SQLITE

Carlos Maldonado

5IV7

1. Introducción a SQLite

1.1 ¿Qué es SQLite?

SQLite es un sistema de gestión de bases de datos relacional (RDBMS) que se distingue por ser **autónomo, sin servidor, de configuración cero y transaccional**. A diferencia de otros sistemas de bases de datos como MySQL o PostgreSQL, SQLite no requiere un proceso de servidor separado. La base de datos completa se almacena en un único archivo cross-platform, lo que la hace ideal para aplicaciones embebidas y móviles.

1.2 Historia y desarrollo

SQLite fue creado por D. Richard Hipp en el año 2000 con el objetivo de proporcionar una base de datos que no requiriera administración. Su diseño se centra en la simplicidad, robustez y portabilidad. Desde su creación, SQLite ha ganado popularidad masiva y se ha convertido en la base de datos más implementada en el mundo, estando incluida en todos los sistemas operativos móviles, navegadores web y numerosas aplicaciones de escritorio.

1.3 Características técnicas principales

- **Sin configuración**: No requiere instalación ni configuración
- **Sin servidor**: La base de datos es un archivo ordinario
- **Transaccional ACID**: Cumple con las propiedades Atomicidad, Consistencia, Aislamiento y Durabilidad
- **Independiente de plataforma**: El mismo archivo de base de datos funciona en diferentes sistemas operativos
- **Tipo de datos dinámico**: Aunque es tipado estáticamente, permite cierta flexibilidad
- **Licencia de dominio público**: Libre de regalías

2. Características de SQLite

2.1 Arquitectura sin servidor

La arquitectura de SQLite difiere fundamentalmente de las bases de datos cliente-servidor tradicionales. En lugar de tener un proceso servidor que maneje las solicitudes de múltiples clientes, SQLite se enlaza directamente con la aplicación. Esto significa que la biblioteca de SQLite se vincula con la aplicación y ambas ejecutan en el mismo espacio de proceso.

Ventajas de esta arquitectura:

- Menor latencia en las operaciones
- No hay overhead de comunicación entre procesos
- Configuración simplificada
- Ideal para aplicaciones de un solo usuario

2.2 Sistema de tipos único

SQLite utiliza un sistema de tipado dinámico y débil, lo que significa que el tipo de dato está asociado con el valor mismo, no con la columna. Esto permite mayor flexibilidad pero requiere cuidado en el diseño del esquema.

Tipos de datos soportados:

- NULL: Valor nulo
- INTEGER: Números enteros con signo
- REAL: Números de punto flotante
- TEXT: Cadenas de texto
- BLOB: Datos binarios

2.3 Compatibilidad con SQL

SQLite soporta la mayoría del estándar SQL-92, incluyendo:

- Consultas SELECT complejas con JOINs
- Transacciones
- Vistas
- Triggers
- Funciones agregadas definidas por el usuario
- Expresiones comunes de tabla (CTE)

3. Ventajas de usar SQLite en Android

3.1 Integración nativa

Android incluye soporte nativo para SQLite a través de las APIs del framework. Esto significa que los desarrolladores no necesitan incluir bibliotecas adicionales o configurar dependencias externas.

3.2 Eficiencia de recursos

SQLite es extremadamente eficiente en el uso de recursos, lo que es crucial en dispositivos móviles con limitaciones de memoria y procesamiento. La base de datos opera completamente en el espacio de usuario sin requerir procesos adicionales.

3.3 Persistencia confiable

Las transacciones ACID garantizan la integridad de los datos incluso en casos de fallos del sistema o interrupciones de la aplicación. Los datos persisten entre sesiones de la aplicación y reinicios del dispositivo.

3.4 Portabilidad

El archivo de base de datos puede ser fácilmente respaldado, transferido entre dispositivos o incluso utilizado en diferentes plataformas.

3.5 Seguridad

Los datos se almacenan localmente en el dispositivo, lo que reduce los riesgos de seguridad asociados con la transmisión de datos a servidores remotos. Además, Android proporciona mecanismos para proteger el acceso a la base de datos.

4. Arquitectura de SQLite en Android

4.1 Componentes del stack de base de datos

El stack completo de base de datos en Android consiste en varias capas

4.2 Ubicación de la base de datos

En Android, las bases de datos SQLite se almacenan por defecto en el directorio /data/data/<package_name>/databases/. Esta ubicación es privada para la aplicación y no es accesible por otras aplicaciones sin permisos root.

4.3 Gestión de versiones

Android proporciona mecanismos para manejar la evolución del esquema de la base de datos a través del control de versiones. Esto permite realizar migraciones de datos cuando la estructura de la base de datos cambia entre versiones de la aplicación.

