SQLite Database in Android

SQLite is a lightweight, self-contained, serverless database engine that is built into Android. It is used for storing and managing structured data in mobile applications efficiently.

Features of SQLite in Android

1. **Embedded Database** – SQLite comes pre-installed with Android, so no extra setup is needed.
2. **Lightweight** – It has a small memory footprint and does not require a separate server.
3. **SQL Support** – Supports SQL syntax like `SELECT`, `INSERT`, `UPDATE`, `DELETE`, etc.
4. **Relational Database** – Allows defining tables, indexes, and relationships.
5. **Local Storage** – Works offline without requiring an internet connection.
6. **ACID-Compliant** – Ensures data integrity with transactions.

How to Use SQLite in Android

1. Creating a Database

You need to create a helper class that extends `SQLiteOpenHelper`.

```
class DBHelper(context: Context) : SQLiteOpenHelper(context,
"MyDatabase.db", null, 1) {

    override fun onCreate(db: SQLiteDatabase) {
        val createTableQuery = "CREATE TABLE users (id INTEGER
PRIMARY KEY, name TEXT, age INTEGER)"
        db.execSQL(createTableQuery)
    }

    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int,
newVersion: Int) {
        db.execSQL("DROP TABLE IF EXISTS users")
        onCreate(db)
    }
}
```

---

2. Inserting Data

```
fun insertUser(name: String, age: Int) {
    val db = this.writableDatabase
    val values = ContentValues()
    values.put("name", name)
    values.put("age", age)
    db.insert("users", null, values)
    db.close()
}
```

---

3. Retrieving Data

```
fun getUsers(): Cursor {
    val db = this.readableDatabase
    return db.rawQuery("SELECT * FROM users", null)
}
```

---

4. Updating Data

```
fun updateUser(id: Int, name: String, age: Int) {
    val db = this.writableDatabase
    val values = ContentValues()
    values.put("name", name)
    values.put("age", age)
    db.update("users", values, "id = ?", arrayOf(id.toString()))
    db.close()
}
```

---

5. Deleting Data

```
fun deleteUser(id: Int) {
    val db = this.writableDatabase
    db.delete("users", "id = ?", arrayOf(id.toString()))
    db.close()
```

```
}
```

Advantages of SQLite in Android

- No need for an external database server.
- Faster for small to medium-sized applications.
- Simple API for CRUD operations.
- Secure and reliable data storage.

Would you like a complete example of an Android app using SQLite with Kotlin?

ContentValues and Cursor in Android SQLite

1. ContentValues

`ContentValues` is a key-value storage class in Android, used to store data before inserting or updating records in an SQLite database.

Example Usage of ContentValues

- Used in `insert()` and `update()` methods of `SQLiteDatabase`.
- It helps structure data in a column-value format.

Example of Using ContentValues

kotlin

```
val values = ContentValues()
values.put("name", "Urmil Bhatt")  // Storing a name
values.put("age", 25)  // Storing an age

val db = writableDatabase
db.insert("users", null, values)  // Inserting into "users" table
db.close()
```

Why Use ContentValues?

- Provides an easy way to store column values.
- Prevents SQL injection.
- Makes the `insert()` and `update()` methods cleaner.

---

2. Cursor

`Cursor` is an interface used to read data from an SQLite database query result. It moves row by row, allowing access to each record.

Example Usage of Cursor

- Used to fetch results from `rawQuery()` or `query()` methods.
- Provides methods to move (`moveToFirst()`, `moveToNext()`, etc.) and retrieve values.

Example of Using Cursor
kotlin

```
val db = readableDatabase
val cursor: Cursor = db.rawQuery("SELECT * FROM users", null)

if (cursor.moveToFirst()) {
    do {
        val id = cursor.getInt(cursor.getColumnIndexOrThrow("id"))
        val name =
cursor.getString(cursor.getColumnIndexOrThrow("name"))
        val age = cursor.getInt(cursor.getColumnIndexOrThrow("age"))

        println("ID: $id, Name: $name, Age: $age")

    } while (cursor.moveToNext())
}
cursor.close()
db.close()
```

Why Use Cursor?

- Efficient way to traverse the result set row by row.
- Provides various methods to get values (`getInt()`, `getString()`, etc.).
- Prevents excessive memory usage by loading only required data.

Comparison: ContentValues vs Cursor

Feature	ContentValues	Cursor
Purpose	Stores data before insertion/update	Reads data from the database
Usage	Used in `insert()` and `update()`	Used in `query()` and `rawQuery()`
Data Type Storage	Key-value pair (column-value)	Row-based access
Example Methods	`put()`, `clear()`	`moveToFirst()`, `moveToNext()`, `getInt()`, `getString()`