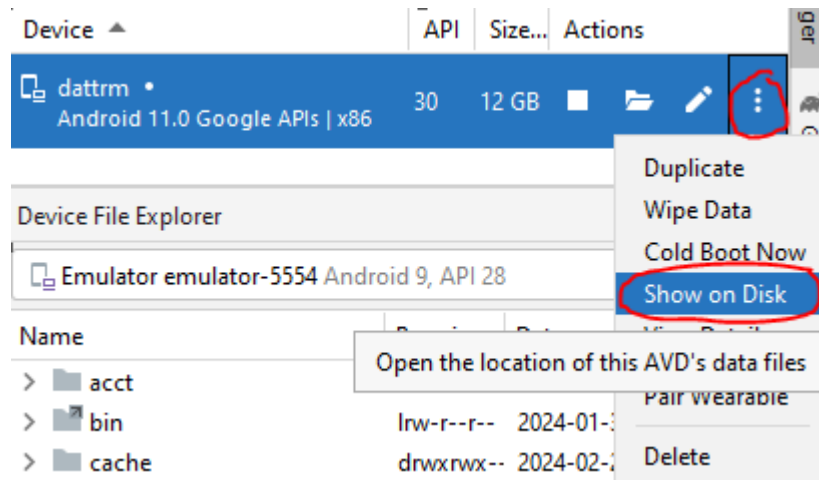


Tìm trong danh sách để mở ứng dụng có sử dụng, nếu thấy xuất hiện bảng dữ liệu tức là ứng dụng đã tạo được dữ liệu, ví dụ:

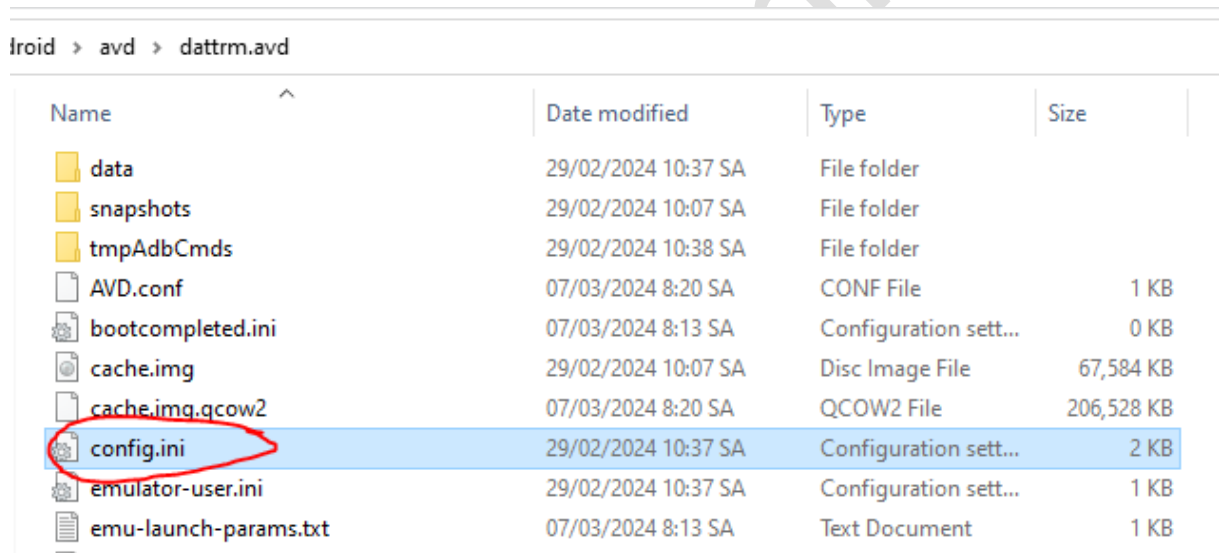


Trong trường hợp **máy ảo không có CH Play** thì thực hiện như sau:

1. Mở file config.ini của máy ảo tại địa chỉ:



Tìm và mở file **config.ini** bằng NotePad:



