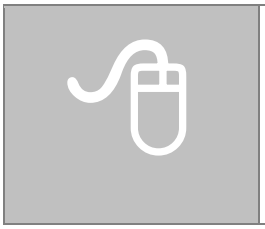
BÀI 1. LƯU TRỮ, TRUY VẤN VÀ SẮP XẾP DỮ LIỆU VỚI SQLITE



✓ **Mục tiêu:**

Biết cách tạo cơ sở dữ liệu SQLite.
Biết cách lưu trữ, truy vấn và sắp xếp dữ liệu với SQLite.

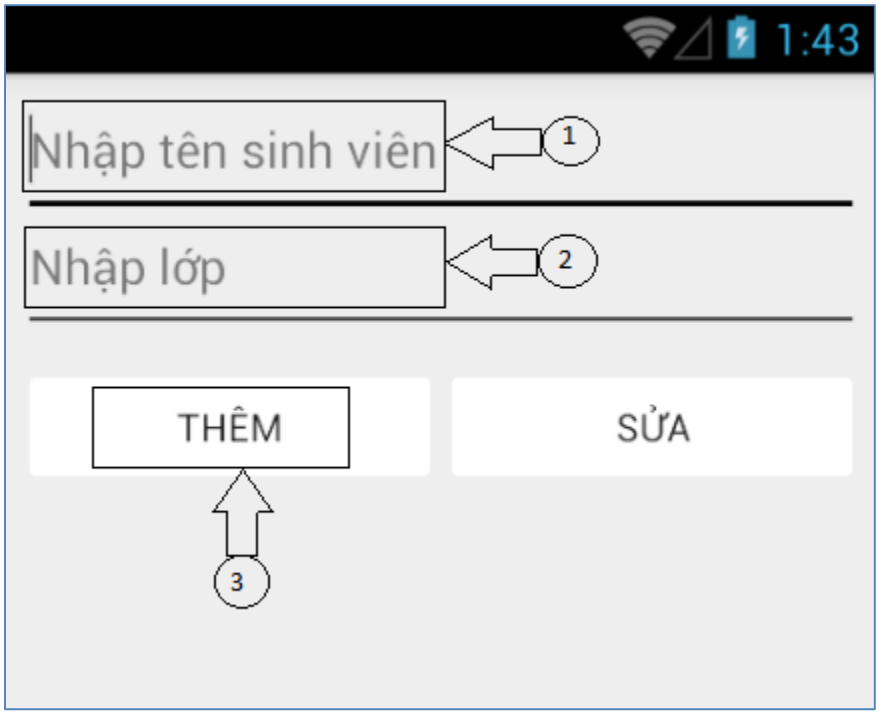
Bài tập 1.1. Tạo một cơ sở dữ liệu SQLite đơn giản

Đề bài:

Tạo một cơ sở dữ liệu SinhVien gồm: Mã sinh viên, Tên sinh viên, Lớp.

- Tạo cơ sở dữ liệu chứa bảng SinhVien.
- Thực hiện thêm, xóa, sửa.
- Lấy tất cả dữ liệu, lấy dữ liệu theo mã sinh viên, sắp xếp theo tên, theo lớp và xây dựng chức năng tìm kiếm sinh viên.

Với giao diện màn hình như sau:



- Sau khi nhấn thêm (3) thì thêm vào ListView (4) như sau:

