Putting SQL to Work:

Introduction to SQLite in Android:

- **SQLite** is a lightweight, built-in database used to store data locally on an Android device. It does not require a server to run — the database is stored directly on the device as a simple file on the disk. SQLite is ideal for small to medium-sized applications that need to store structured data efficiently.
- In Android, SQLite is essential for managing data without relying on external servers. It allows apps to work offline, save user information, settings, and other structured data seamlessly. Being lightweight, fast, and reliable, SQLite ensures smooth app performance and effective local data handling.

Features:

- **Lightweight:** SQLite is very small in size and requires minimal memory, making it perfect for mobile devices.
- **Server less & Self-contained:** No need for a server — the database is stored as a single file on the device. All database logic (engine, transactions, tables, etc.) is contained within a single disk file.
- **Zero Configurations:** No installation, no setup, and no administration required.
- **Reliable and Robust:** Supports ACID properties (Atomicity, Consistency, Isolation, Durability) ensuring safe and secure transactions.
- **Simple API:** Easy-to-use APIs provided by Android for CRUD operations (Create, Read, Update, Delete).
- **Cross-platform:** Works across different operating systems and platforms.
- **Data Storage:** Perfect for storing structured data like user profiles, settings, history, and more.
- **Readable and Editable:** SQLite database files are readable and can be edited manually with tools if needed.
- **Supports SQL Standards:** You can use common SQL commands like SELECT, INSERT, UPDATE, DELETE, etc.

Common applications of SQLite database:

- **Mobile Applications**: Used in Android and iOS apps for local data storage (e.g., user settings, app data).
- **Web Browsers**: Browsers like **Google Chrome** and **Mozilla Firefox** use SQLite to store bookmarks, cookies, history, and session data.
- **Embedded Systems**: Used in devices like smart TVs, IoT gadgets, and GPS devices where a lightweight database is needed.
- **Desktop Applications**: Apps like **Skype** and **Drop box** use SQLite to manage user data locally.

- **Testing and Prototyping**: Used by developers to quickly test and prototype applications without needing a full database server.
- **Data Analysis Tools**: Lightweight databases for managing datasets locally in research or analytics projects.

In and OUT of SQLite:

In of SQLite (Inserting Data into SQLite)

- **Inserting Data:**
  You insert (write) data into an SQLite database in Android using the SQLiteDatabase class. The most common way to insert data is by using ContentValues, which is a key-value store for inserting data into tables. Use insert() method with ContentValues.

IN: Inserts "Hello from Database!" into a table.

1. **Create ContentValues :** Create a map (key-value pairs) where keys = column names, Values = data you want to insert.
2. **Open Database :** Open the database using `getWritableDatabase()`.
3. **Insert into Table :** Use `db.insert()` method to insert data into the table.
4. **Close Database**: After inserting, you can close the database to free resources.

```
// 1. Open database
SQLiteDatabase db = dbHelper.getWritableDatabase();

// 2. Prepare data
        ContentValues values = new ContentValues();
        values.put("message", "Hello from Database!");

// 3. Insert into table

      db.insert("messages", null, values);

// 4. (Optional) Close database
        db.close();
```

This **inserts** the "Hello" message into the **messages** table.

Out of SQLite (Reading Data from SQLite)

- **Reading Data:**
  You retrieve (read) data from an SQLite database using the SQLiteDatabase class and Cursor objects. The Cursor object allows you to iterate over the result set returned by a query. Use query() method and read using Cursor.

OUT: Reads "Hello from Database!" and displays it.

1.**Open Database:** Open the database using `getReadableDatabase()`.
2. **Query the Table:** Use `db.query()` to fetch data.
3. **Use Cursor:** Cursor is used to move through the rows of results.
4. **Read Data from Cursor:** Get each column's value using cursor methods like `getString()`, `getInt()`.
5. **Close Cursor and Database:** After reading, close cursor and (optionally) the database.

```
public String getMessage() {
      SQLiteDatabase db = this.getReadableDatabase();
Cursor cursor = db.query("messages", new String[]{"message"}, null, null, null, null, null);
if (cursor.moveToFirst()) {
   String message = cursor.getString(cursor.getColumnIndex("message"));
}
cursor.close();
```

This **reads** the "Hello" message from the **messages** table.

Hello database:

We create a **simple database** that:

1. Create a database.
2. Create a table (messages).
3. Insert "Hello" message into the table.
4. Read the "Hello" message from the table.

```
public class HelloDatabaseHelper extends SQLiteOpenHelper {
   // Database name and version
   private static final String DATABASE_NAME = "hello.db";
   private static final int DATABASE_VERSION = 1;
   // Constructor
   public HelloDatabaseHelper(Context context) {
      super(context, DATABASE_NAME, null, DATABASE_VERSION);
   }
   // Create table when database is created
   @Override
   public void onCreate(SQLiteDatabase db) {
      db.execSQL("CREATE TABLE messages (id INTEGER PRIMARY KEY AUTOINCREMENT, message
TEXT);");

      // Insert default "Hello" message
      ContentValues values = new ContentValues();
```

```
    values.put("message", "Hello from Database!");
    db.insert("messages", null, values);
  }
  // Read message from the database
   public String getMessage() {
      SQLiteDatabase db = this.getReadableDatabase();
Cursor cursor = db.query("messages", new String[]{"message"}, null, null, null, null, null);
if (cursor.moveToFirst()) {
   String message = cursor.getString(cursor.getColumnIndex("message"));
}
cursor.close();
    db.close();
    return message;
  }
```

Data Binding:

Data Binding is a technique that lets you:

- Directly bind (connect) UI components (like TextView, EditText) to data sources (like database values, variables).
- No need for a lot of `findViewById()`.
- Updates happen automatically when data changes.

Main Code for Data Binding:

1.Enable the Data Binding in gradle:

build.gradle (Enable Data Binding)

```
android {
   buildFeatures {
      dataBinding true
   }
}
```

2. Define the layout file:

XML Layout (example: `activity_main.xml`)

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">
```

```
    <data>
        <variable
            name="message"
            type="String" />
    </data>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{message}" />
    </LinearLayout>

</layout>
```

Step 3: MainActivity.java

```
ActivityMainBinding  binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
binding.setMessage("Hello from Database!");
```