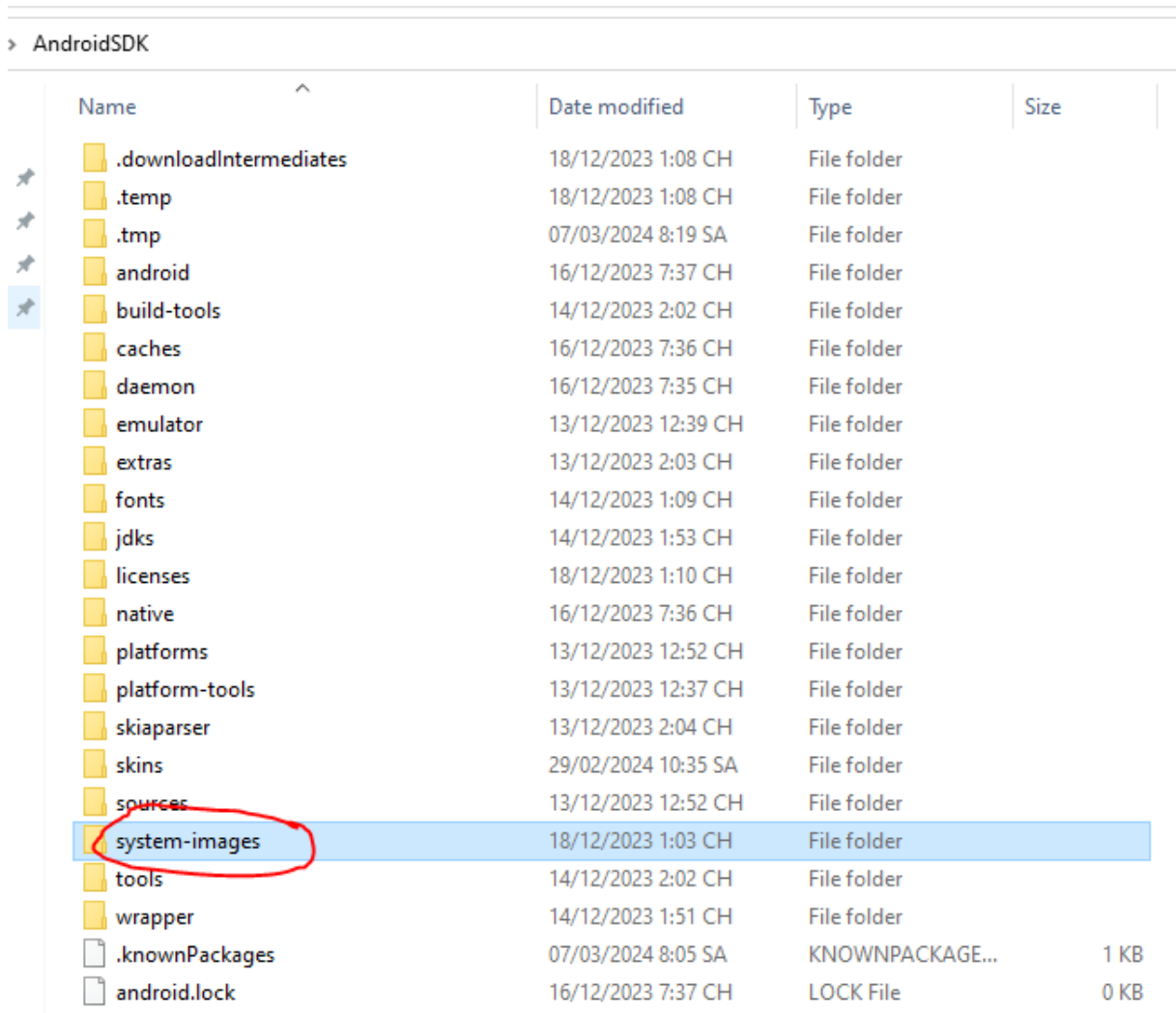
Tìm 2 dòng như trong ảnh:

```
image.sysdir.1 = system-images\android-30\google_api\x86\
runtime.network.latency = none
runtime.network.speed = full
sdcard.size = 512M
showDeviceFrame = yes
skin.dynamic = yes
skin.name = pixel_5
skin.path = D:\AndroidSDK\skins\pixel_5
```

The image shows two red annotations: a red circle around 'system-images\android-30' and a red circle around 'D:\AndroidSDK\skins\pixel\_5'. A red number '1' is written next to the first line, and a red number '2' is written next to the last line.

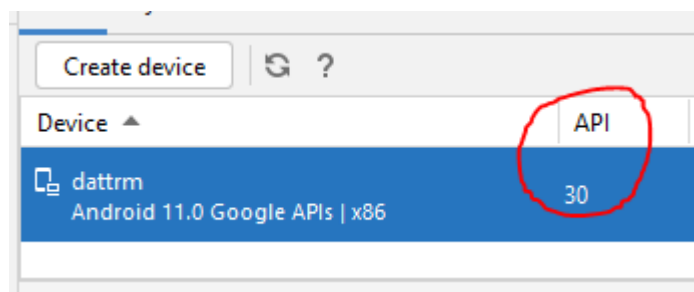
Trong đó:

**Dòng 2:** Thư mục cài đặt SDK của máy tính, mở thư mục này để tìm thư mục trên dòng 1 system-images:



AndroidSDK				
Name	Date modified	Type	Size	
.downloadIntermediates	18/12/2023 1:08 CH	File folder		
.temp	18/12/2023 1:08 CH	File folder		
.tmp	07/03/2024 8:19 SA	File folder		
android	16/12/2023 7:37 CH	File folder		
build-tools	14/12/2023 2:02 CH	File folder		
caches	16/12/2023 7:36 CH	File folder		
daemon	16/12/2023 7:35 CH	File folder		
emulator	13/12/2023 12:39 CH	File folder		
extras	13/12/2023 2:03 CH	File folder		
fonts	14/12/2023 1:09 CH	File folder		
jdk	14/12/2023 1:53 CH	File folder		
licenses	18/12/2023 1:10 CH	File folder		
native	16/12/2023 7:36 CH	File folder		
platforms	13/12/2023 12:52 CH	File folder		
platform-tools	13/12/2023 12:37 CH	File folder		
skiaarser	13/12/2023 2:04 CH	File folder		
skins	29/02/2024 10:35 SA	File folder		
sources	13/12/2023 12:52 CH	File folder		
system-images	18/12/2023 1:03 CH	File folder		
tools	14/12/2023 2:02 CH	File folder		
wrapper	14/12/2023 1:51 CH	File folder		
.knownPackages	07/03/2024 8:05 SA	KNOWNPACKAGE...	1 KB	
android.lock	16/12/2023 7:37 CH	LOCK File	0 KB	