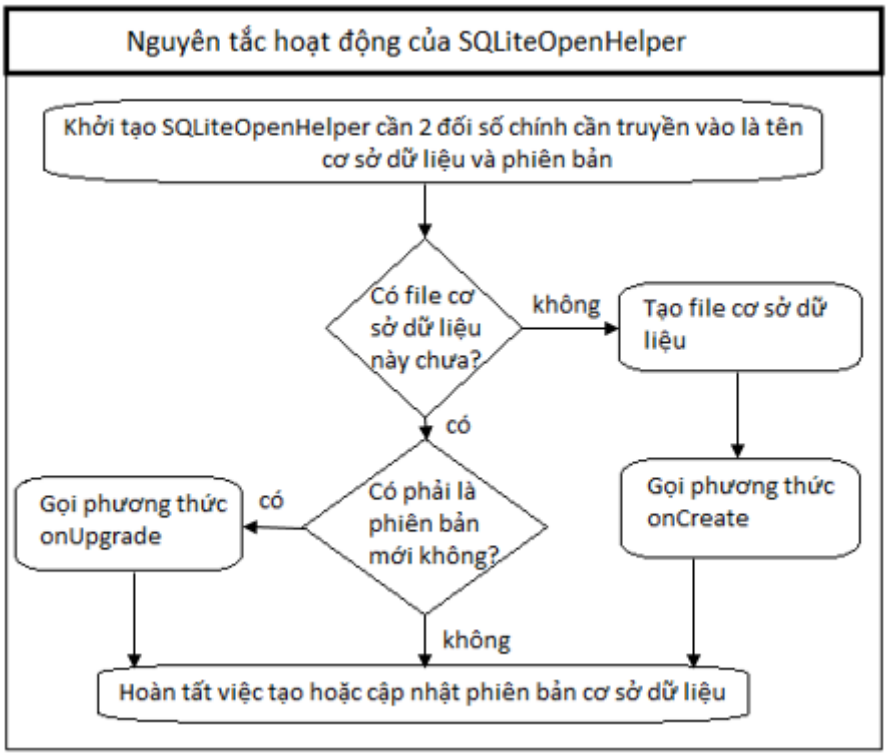
- Khi nhấn vào item (5) thì hiển thị tên và lớp của item này lên 2 EditText tương ứng. Sau đó, bạn có thể chỉnh sửa lại thông tin trên 2 EditText rồi ấn nút sửa (6) để cập nhật lại dữ liệu và hiển thị lên ListView (4).
- Khi nhấn vào ImageView (7) dòng nào thì xóa dòng đó trên ListView.

Chú ý: Mọi thay đổi trên ListView (4) thì dữ liệu trên SQLite đều thay đổi theo tương ứng.

Gợi ý thực hiện:

- Tạo lớp kế thừa lớp "SQLiteOpenHelper" để tạo cơ sở dữ liệu, bảng và viết các phương thức truy vấn (các bạn có thể tách các lớp ra để dễ nhìn và nếu có thêm nhiều bảng sẽ không bị rối).

  Mô tả nguyên tắc hoạt động của SQLiteOpenHelper:



- Xin quyền ghi file.

Hướng dẫn chi tiết:

Bước 1: Tạo cở sở dữ liệu và bảng trên SQLite.

Bước 1. 1: Tạo lớp "DBHelper" kế thừa "SQLiteOpenHelper" sau đó Override lại các phương thức của lớp này. "SQLiteOpenHelper" là lớp dùng để tạo cơ sở dữ liệu và các bản có trong cơ sở dữ liệu.

Bước 1. 2: Khai báo các biến hằng (tên bảng, cột,...) dùng để tiện cho việc sử dụng khi truy vấn, thêm, xóa, sửa,...

```
// Tên cơ sở dữ liệu
private static final String TEN_DATABASE= "QuanLySinhVien";
// Tên bảng
public static final String TEN_BANG_SINHVIEN = "SinhVien";
// Bảng gồm 3 cột _id, _ten và _lop.
public static final String COT_ID = "_id";
public static final String COT_TEN = "_ten";
public static final String COT_LOP = "_lop";
```

Bước 1. 3: Viết câu lệnh tạo bảng.

```
/*
* Câu lện tạo bảng.
```

```
* Trong đó: “integer primary key autoincrement”
*  là khóa chính tự tăng và có kiểu dữ liệu là int;
*  “text not null” là không được để trống và kiểu
*  dữ liệu là String.
*/
private static final String TAO_BANG_SINHVIEN = ""
      + "create table " + TEN_BANG_SINHVIEN + " ( "
      + COT_ID + " integer primary key autoincrement ,"
      + COT_TEN + " text not null, "
      + COT_LOP + " text not null );";
```