5. Componentes principales de Android para SQLite

5.1 SQLiteOpenHelper

SQLiteOpenHelper es una clase abstracta que gestiona la creación y actualización de la base de datos. Los desarrolladores deben extender esta clase e implementar los métodos onCreate() y onUpgrade().

Métodos principales:

- onCreate(SQLiteDatabase db): Se ejecuta cuando la base de datos se crea por primera vez
- onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion): Se ejecuta cuando la versión de la base de datos cambia
- getReadableDatabase(): Obtiene una base de datos para lectura
- getWritableDatabase(): Obtiene una base de datos para escritura

5.2 SQLiteDatabase

Representa la base de datos y proporciona métodos para realizar operaciones CRUD (Create, Read, Update, Delete).

Operaciones principales:

- insert(): Insertar nuevos registros
- query() o rawQuery(): Consultar datos
- update(): Actualizar registros existentes
- delete(): Eliminar registros
- execSQL(): Ejecutar sentencias SQL directamente

5.3 Cursor

Un Cursor proporciona acceso aleatorio a los resultados de una consulta. Actúa como un iterador sobre los registros devueltos por una consulta a la base de datos.

Métodos importantes del Cursor:

- moveToFirst(), moveToNext(): Navegación por los resultados
- getString(), getInt(), etc.: Obtención de valores de columnas
- getCount(): Número total de registros
- close(): Liberación de recursos

5.4 ContentValues

ContentValues es una clase que almacena pares clave-valor, donde las claves son los nombres de las columnas y los valores son los datos a insertar o actualizar. Simplifica las operaciones de inserción y actualización.

6. Implementación paso a paso

6.1 Diseño del esquema de base de datos

Antes de implementar la base de datos, es crucial diseñar el esquema cuidadosamente. Esto incluye:

- Definir tablas y sus relaciones
- Especificar tipos de datos para cada columna
- Identificar claves primarias y foráneas
- Planificar índices para mejorar el rendimiento

6.2 Creación de la clase Database Helper

```
public class DatabaseHelper extends SQLiteOpenHelper {

  // Información de la base de datos
  private static final String DATABASE_NAME = "mi_aplicacion.db";
  private static final int DATABASE_VERSION = 1;

  // Nombres de tablas
  public static final String TABLE_TAREAS = "tareas";

  // Columnas de la tabla tareas
  public static final String COLUMN_ID = "_id";
  public static final String COLUMN_TITULO = "titulo";
  public static final String COLUMN_DESCRIPCION = "descripcion";
  public static final String COLUMN_FECHA_CREACION = "fecha_creacion";
  public static final String COLUMN_COMPLETADA = "completada";

  // Sentencia SQL para crear la tabla
  private static final String CREATE_TABLE_TAREAS =
    "CREATE TABLE " + TABLE_TAREAS + "(" +
      COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
      COLUMN_TITULO + " TEXT NOT NULL, " +
      COLUMN_DESCRIPCION + " TEXT, " +
      COLUMN_FECHA_CREACION + " DATETIME DEFAULT CURRENT_TIMESTAMP,
" +
      COLUMN_COMPLETADA + " INTEGER DEFAULT 0" +
    ");";

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

  @Override
  public void onCreate(SQLiteDatabase db) {
    db.execSQL(CREATE_TABLE_TAREAS);
  }
```

```
  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // En caso de actualización, eliminar la tabla existente y recrearla
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_TAREAS);
    onCreate(db);
  }
}
```

6.3 Operaciones CRUD básicas

Inserción de datos:

```
public long insertarTarea(String titulo, String descripcion) {
  SQLiteDatabase db = this.getWritableDatabase();
  ContentValues values = new ContentValues();
  values.put(COLUMN_TITULO, titulo);
  values.put(COLUMN_DESCRIPCION, descripcion);

  long id = db.insert(TABLE_TAREAS, null, values);
  db.close();
  return id;
}
```

Consulta de datos:

```
public List<Tarea> obtenerTodasLasTareas() {
  List<Tarea> tareas = new ArrayList<>();
  SQLiteDatabase db = this.getReadableDatabase();

  String[] columnas = {
    COLUMN_ID,
    COLUMN_TITULO,
    COLUMN_DESCRIPCION,
    COLUMN_FECHA_CREACION,
    COLUMN_COMPLETADA
  };
```

```
  Cursor cursor = db.query(
    TABLE_TAREAS,
    columnas,
    null, null, null, null,
    COLUMN_FECHA_CREACION + " DESC"
  );

  if (cursor != null && cursor.moveToFirst()) {
    do {
      Tarea tarea = new Tarea();
      tarea.setId(cursor.getLong(cursor.getColumnIndex(COLUMN_ID)));

tarea.setTitulo(cursor.getString(cursor.getColumnIndex(COLUMN_TITULO)));

tarea.setDescripcion(cursor.getString(cursor.getColumnIndex(COLUMN_DESCRI
PCION)));

tarea.setFechaCreacion(cursor.getString(cursor.getColumnIndex(COLUMN_FEC
HA_CREACION)));

tarea.setCompletada(cursor.getInt(cursor.getColumnIndex(COLUMN_COMPLETA
DA)) == 1);

      tareas.add(tarea);
    } while (cursor.moveToNext());

    cursor.close();
  }

  db.close();
  return tareas;
}
```