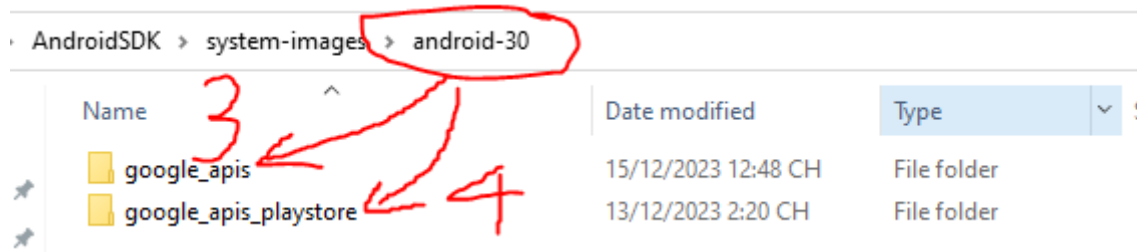
Mở thư mục này và tìm đúng phiên bản API đang cài của máy ảo, có thể tìm phiên bản trong Device Manager của Android Studio:



Create device		?
Device	API	
dattm Android 11.0 Google APIs   x86	30	1

hoặc dòng 1 chính là phiên bản API của máy ảo.

Mở thư mục của dòng 1:



Trong dòng 1 đang có là tên thư mục 3, cần thay nó bằng tên thư mục của dòng 4:

`image.sysdir.1 = system-images\android-30\google_api\x86\`

trở thành:

`image.sysdir.1 = system-images\android-30\google_api_playstore\x86\`

Lưu file config.ini vừa sửa, khởi động lại máy ảo và Android Studio sẽ thấy xuất hiện CH Play.

## Bài toán

Xây dựng ứng dụng quản lý sản phẩm (Mã sản phẩm, Tên sản phẩm, Số lượng sản phẩm, Giá sản phẩm)

Ví dụ giao diện ứng dụng:

QUẢN LÝ SẢN PHẨM

Mã sản phẩm:

editTextMaSP

Tên sản phẩm:

editTextTenSP

Số lượng sản phẩm:

editTextSoSP

Giá sản phẩm:

editTextGiaSP

THÊM

SỬA

XÓA

listViewSanPham

QUẢN LÝ SẢN PHẨM

Mã sản phẩm:

Nhập mã sản phẩm...

Tên sản phẩm:

Nhập tên sản phẩm...

Số lượng sản phẩm:

Số lượng...

Giá sản phẩm:

Giá...

THÊM

SỬA

XÓA

3 - so1 - 10 - 1001.0

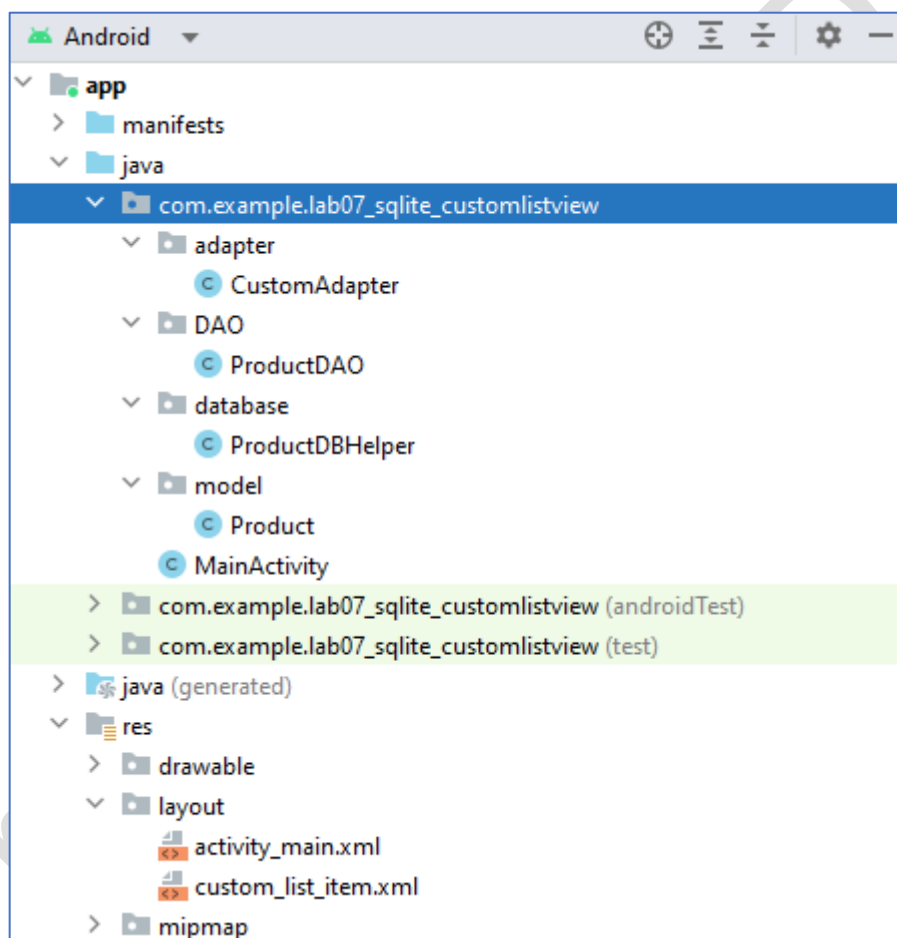
Sử dụng các EditText để nhập dữ liệu, sử dụng ListView để hiển thị dữ liệu.