Bước 1. 4: Để tạo cơ sở dữ liệu phải thông qua hàm khởi tạo 4 tham số của lớp “SQLiteOpenHelper” mà chúng ta đã kế thừa. Chúng ta viết trong hàm khởi tạo của lớp “DBHelper” như sau:

```
public DBHelper(Context context) {
   super(context, TEN_DATABASE, null, 1);
}
```

Giải thích: các đối số tương ứng như trong hàm khởi tạo:

- context: dùng để mở hoặc tạo cơ sở dữ liệu
- *TEN_DATABASE*: tên file cơ sở dữ liệu.
- **null**: dùng để tạo đối tượng Cursor, hoặc null thì mặc định.
- 1: phiên bản của cơ sở dữ liệu, nếu là cơ sở dữ liệu cũ thì nó sẽ vào phương thức “onUpgrade” ngược lại thì vào “onDowngrade”.

Bước 1. 5: Dùng cơ sở dữ liệu vừa tạo xong truy xuất đến phương thức “execSQL” để tạo bảng cho cơ sở dữ liệu.

```
/*
 * Tạo bảng cho cơ sở dữ liệu đã được tạo ở
 *  hàm khởi tạo thông qua lớp SQLiteDatabase
 */
@Override
public void onCreate(SQLiteDatabase db) {
   db.execSQL(TAO_BANG_SINHVIEN);
}
```

Bước 2: Tạo đối tượng SinhVien, gồm các thuộc tính như sau:

```
private long _id;
private String _ten;
private String _lop;
```

Bước 3: Viết các phương thức truy vấn, thêm, xóa, sửa cơ sở dữ liệu.

Bước 3. 1: Tạo lớp "MyDatabase" dùng để hỗ trợ các phương thức tương tác với cơ sở dữ liệu, giảm bớt các đối tượng không cần thiết khi các lớp giao diện muốn truy xuất lấy dữ liệu.

Bước 3. 2: Khai báo 2 lớp SQLiteDatabase và DBHelper:

```
/*
 *  Lớp "SQLiteDatabase" dùng để hỗ trợ tương
 *   tác với cơ sở dữ liệu.
 */
SQLiteDatabase database;

/*
 *  Còn lớp "DBHelper" là lớp chúng
 *   ta vừa tạo, dùng để tạo Cơ sở dữ liệu và bảng.
 */
DBHelper helper;
```

Bước 3. 3: Viết hàm khởi tạo cho lớp "MyDatabase":

```
public MyDatabase(Context context) {
   helper = new DBHelper(context);

   /*
    *  Phương thức "getWritableDatabase()" dùng
    *   để tạo hoặc mở cơ sở dữ liệu để đọc và
    *   ghi vào cơ sở dữ liệu.
    */
   database = helper.getWritableDatabase();
}
```

Bước 3. 4: Thực hiện lấy tất cả dữ liệu trong cơ sở dữ liệu:

```
public Cursor layTatCaDuLieu() {
   // Biến cot là khai báo danh sách các cột cần lấy.
   String[] cot = { DBHelper.COT_ID,
         DBHelper.COT_TEN,
         DBHelper.COT_LOP };

   /*
    *  Cursor như là 1 bảng cơ sở dữ liệu được trả ra
    *   sau khi truy vấn trong cơ sở dữ liệu.
    */
   Cursor cursor = null;
```

```
    /*
     * Dùng lớp "SQLiteDatabase" truy xuất đến phương
     *  thức "query" để lấy dữ liệu trong cơ sở dữ liệu ra.
     *  Ở trong phương thức này là yêu cầu lấy tất cả nên
     *  chỉ cần truyền vào các tham số như: tên bảng, các
     *  cột cần lấy (cot) và sắp xếp nếu cần.
     */
    cursor = database.query(DBHelper.
          TEN_BANG_SINHVIEN, cot, null, null, null, null,
          DBHelper.COT_ID + " DESC");

    return cursor;
}
```

