Manejo de SQLite en Android Studio con Java

1. Introducción: ¿Qué es SQLite?

SQLite es un sistema de gestión de bases de datos relacional (RDBMS) contenido en una pequeña biblioteca C. A diferencia de otros sistemas de bases de datos, SQLite no necesita un proceso de servidor separado. Es **"serverless"**, **"self-contained"** y **"zero-configuration"**.

Es la base de datos más utilizada en el mundo, y es el estándar para el almacenamiento de datos locales en dispositivos Android.

---

2. ¿Por qué usar SQLite en Android?

- **Integración N nativa:** Android proporciona APIs directas para interactuar con SQLite.
- **Ligero:** No requiere mucha memoria o CPU, ideal para dispositivos móviles.
- **Almacenamiento Offline:** Permite que tu aplicación funcione sin conexión a internet.
- **Rendimiento:** Acceso rápido a los datos locales.
- **Persistencia de datos:** Los datos permanecen incluso si la aplicación se cierra o el dispositivo se reinicia.

---

3. Conceptos Clave de SQLite en Android

Para interactuar con SQLite en Android, usaremos principalmente estas clases:

- **SQLiteOpenHelper**: Una clase de ayuda para gestionar la creación y actualización de la base de datos y sus tablas.
- **SQLiteDatabase**: La clase principal para realizar operaciones CRUD (Crear, Leer, Actualizar, Borrar) en la base de datos.
- **ContentValues**: Un objeto para almacenar un conjunto de valores que se pueden insertar o actualizar en la base de datos.
- **Cursor**: Una interfaz que proporciona acceso de lectura y escritura a los resultados de una consulta de base de datos.

---

4. Paso a Paso: Implementando SQLite en Android Studio

Vamos a construir un ejemplo sencillo para manejar una lista de "Contactos".

Paso 1: Define el Esquema de la Base de Datos

Es una buena práctica definir las constantes para el nombre de la base de datos, las tablas y las columnas.

Java

```java
// Contrato de la base de datos (DBHelper.java)
public final class ContactosContract {
    private ContactosContract() {}
    public static class ContactoEntry implements BaseColumns {
        public static final String TABLE_NAME = "contactos";
        public static final String COLUMN_NAME_NOMBRE = "nombre";
        public static final String COLUMN_NAME_TELEFONO = "telefono";
        public static final String COLUMN_NAME_EMAIL = "email";
    }

    private static final String SQL_CREATE_ENTRIES =
            "CREATE TABLE " + ContactoEntry.TABLE_NAME + " (" +
                    ContactoEntry._ID + " INTEGER PRIMARY KEY," +
                    ContactoEntry.COLUMN_NAME_NOMBRE + " TEXT," +
                    ContactoEntry.COLUMN_NAME_TELEFONO + " TEXT," +
                    ContactoEntry.COLUMN_NAME_EMAIL + " TEXT)";

    private static final String SQL_DELETE_ENTRIES =
            "DROP TABLE IF EXISTS " + ContactoEntry.TABLE_NAME;
}
```

Paso 2: Crea tu SQLiteOpenHelper Personalizado

Esta clase se encargará de crear la base de datos por primera vez y de manejar las actualizaciones.

Java

```java
// ContactosDbHelper.java
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class ContactosDbHelper extends SQLiteOpenHelper {
    public static final int DATABASE_VERSION = 1;
    public static final String DATABASE_NAME = "Contactos.db";

    public ContactosDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(ContactosContract.SQL_CREATE_ENTRIES);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Esta política de actualización es simple: descarta los datos y recrea la base de datos
        db.execSQL(ContactosContract.SQL_DELETE_ENTRIES);
        onCreate(db);
    }

    @Override
```

```
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        onUpgrade(db, oldVersion, newVersion);
    }
}
```

Paso 3: Realiza Operaciones CRUD (Crear, Leer, Actualizar, Borrar)

Ahora, veremos cómo usar SQLiteDatabase para interactuar con la base de datos.

A. Insertar Datos

Java

```
// En tu Activity o en una clase de repositorio
ContactosDbHelper dbHelper = new ContactosDbHelper(getContext());
SQLiteDatabase db = dbHelper.getWritableDatabase(); // Para escribir datos

// Crea un nuevo mapa de valores, donde los nombres de las columnas son las claves
ContentValues values = new ContentValues();
values.put(ContactosContract.ContactoEntry.COLUMN_NAME_NOMBRE, "Juan Pérez");
values.put(ContactosContract.ContactoEntry.COLUMN_NAME_TELEFONO, "123-456-7890");
values.put(ContactosContract.ContactoEntry.COLUMN_NAME_EMAIL, "juan.perez@example.com");

// Inserta la nueva fila, devolviendo el valor de la clave primaria de la nueva fila.
long newRowId = db.insert(ContactosContract.ContactoEntry.TABLE_NAME, null, values);

// Cierra la conexión cuando hayas terminado (¡Importante!)
db.close();
```