Các nút chức năng tương ứng:

- Thêm: Thêm dữ liệu vào CSDL
- Sửa: Sửa dữ liệu
- Xóa: Xóa dữ liệu.

## Hướng dẫn thực hiện

Ví dụ dưới đây bố trí các class và layout như sau:



Đường đi của luồng dữ liệu trong ví dụ này có thể mô tả như sau:

- Người dùng tương tác với giao diện người dùng (trong ví dụ là activity\_main.xml), chẳng hạn như nhấn nút để thêm, sửa hoặc xóa sản phẩm.
- Sự kiện người dùng được gửi đến Activity tương ứng (trong ví dụ là MainActivity).

- Trong Activity, sự kiện được xử lý và chuyển cho CustomAdapter để xử lý sự kiện cụ thể như thêm, sửa hoặc xóa sản phẩm.
- CustomAdapter gửi yêu cầu tương ứng (thêm, sửa hoặc xóa sản phẩm) cho ProductDAO.
- Trong ProductDAO, các phương thức thích hợp được gọi để thực hiện thêm, sửa hoặc xóa dữ liệu trong cơ sở dữ liệu (ProductDBHelper).
- Sau khi hoàn thành, ProductDAO trả lại kết quả cho CustomAdapter.
- CustomAdapter cập nhật dữ liệu và thông báo cho Activity về bất kỳ thay đổi nào trong danh sách sản phẩm.
- Activity cập nhật giao diện người dùng dựa trên dữ liệu mới từ CustomAdapter.

## 1. Thiết kế giao diện trong activity\_main.xml.

- Dùng các EditText để nhập dữ liệu.
- Các Button: Thêm, Sửa, Xóa.
- ListView để hiển thị dữ liệu.

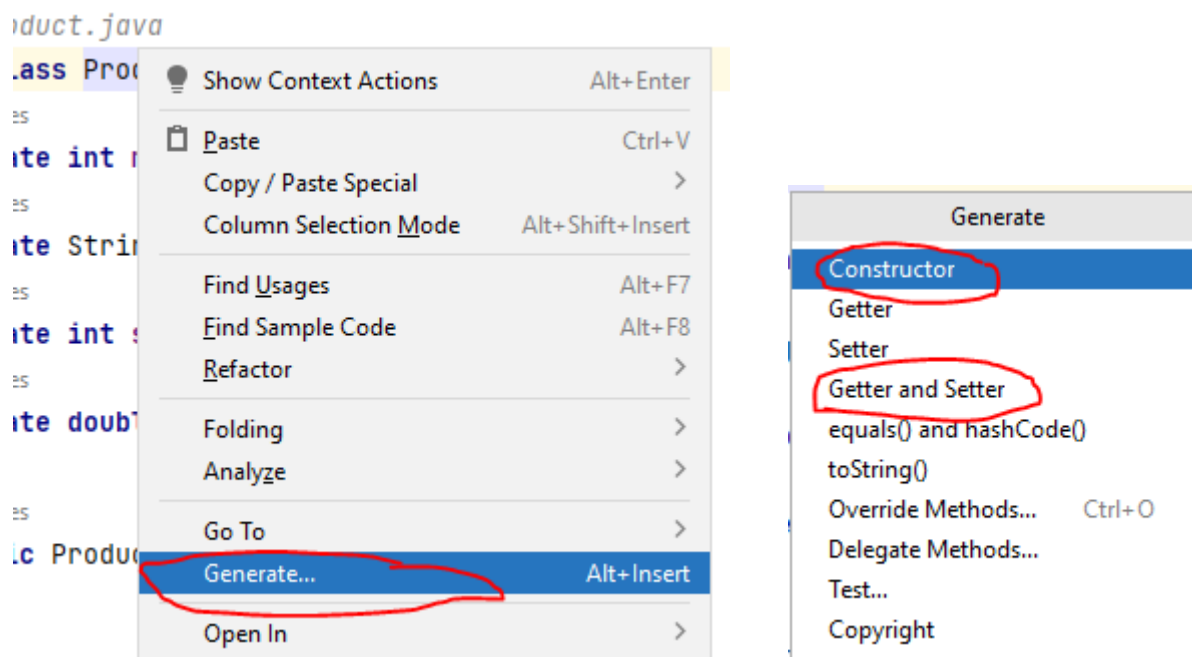
## 2. Tạo model Product.java tương ứng

Bài toán cần làm việc với Sản phẩm (Mã, tên, số lượng, giá) nên cũng cần tạo lớp đối tượng có các thuộc tính tương ứng

```
//Lớp Product.java
public class Product {
    3 usages
    private int maSP;
    3 usages
    private String tenSP;
    3 usages
    private int soSP;
    3 usages
    private double giaSP;
```