Bước 3. 5: Thực hiện thêm vào cơ sở dữ liệu:

```
public long them(SinhVien sinhVien) {
    /*
     * ContentValues là đối tượng lưu trữ dữ liệu, và
     *  SQLiteDatabase sẽ nhận dữ liệu thông qua đối tượng
     *  này để thực hiện các câu lệnh truy vấn.
     */
    ContentValues values = new ContentValues();
    values.put(DBHelper.COT_TEN,
          sinhVien.get_ten());
    values.put(DBHelper.COT_LOP,
          sinhVien.get_lop());

    /*
     * Thêm vào cơ sở dữ liệu cần 2 đối số chính là
     *  Tên Bảng và dữ liệu cần thêm.
     */
    return database.insert(DBHelper.
             TEN_BANG_SINHVIEN, null, values);
}
```

Bước 3. 6: Thực hiện xóa một dòng dữ liệu:

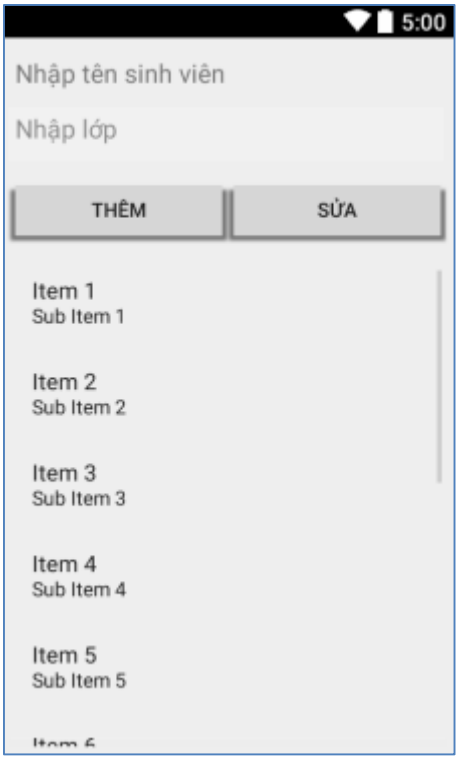
```
public long xoa(SinhVien sinhVien) {
    /*
     * Tương tự cho xóa một dòng dữ liệu cần
     *  3 đối số là tên bảng, câu điều kiện và
     *  cuối cùng là đối số cho câu điều kiện
     *  nhưng có thể gộp 2 đối số làm 1 như ở
     *  dưới.
     */
    return database.delete(DBHelper
```

```
            .TEN_BANG_SINHVIEN, DBHelper
            .COT_TEN + " = " + "'" +
               sinhVien.get_ten() + "'", null);
    }
```

Bước 3. 7: Thực hiện sửa một dòng dữ liệu:

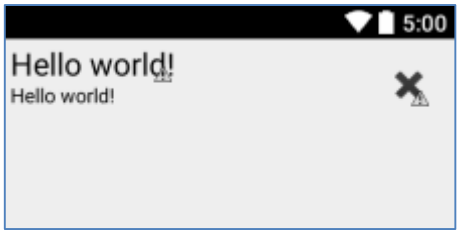
```
public long sua(SinhVien sinhVien) {
   ContentValues values = new ContentValues();
   values.put(DBHelper.COT_TEN,
         sinhVien.get_ten());
   values.put(DBHelper.COT_LOP,
         sinhVien.get_lop());
   /*
    * Sửa 1 dòng dữ liệu thì cần 3 đối số chính là
    *  tên bảng, dữ liệu cần sửa và câu điều kiện
    */
   return database.update(DBHelper
         .TEN_BANG_SINHVIEN, values,
         DBHelper.COT_ID + " = "
         + sinhVien.get_id(), null);
}
```

Bước 4: Thiết kế giao diện "activity_main.xml" để tương tác với cơ sở dữ liệu:



Hình 1.1

Bước 5: Thiết kế item "design_item_list_view.xml" cho ListView như sau:



Hình 1.2

Bước 6: Viết xử lý trong class "MainActivity.java".