B. Leer Datos

Java

```
// En tu Activity o en una clase de repositorio
ContactosDbHelper dbHelper = new ContactosDbHelper(getContext());
SQLiteDatabase db = dbHelper.getReadableDatabase(); // Para leer datos

// Define las columnas que quieres recuperar
String[] projection = {
    ContactosContract.ContactoEntry._ID,
    ContactosContract.ContactoEntry.COLUMN_NAME_NOMBRE,
    ContactosContract.ContactoEntry.COLUMN_NAME_TELEFONO
};

// Define la cláusula WHERE (opcional)
String selection = ContactosContract.ContactoEntry.COLUMN_NAME_NOMBRE + " = ?";
String[] selectionArgs = { "Juan Pérez" }; // Valores para la cláusula WHERE

// Cómo ordenar los resultados (opcional)
String sortOrder = ContactosContract.ContactoEntry.COLUMN_NAME_NOMBRE + " DESC";

Cursor cursor = db.query(
    ContactosContract.ContactoEntry.TABLE_NAME,   // La tabla a consultar
    projection,                                   // Las columnas a devolver
    selection,                                    // Las columnas para la cláusula WHERE
    selectionArgs,                                // Los valores para la cláusula WHERE
    null,                                         // No agrupar las filas
    null,                                         // No filtrar por grupos de filas
    sortOrder                                     // El orden de clasificación
);
```

```
List<String> itemNames = new ArrayList<>();
while(cursor.moveToNext()) {
    String nombre = cursor.getString(
        cursor.getColumnIndexOrThrow(ContactosContract.ContactoEntry.COLUMN_NAME_NOMBRE));
    String telefono = cursor.getString(
        cursor.getColumnIndexOrThrow(ContactosContract.ContactoEntry.COLUMN_NAME_TELEFONO));
    itemNames.add(nombre + " - " + telefono);
}
cursor.close(); // Cierra el cursor (¡Importante!)
db.close();
```

C. Actualizar Datos

Java

```
// En tu Activity o en una clase de repositorio
ContactosDbHelper dbHelper = new ContactosDbHelper(getContext());
SQLiteDatabase db = dbHelper.getWritableDatabase();

// Nuevos valores para una fila.
ContentValues values = new ContentValues();
values.put(ContactosContract.ContactoEntry.COLUMN_NAME_TELEFONO, "987-654-3210");

// ¿Cuál fila queremos actualizar?
String selection = ContactosContract.ContactoEntry.COLUMN_NAME_NOMBRE + " LIKE ?";
String[] selectionArgs = { "Juan Pérez" };

int count = db.update(
    ContactosContract.ContactoEntry.TABLE_NAME,
```

```
    values,
    selection,
    selectionArgs
);

db.close();
```

D. Borrar Datos

Java

```
// En tu Activity o en una clase de repositorio
ContactosDbHelper dbHelper = new ContactosDbHelper(getContext());
SQLiteDatabase db = dbHelper.getWritableDatabase();

// Define 'where' part of query.
String selection = ContactosContract.ContactoEntry.COLUMN_NAME_NOMBRE + " LIKE ?";
// Specify arguments in placeholder order.
String[] selectionArgs = { "Juan Pérez" };
// Issue SQL statement.
int deletedRows = db.delete(ContactosContract.ContactoEntry.TABLE_NAME, selection, selectionArgs);

db.close();
```

---

5. Buenas Prácticas y Consejos

- **Cierra tus recursos:** Siempre cierra SQLiteDatabase y Cursor cuando hayas terminado con ellos para evitar fugas de memoria. Usa db.close() y cursor.close().
- **Contrato de la Base de Datos:** Define todas tus constantes de tabla y columna en una clase de "contrato" (como ContactosContract) para evitar errores tipográficos y mejorar la mantenibilidad.
- **Frameworks ORM:** Para aplicaciones más grandes o complejas, considera usar una biblioteca ORM (Object-Relational Mapping) como **Room Persistence Library** (parte de Android Jetpack). Simplifica mucho el manejo de SQLite.

---

6. Herramientas Útiles

- Database Inspector en Android Studio: A partir de Android Studio 4.1, puedes inspeccionar, consultar y modificar las bases de datos de tu aplicación directamente desde el IDE. ¡Es una herramienta increíble!

* Para usarlo, ejecuta tu app en un dispositivo o emulador, ve a **View > Tool Windows > Database Inspector**.

- **DB Browser for SQLite:** Una herramienta de escritorio gratuita y de código abierto para crear, diseñar y editar archivos de base de datos SQLite. Útil para verificar tus .db files manualmente.

---

7. Conclusión

El manejo de SQLite es fundamental para muchas aplicaciones Android que necesitan persistir datos localmente. Dominar SQLiteOpenHelper, SQLiteDatabase y Cursor te dará una base sólida para construir aplicaciones robustas.