Sau khi tạo thuộc tính, tiếp tục tạo các Constructor, Getter và Setter:

Tạo nhanh bằng cách click chuột phải, chọn **Generate...**



### **3. Tạo lớp ProductDBHelper.java để quản lý cơ sở dữ liệu.**

Lớp **ProductDBHelper** trong bài thực hành này là một lớp được tạo ra từ SQLiteOpenHelper dùng để quản lý cơ sở dữ liệu SQLite tương ứng cho đối tượng Product vừa tạo. Một số chức năng của lớp này:

Tạo hoặc cập nhật cơ sở dữ liệu: Phương thức onCreate được gọi khi cơ sở dữ liệu được tạo lần đầu tiên. Trong phương thức này, bạn định nghĩa cấu trúc của các bảng trong cơ sở dữ liệu và có thể thêm dữ liệu mẫu vào bảng. Phương thức onUpgrade được gọi khi phiên bản cơ sở dữ liệu đã tồn tại và có một phiên bản mới được cài đặt. Trong phương thức này, bạn có thể thực hiện các thay đổi cơ bản, như thêm hoặc xóa bảng, khi cập nhật cơ sở dữ liệu.

Quản lý kết nối đến cơ sở dữ liệu: Lớp ProductDBHelper mở kết nối đến cơ sở dữ liệu khi được khởi tạo và đóng kết nối khi không cần thiết nữa. Khi bạn cần truy cập đến cơ sở dữ liệu, bạn có thể tạo một thể hiện của ProductDBHelper và sử dụng nó để thực hiện các thao tác đọc và ghi dữ liệu.

Thực hiện các lệnh SQL: Lớp ProductDBHelper cho phép bạn thực thi các lệnh SQL trên cơ sở dữ liệu, bao gồm việc tạo bảng, thêm, sửa, xóa dữ liệu.

Trong đoạn mã này, lớp ProductDBHelper được sử dụng để tạo một cơ sở dữ liệu có tên là "ProductDB" với một bảng có tên là "tableProduct". Bảng này có các cột maSP, tenSP, soSP, và giaSP. Sau đó, bạn thêm một số dữ liệu mẫu vào bảng trong phương thức onCreate. Nếu có bất kỳ thay đổi nào trong cấu trúc của bảng khi cập nhật ứng dụng, phương thức onUpgrade sẽ được gọi để thực hiện cập nhật cơ sở dữ liệu.

```
3 usages
7 public class ProductDBHelper extends SQLiteOpenHelper {
8     1 usage
9     private static final String DB_NAME = "ProductDB";
10    1 usage
11    private static final int DB_VERSION = 1;
12    1 usage
13    public ProductDBHelper(Context context){
14        //Tạo cơ sở dữ liệu
15        super(context,DB_NAME, factory: null,DB_VERSION);
16    }
17    @Override
18    public void onCreate(SQLiteDatabase sqLiteDatabase) {
19        //Lệnh SQL tạo bảng dữ liệu
20        String create_table_sql="CREATE TABLE tableProduct(\n" +
21            "\tmaSP integer PRIMARY KEY,\n" +
22            "\ttenSP text,\n" +
23            "\tsoSP integer,\n" +
24            "\tgiaSP double\n" +
25            ")";
26        //Thực thi câu lệnh tạo bảng
27        sqLiteDatabase.execSQL(create_table_sql);
28        //Lệnh SQL thêm dữ liệu mẫu vào bảng
29        //Có thể thêm trực tiếp từ layout, không cần có bước thêm dữ liệu mẫu này
30        String insert_table_sql = "INSERT INTO tableProduct VALUES (1,'ten1',12,13),(2,'ten2',22,23)";
31        sqLiteDatabase.execSQL(insert_table_sql);
32    }
33    10 usages
34    @Override
35    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {
36        if(oldVersion!=newVersion){
37            sqLiteDatabase.execSQL("DROP TABLE IF EXISTS tableProduct");
38            onCreate(sqLiteDatabase);
39        }
40    }
41 }
```

Ghi chú: Có thể kiểm tra các câu lệnh SQLite có thực thi chính xác không tại địa chỉ:

<https://sqliteonline.com/>

#### 4. Tạo lớp **ProductDAO.java** để tương tác với cơ sở dữ liệu.

Trong ngữ cảnh phát triển phần mềm, DAO là viết tắt của Data Access Object. DAO thường được sử dụng để quản lý tất cả các thao tác liên quan đến cơ sở dữ liệu SQLite, bao gồm thêm, sửa, xóa, và truy vấn dữ liệu. Điều này giúp tách biệt logic của ứng dụng khỏi logic truy cập cơ sở dữ liệu, làm cho mã trở nên dễ đọc hơn và dễ bảo trì hơn.