Bước 6. 1: Khai báo một số lớp:

```
/*
 * listView dùng để hiển thị cơ sở dữ liệu
 *  cho người dùng xem.
 */
public static ListView listView;

// dữ liệu để hiển thị lên listView
public static ArrayList<SinhVien> sinhViens;

// Lớp để hỗ trợ truy vấn dữ liệu
public static MyDatabase database;

// Ô nhập liệu
private EditText editTextTen, editTextLop;
```

Bước 6. 2: Khai báo một số biến dùng để lưu trữ giá trị nhập và tương tác từ người dùng:

```
private String ten, lop;
private static long id = -1;
```

Bước 6. 3: Viết phương thức lấy dữ liệu gán vào ArrayList:

```
public void capNhatDuLieu() {
   if (sinhViens == null) {
      sinhViens = new ArrayList<SinhVien>();
   } else {
      sinhViens.removeAll(sinhViens);
   }

   // Lấy dữ liệu, dùng Cursor nhận lại
   Cursor cursor = database.layTatCaDuLieu();
```

```
    if (cursor != null) {
       /*
        * Di chuyển đến từng dòng dữ liệu
        *  thông qua phương thức moveToNext
        */
       while (cursor.moveToNext()) {
          SinhVien sinhVien = new SinhVien();

          /*
           * Mỗi dòng dữ liệu chúng ra sẽ lấy
           *  theo cột và gán vào đối tượng
           *  SinhVien
           */
          sinhVien.set_id(Integer.parseInt
                (cursor.getString(cursor
                .getColumnIndex
                (DBHelper.COT_ID))));
          sinhVien.set_ten(cursor.getString
                (cursor
                .getColumnIndex
                (DBHelper.COT_TEN)));
          sinhVien.set_lop(cursor.getString
                (cursor
                .getColumnIndex
                (DBHelper.COT_LOP)));

          // thêm vào danh sách SinhVien
          sinhViens.add(sinhVien);
       }
    }
}
```

Bước 6. 4: Thiết lập cho ListView:

```
// Hiển thị danh sách đã lấy được lên listView
    if (sinhViens != null) {
       listView.setAdapter
       (new MyAdapter(getApplicationContext()));
    }
    // Bắt sự kiện khi click lên ListView
    listView.setOnItemClickListener(new
          OnItemClickListener() {

       @Override
       public void onItemClick(AdapterView<?> parent,
             View view, int position, long id) {
          editTextTen.setText
```

```
            (sinhViens.get(position).get_ten());
         editTextLop.setText
         (sinhViens.get(position).get_lop());
         MainActivity.id = id;
      }
   });
```

Bước 6. 5: Viết phương thức lấy dữ liệu khi người dùng nhập:

```
/*
 * Lấy dữ liệu từ người dùng nhập
 *  gán vào đối tượng SinhVien
 */
public SinhVien layDuLieuNguoiDung() {
   ten = editTextTen.getText().toString();
   lop = editTextLop.getText().toString();
   if (ten.trim().length() == 0
         || lop.trim().length() == 0)
      return null;
   SinhVien sinhVien = new SinhVien();
   sinhVien.set_id(id);
   sinhVien.set_ten(ten);
   sinhVien.set_lop(lop);
   return sinhVien;
}
```

Bước 6. 6: Bắt sự kiện và xử lý khi người dùng nhấn yêu cầu thêm dữ liệu:

```
/*
 * Thêm dữ liệu vào cơ sở dữ liệu
 */
public void them(View view) {
   SinhVien sinhVien1 = layDuLieuNguoiDung();
   if (sinhVien1 != null) {
      if (database.them(sinhVien1) != -1) {
         sinhViens.add(sinhVien1);
         capNhatDuLieu();

         // Cập nhật lại danh sách
         listView.invalidateViews();

         editTextTen.setText(null);
         editTextLop.setText(null);
         id = -1;
      }
   }
```

```
}
```

Bước 6. 7: Bắt sự kiện và xử lý khi người dùng yêu cầu sửa cơ sở dữ liệu:

```
/*
 * Cập nhật cơ sở dữ liệu
 */
public void sua(View view) {

   SinhVien sinhVien1 = layDuLieuNguoiDung();

   if (sinhVien1 != null && id != -1) {
      database.sua(sinhVien1);
      capNhatDuLieu();

      // Cập nhật lại danh sách
      listView.invalidateViews();

      editTextTen.setText(null);
      editTextLop.setText(null);
      id = -1;
   }
}
```

