W3GRADS

SQLite in Android

What is SQLite in Android?

SQLite is a **lightweight, built-in database** engine in Android.
It allows your app to **store structured data** locally — meaning **no internet connection** is required.

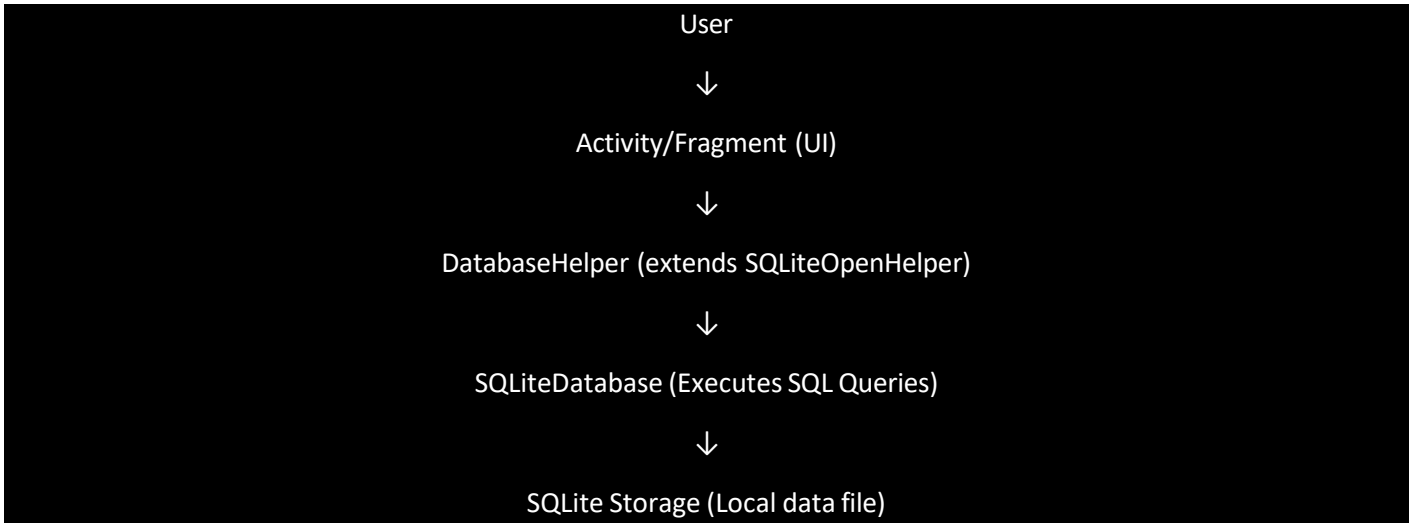
So, if We want to **store, read, update, or delete** data on the user's device (like contacts, notes, or app settings), **SQLite** is the perfect solution.

How SQLite Works in Android

When We use SQLite in Android, there are typically **three key layers** working together:

1. **Database Helper (SQLiteOpenHelper)** → creates, upgrades, and manages the database
2. **Database Operations (SQLiteDatabase, ContentValues, Cursor)** → actually read/write data
3. **App Layer (Activity/Fragment)** → interacts with the user and calls the helper methods

So your app's flow looks like this:



CRUD Operations

- CRUD = Create, Read, Update, Delete

Operation	SQL Command (short notes)
Create Table	CREATE TABLE
Insert Data	INSERT
Read Data	SELECT

Edit Data	UPDATE
Remove Data	DELETE

Needed Classes to Use Sqlite in android

CLASS 1: SQLiteOpenHelper — the Heart of the Database System

What it is:

SQLiteOpenHelper is an **abstract helper class** provided by Android to manage database creation, connection, and version control.

Why we use it:

- It automatically handles **database creation** and **upgrades**.
- It ensures your database is opened efficiently.
- It prevents repetitive boilerplate code.

How it works:

We **extend this class** and implement two required methods:

1. onCreate(SQLiteDatabase db) → called the first time database is created
2. onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) → called when database schema changes (like adding a new column)

Example Code:

```
public class DatabaseHelper extends SQLiteOpenHelper {
    public DatabaseHelper(Context context) {
        super(context, "StudentDB", null, 1);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        // Called once when DB is created
        db.execSQL("CREATE TABLE students (id INTEGER PRIMARY KEY
AUTOINCREMENT, name TEXT, age INTEGER)");
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Called when DB version changes
        db.execSQL("DROP TABLE IF EXISTS students");
        onCreate(db);
    }
}
```

CLASS 2: SQLiteDatabase — the Actual Database Engine

What it is:

SQLiteDatabase represents your **actual database** where all SQL operations happen.

Why we use it:

It allows Us to:

- Execute **SQL commands**
- Perform **insert, update, delete, select**
- Get readable and writable access to data

How it works:

SQLiteOpenHelper gives us two versions of this object:

- getReadableDatabase() → use for reading data
- getWritableDatabase() → use for inserting, updating, or deleting data.

Example Code:

```
SQLiteDatabase db = this.getWritableDatabase(); // opens DB for writing

db.execSQL("INSERT INTO students(name, age) VALUES('John', 25)");
```

CLASS 3: ContentValues — the Data Container

< **What it is:**

ContentValues is a **key-value pair** class used to safely insert or update data in the database.

Instead of writing raw SQL strings, we use ContentValues to map **column names → values**.

Why we use it:

- Prevents SQL injection
- Easier to maintain
- More readable

Example Code:

```
ContentValues values = new ContentValues();
values.put("name", "Emma");
values.put("age", 22);
db.insert("students", null, values);
```

CLASS 4: Cursor — the Data Reader

What it is:

A **Cursor** is like a pointer or iterator for your query results.
When we run a SELECT query, the database returns a Cursor that holds all matching rows.

Why we use it:

- To **read data row by row**
- To access columns by their names or index
- To loop through query results efficiently

How it Works:

```
Cursor cursor = db.rawQuery("SELECT * FROM students", null);
if (cursor.moveToFirst()) {
    do {
        String name = cursor.getString(cursor.getColumnIndexOrThrow("name"));
        int age = cursor.getInt(cursor.getColumnIndexOrThrow("age"));
        Log.d("DATA", "Name: " + name + " | Age: " + age);
    } while (cursor.moveToNext());
}
```

```
cursor.close();
```

How SQLite Works in Android Program

Android provides a helper class called **SQLiteOpenHelper** to manage databases easily.

This class:

1. Creates the database the first time it's needed.
2. Handles **version upgrades** (if we change the schema).
3. Gives us readable and writable database access.

Why Use SQLite?

Here's why developers love using SQLite in Android:

1. 🟩 **Offline storage** — works without internet.
2. 🟩 **Fast and reliable** — uses SQL queries.
3. 🟩 **Integrated with Android** — no setup needed.
4. 🟩 **Lightweight** — no server or installation required.
5. 🟩 **Data persistence** — data remains even after the app closes

Structure of SQLite Database

An SQLite database is organized like a normal SQL database:

- **Database** → stores tables
- **Table** → stores rows (records) and columns (fields)
- **Row** → represents a single record
- **Column** → represents a data field (like name, age, email)

Advantages of SQLite in Android

No setup required — built into Android
Lightweight and fast
Supports standard SQL syntax
Works fully ofline
Easy to integrate with UI

Limitation of SQLite

Not ideal for large-scale or complex relational data
No direct cloud sync (must handle manually)
Multi-threading requires careful management
Manual query writing (boilerplate code)