Các thành phần chính của một DAO bao gồm:

- **Interface hoặc lớp trừu tượng:** Định nghĩa các phương thức trừu tượng cho việc truy cập dữ liệu, chẳng hạn như thêm, sửa, xóa, hoặc truy vấn dữ liệu.
- **Các lớp cụ thể của DAO:** Các lớp này cài đặt các phương thức được định nghĩa trong interface hoặc lớp trừu tượng, và chịu trách nhiệm thực hiện các thao tác thực tế với cơ sở dữ liệu.

Lớp **ProductDAO** dưới đây là một lớp thiết kế để tương tác với cơ sở dữ liệu SQLite. Đây là một số chức năng của lớp này:

- **Khởi tạo và quản lý kết nối đến cơ sở dữ liệu:** Trong hàm khởi tạo của lớp, bạn tạo một thể hiện của lớp ProductDBHelper để tạo hoặc mở kết nối đến cơ sở dữ liệu. Bằng cách này, bạn có thể sử dụng các phương thức của SQLiteDatabase để thực hiện các thao tác với cơ sở dữ liệu như thêm, xóa, sửa, hoặc truy vấn dữ liệu.
- **Thực hiện các thao tác CRUD (Create, Read, Update, Delete) trên cơ sở dữ liệu:**
  - ✓ Phương thức **addProduct** được sử dụng để thêm một sản phẩm mới vào cơ sở dữ liệu.
  - ✓ Phương thức **deleteProduct** được sử dụng để xóa một sản phẩm khỏi cơ sở dữ liệu dựa trên mã sản phẩm.
  - ✓ Phương thức **updateProduct** được sử dụng để cập nhật thông tin của một sản phẩm đã tồn tại trong cơ sở dữ liệu.
  - ✓ Phương thức **getListProduct** được sử dụng để truy vấn và lấy danh sách các sản phẩm từ cơ sở dữ liệu.

- **Sử dụng đối tượng ContentValues để thêm và cập nhật dữ liệu:** Đối tượng ContentValues được sử dụng để đóng gói các cặp "key-value", trong đó "key" là tên của cột và "value" là giá trị tương ứng. Đối tượng này được sử dụng trong các phương thức addProduct và updateProduct để thêm hoặc cập nhật dữ liệu vào cơ sở dữ liệu.
- **Xử lý ngoại lệ và ghi log:** Trong phương thức getListProduct, bạn đã bao quanh các thao tác với cơ sở dữ liệu trong một khối try-catch để bắt và xử lý các ngoại lệ. Nếu có lỗi xảy ra trong quá trình truy vấn dữ liệu, bạn ghi log lỗi để dễ dàng theo dõi và gỡ lỗi.

```
public class ProductDAO {
    private ProductDBHelper productDBHelper;
    private SQLiteDatabase sqLiteDatabase;
    public ProductDAO(Context context){
        productDBHelper = new ProductDBHelper(context); //Gọi lệnh tạo DB
        sqLiteDatabase = productDBHelper.getWritableDatabase();
    }
    //Hàm Thêm: CREATE (C trong CRUD)
    public long addProduct(Product product){
        ContentValues contentValues = new ContentValues();
        contentValues.put("maSP", product.getMaSP());
        contentValues.put("tenSP", product.getTenSP());
        contentValues.put("soSP", product.getSoSP());
        contentValues.put("giaSP", product.getGiaSP());
        long checkDB=
        sqLiteDatabase.insert("tableProduct", null, contentValues);
        return checkDB;
    }
    //Hàm xóa: DELETE (D trong CRUD)
    public long deleteProduct(int maSP){
        long checkDB=sqLiteDatabase.delete("tableProduct", "maSP=?", new
        String[]{String.valueOf(maSP)});
        return checkDB;
    }
    //Hàm sửa: UPDATE (U trong CRUD)
    public long updateProduct(Product product){
        ContentValues contentValues = new ContentValues();
        contentValues.put("maSP", product.getMaSP());
        contentValues.put("tenSP", product.getTenSP());
        contentValues.put("soSP", product.getSoSP());
        contentValues.put("giaSP", product.getGiaSP());
```

```

        long
        checkDB=sqLiteDatabase.update("tableProduct",contentValues,"maSP=?",new
        String[]{String.valueOf(product.getMaSP())});
        return checkDB;
    }
    //Hàm đọc dữ liệu:READ (R trong CRUD)
    //Có thể dùng List hoặc ArrayList
    public ArrayList<Product> getListProduct(){
        ArrayList<Product> productArrayList = new ArrayList<>();
        sqLiteDatabase = productDBHelper.getReadableDatabase();
        try {
            Cursor cursor=sqLiteDatabase.rawQuery("SELECT * from
tableProduct",null);
            if (cursor.getCount(>0){
                cursor.moveToFirst();
                do {
                    productArrayList.add(new Product(cursor.getInt(0),//Mã sản
phẩm kiểu int
                                cursor.getString(1),//Tên sản phẩm kiểu String
                                cursor.getInt(2),//Số lượng sản phẩm kiểu int
                                cursor.getLong(3)));//Giá sản phẩm kiểu long
                }while (cursor.moveToNext());
            }
        }catch (Exception e){
            Log.e("TAG",e.getMessage());
        }
        return productArrayList;
    }
}