Bước 7: Viết adapter cho ListView. Đầu tiên kết thừa lại lớp "BaseAdapter" và Override lại các phương thức của lớp này và viết xử lý như sau:

```
public class MyAdapter extends BaseAdapter {

  /*
   *  LayoutInflater là lớp hỗ trợ lấy tài nguyên layout
   *   trong thư mục res
   */
  LayoutInflater inflater;
  TextView textView;
  Context context;
  public MyAdapter(Context context) {
     /*
      *  Khởi tạo lớp LayoutInflater
      *   thông qua Context
      */
     inflater = LayoutInflater.from(context);
     this.context = context;
  }
  @Override
```

```
public int getCount() {
   // Tổng số lượng dòng dữ liệu
   return MainActivity.sinhViens.size();
}
@Override
public Object getItem(int position) {
   /*
    * Lấy đối tượng của 1 dòng dữ liệu
    *  theo vị trí (position)
    */
   return MainActivity.sinhViens.get(position);
}
@Override
public long getItemId(int position) {
   /*
    * Lấy id của 1 dòng dữ liệu
    *  theo vị trí (position)
    */
   return MainActivity.sinhViens.get(position).get_id();
}

@Override
public View getView(final int position, View convertView,
      ViewGroup parent) {
   // Lấy layout từ thư mục res
   View view = inflater.inflate(R.layout
         .design_item_list_view, null);

   // Thiết lập thông tin hiển thị
   textView = (TextView) view.findViewById(R.id.tv_ten);
   textView.setText(MainActivity.sinhViens
         .get(position).get_ten());
   textView = (TextView) view.findViewById(R.id.tv_time);
   textView.setText(MainActivity.sinhViens
         .get(position).get_lop());

   // Bắt sự kiện click lên imageView.
   ((ImageView) view.findViewById(R.id.imageView))
         .setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
               MainActivity.database.xoa
               (MainActivity.sinhViens.get(position));
               MainActivity.sinhViens.remove(position);
               // MainActivity.listView.invalidateViews();
               notifyDataSetChanged();
            }
         });
   return view;
```

```
    }
}
```

Bài tập 1.2. Tạo cơ sở dữ liệu SQLite phức tạp hơn

Đề bài:

- Tạo một cơ sở dữ liệu SQLite gồm:
    - Bảng Nhân Viên: Mã NV, Họ Tên, Ngày Sinh, Giới Tính, Địa Chỉ, Điện Thoại, Ghi Chú.
    - Bảng Khách Hàng: Mã KH, Họ Tên, Ngày Sinh, Giới Tính, Email, Điện Thoại. Bảng Hóa Đơn: Mã HD, Ngày Bán, Mã NV, Mã KH, Tổng Tiền.
    - Bảng Chi Tiết Hóa Đơn: Mã HD, Mã Sản Phẩm, Giá Bán, Số Lượng, Tổng Tiền.
    - Bảng Sản Phẩm: Mã SP, Tên SP, Ngày SX, Giá Bán, Số Lượng.
- Tạo cơ sở dữ liệu SQLite có một bảng Nhân Viên. Thực hiện thêm, xóa, sửa, sắp xếp, lấy tất cả dữ liệu, lấy theo mã, lấy theo tên.
- Thêm 3 bảng còn lại vào cơ sở dữ liệu vừa tạo. Ràng buộc khóa chính, khóa ngoại. Xây dựng chức năng: tìm kiếm (Nhân Viên, Sản Phẩm, Khách Hàng), thực hiện mua/ bán hàng, thống kê số sản phẩm bán được, doanh thu theo thời gian nhập (ngày bắt đầu và ngày kết thúc), thống kê sản phẩm bán nhiều nhất trong tháng.

Giao diện: Tự thiết kế.

Gợi ý thực hiện:

- Tạo mô hình 3 – Layer để dễ quản lý.
- Tạo liên kết khóa ngoại như sau: “FOREIGN KEY (ten_cot_cua_bang_can_lien_ket) REFERENCES ten_bang_lien_ket (khoa_chinh) ”.