```

## **5. Xây dựng lớp Adapter – CustumAdapter.java**

Lưu ý: Trong CustomAdapter có sử dụng layout custom\_list\_item.xml nên cần thiết kế trước layout này. Layout gồm:

//textViewMaSP hiển thị mã Sản phẩm

//textViewTenSP hiển thị tên sản phẩm

//textViewSoSP hiển thị số lượng sản phẩm

//textViewGiaSP hiển thị giá sản phẩm

```

public class CustomAdapter extends BaseAdapter {
    private Context mContext;
    private ArrayList<Product> mProductList;

    public CustomAdapter(Context context, ArrayList<Product> productList) {
        mContext = context;
        mProductList = productList;
    }

    @Override
    public int getCount() {
        return mProductList.size();
    }

    @Override
    public Object getItem(int position) {
        return mProductList.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent){
        View view = convertView;
        if (view==null){
            LayoutInflater inflater = (LayoutInflater)
mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            view = inflater.inflate(R.layout.custom_list_item,null);
            //custom_list_item là layout tùy chỉnh của ListView dùng để hiển thị
nội dung
        }
        // Ánh xạ các thành phần trong layout của mỗi item trong ListView
        TextView textViewMaSP = view.findViewById(R.id.textViewMaSP);
        TextView textViewTenSP = view.findViewById(R.id.textViewTenSP);
        TextView textViewSoSP = view.findViewById(R.id.textViewSoSP);
        TextView textViewGiaSP = view.findViewById(R.id.textViewGiaSP);
        // Lấy đối tượng Product tương ứng với vị trí position
        Product product = mProductList.get(position);

        // Thiết lập dữ liệu cho các thành phần trong layout của mỗi item
        textViewMaSP.setText(String.valueOf(product.getMaSP()));
    }
}

```

```

        textViewTenSP.setText(product.getTenSP());
        textViewSoSP.setText(String.valueOf(product.getSoSP()));
        textViewGiaSP.setText(String.valueOf(product.getGiaSP()));

        // Tùy chỉnh định dạng các thành phần trong layout tùy chỉnh của mỗi item
        // Ví dụ: Thiết lập màu sắc, kích thước, font chữ, v.v.
        textViewMaSP.setTextColor(Color.RED);
        textViewTenSP.setTextSize(TypedValue.COMPLEX_UNIT_SP, 18); // Kích thước
chữ là 18sp
        // Các tùy chỉnh khác tùy thuộc vào yêu cầu cụ thể của bạn

        return view;
    }
}

```

## **6. Tạo lớp xử lý chính MainActivity.java**

```

public class MainActivity extends AppCompatActivity {
    EditText editTextMaSP, editTextTenSP, editTextSoSP, editTextGiaSP;
    Button buttonThem, buttonSua, buttonXoa;
    ListView listViewDSSP;

    ArrayList<Product> productArrayList = new ArrayList<>();
    ProductDAO productDAO;
    Context context = this;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        editTextMaSP = findViewById(R.id.editTextMaSP);
        editTextTenSP = findViewById(R.id.editTextTenSP);
        editTextSoSP = findViewById(R.id.editTextSoSP);
        editTextGiaSP = findViewById(R.id.editTextGiaSP);
        buttonThem = findViewById(R.id.buttonThem);
        buttonSua = findViewById(R.id.buttonSua);
        buttonXoa = findViewById(R.id.buttonXoa);
        listViewDSSP = findViewById(R.id.listViewSanPham);

        productDAO = new ProductDAO(context);
        productArrayList = productDAO.getListProduct();

        // Tạo adapter tùy chỉnh và thiết lập cho ListView
        CustomAdapter customAdapter = new CustomAdapter(context, productArrayList);
    }
}

```

```
listViewDSSP.setAdapter(customAdapter);
```

```
// Chức năng nút Thêm
```

```
buttonThem.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
```

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

```
        // Thêm sản phẩm mới vào cơ sở dữ liệu
```

```
        Product product = new Product();
```

```
        product.setMaSP(Integer.parseInt(editTextMaSP.getText().toString()));
```

```
        product.setTenSP(editTextTenSP.getText().toString());
```

```
        product.setSoSP(Integer.parseInt(editTextSoSP.getText().toString()));
```

```
        product.setGiaSP(Long.parseLong(editTextGiaSP.getText().toString()));
```

```
        produtcDAO.addProdut(product);
```

```
        // Cập nhật danh sách sản phẩm và thông báo cho adapter biết dữ liệu
```

```
        productArrayList.clear();
```

```
        productArrayList.addAll(produtcDAO.getListProduct());
```

```
        customAdapter.notifyDataSetChanged();
```

```
    }
```

```
});
```

```
// Chức năng nút Sửa
```

```
buttonSua.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
```

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

```
        // Cập nhật thông tin sản phẩm trong cơ sở dữ liệu
```

```
        Product product = new Product();
```

```
        product.setMaSP(Integer.parseInt(editTextMaSP.getText().toString()));
```

```
        product.setTenSP(editTextTenSP.getText().toString());
```

```
        product.setSoSP(Integer.parseInt(editTextSoSP.getText().toString()));
```

```
        product.setGiaSP(Long.parseLong(editTextGiaSP.getText().toString()));
```

```
        produtcDAO.updateProduct(product);
```

```
        // Cập nhật danh sách sản phẩm và thông báo cho adapter biết dữ liệu
```

```
        productArrayList.clear();
```

```
        productArrayList.addAll(produtcDAO.getListProduct());
```

```
        customAdapter.notifyDataSetChanged();
```

```
    }
```

```
});
```

```
// Chức năng nút Xóa
```

```
buttonXoa.setOnClickListener(new View.OnClickListener() {
```

đã thay đổi

đã thay đổi

```
@Override
public void onClick(View v) {
    // Xóa sản phẩm khỏi cơ sở dữ liệu
    int maSP = Integer.parseInt(editTextMaSP.getText().toString());
    produtcDAO.deleteProduct(maSP);

    // Cập nhật danh sách sản phẩm và thông báo cho adapter biết dữ liệu
    đã thay đổi
    productArrayList.clear();
    productArrayList.addAll(produtcDAO.getListProduct());
    customAdapter.notifyDataSetChanged();
}
});
}
}
```

Chạy thử ứng dụng để kiểm tra kết quả.

Lưu ý: Việc bố trí các class và layout như trên chỉ là ví dụ, trong thực tế các bạn có thể sử dụng mô hình phổ biến khi xây dựng ứng dụng Android là MVC hoặc MVVM để việc bố trí được tường minh hơn.

### **Ví dụ cấu trúc dự án theo mô hình MVC:**

- controller
  - ProductController.java
- dao
  - ProductDAO.java
- database
  - ProductDBHelper.java
- model
  - Product.java
- view
  - MainActivity.java
  - adapter
    - ProductAdapter.java
- res
  - layout
    - activity\_main.xml
    - item\_product.xml
    - layout\_add\_product.xml
    - layout\_edit\_product.xml

Trong cấu trúc này:

- controller: Chứa lớp điều khiển, ví dụ: ProductController. Lớp này sẽ là trung gian giữa giao diện người dùng và dữ liệu, xử lý các sự kiện người dùng và gọi các phương thức từ DAO để thao tác với dữ liệu.
- dao: Chứa lớp DAO để tương tác với cơ sở dữ liệu.
- database: Chứa lớp hỗ trợ cơ sở dữ liệu, như ProductDBHelper.
- model: Chứa lớp mô hình, ví dụ: Product.
- view: Chứa các thành phần giao diện người dùng, bao gồm MainActivity và các lớp adapter.
- adapter: Chứa ProductAdapter để quản lý RecyclerView.
- res/layout: Chứa các tệp layout XML cho giao diện người dùng.

Trong ProductController, bạn có thể thực hiện các phương thức như thêm, sửa, xóa sản phẩm, cũng như tải dữ liệu từ cơ sở dữ liệu và cung cấp cho MainActivity để hiển thị trên giao diện người dùng. Trong

MainActivity, bạn sẽ quản lý việc hiển thị dữ liệu và gắn kết với ProductController để xử lý các sự kiện người dùng và cập nhật giao diện người dùng.

### **Ví dụ cấu trúc dự án theo mô hình MVVM:**

- dao
  - ProductDAO.java
- database
  - ProductDBHelper.java
- model
  - Product.java
- ui
  - MainActivity.java
- adapter
  - ProductAdapter.java
- viewmodel
  - ProductViewModel.java
- res
  - layout
    - activity\_main.xml
    - item\_product.xml
    - layout\_add\_product.xml
    - layout\_edit\_product.xml

Trong cấu trúc này:

- dao: Chứa lớp DAO để tương tác với cơ sở dữ liệu.
- database: Chứa lớp hỗ trợ cơ sở dữ liệu, như ProductDBHelper.
- model: Chứa lớp mô hình, ví dụ: Product.
- ui: Chứa các thành phần giao diện người dùng, bao gồm MainActivity và các lớp adapter.
- adapter: Chứa ProductAdapter để quản lý RecyclerView.
- viewmodel: Chứa ProductViewModel để quản lý dữ liệu và logic liên quan đến giao diện người dùng.
- res/layout: Chứa các tệp layout XML cho giao diện người dùng.

Trong ProductViewModel, bạn có thể thực hiện các phương thức như thêm, sửa, xóa sản phẩm, cũng như tải dữ liệu từ cơ sở dữ liệu và cung cấp cho MainActivity thông qua LiveData để hiển thị trên giao diện người dùng. Trong MainActivity, bạn sẽ quản lý việc hiển thị dữ liệu và gắn kết

với ProductViewModel để xử lý các sự kiện người dùng và cập nhật giao diện người dùng.

Trương Mạnh Đạt