Actualización de datos:

```
public int actualizarTarea(Tarea tarea) {
  SQLiteDatabase db = this.getWritableDatabase();
  ContentValues values = new ContentValues();
  values.put(COLUMN_TITULO, tarea.getTitulo());
  values.put(COLUMN_DESCRIPCION, tarea.getDescripcion());
  values.put(COLUMN_COMPLETADA, tarea.isCompletada() ? 1 : 0);

  int filasAfectadas = db.update(
    TABLE_TAREAS,
    values,
    COLUMN_ID + " = ?",
    new String[]{String.valueOf(tarea.getId())}
  );

  db.close();
  return filasAfectadas;
}
```

Eliminación de datos:

```
public void eliminarTarea(long id) {
  SQLiteDatabase db = this.getWritableDatabase();
  db.delete(
    TABLE_TAREAS,
    COLUMN_ID + " = ?",
    new String[]{String.valueOf(id)}
  );
  db.close();
}
```

7. Ejemplo Práctico: Lista de Tareas

Paso 1: Crear clase DatabaseHelper

```
public class DatabaseHelper extends SQLiteOpenHelper {
  private static final String DATABASE_NAME = "tareas.db";
  private static final int DATABASE_VERSION = 1;

  private static final String TABLE_TAREAS = "tareas";
  private static final String COLUMN_ID = "_id";
  private static final String COLUMN_TITULO = "titulo";
  private static final String COLUMN_COMPLETADA = "completada";

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

  @Override
  public void onCreate(SQLiteDatabase db) {
    String CREATE_TABLE = "CREATE TABLE " + TABLE_TAREAS + "(" +
        COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
        COLUMN_TITULO + " TEXT, " +
        COLUMN_COMPLETADA + " INTEGER DEFAULT 0)";
    db.execSQL(CREATE_TABLE);
  }

  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_TAREAS);
    onCreate(db);
  }

  // INSERTAR tarea
  public long agregarTarea(String titulo) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(COLUMN_TITULO, titulo);
    long id = db.insert(TABLE_TAREAS, null, values);
    db.close();
```

```
        return id;
    }

    // OBTENER todas las tareas
    public List<String> obtenerTareas() {
        List<String> tareas = new ArrayList<>();
        SQLiteDatabase db = this.getReadableDatabase();

        String[] columnas = {COLUMN_ID, COLUMN_TITULO,
COLUMN_COMPLETADA};
        Cursor cursor = db.query(TABLE_TAREAS, columnas, null, null, null, null, null);

        if (cursor.moveToFirst()) {
            do {
                String tarea = cursor.getString(1); // COLUMN_TITULO
                tareas.add(tarea);
            } while (cursor.moveToNext());
        }
        cursor.close();
        db.close();
        return tareas;
    }

    // ELIMINAR tarea
    public void eliminarTarea(String titulo) {
        SQLiteDatabase db = this.getWritableDatabase();
        db.delete(TABLE_TAREAS, COLUMN_TITULO + " = ?", new String[]{titulo});
        db.close();
    }
}
```

Paso 2: Layout principal (activity_main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:id="@+id/editTextTarea"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Escribe una tarea..." />

    <Button
        android:id="@+id/btnAgregar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Agregar Tarea"
        android:layout_marginTop="8dp" />

    <ListView
        android:id="@+id/listViewTareas"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="16dp" />

</LinearLayout>
```

Paso 3: MainActivity

```
public class MainActivity extends AppCompatActivity {
  private DatabaseHelper dbHelper;
  private EditText editTextTarea;
  private Button btnAgregar;
  private ListView listViewTareas;
  private ArrayAdapter<String> adapter;
  private List<String> listaTareas;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    // Inicializar base de datos
    dbHelper = new DatabaseHelper(this);

    // Referencias a vistas
    editTextTarea = findViewById(R.id.editTextTarea);
    btnAgregar = findViewById(R.id.btnAgregar);
    listViewTareas = findViewById(R.id.listViewTareas);

    // Cargar tareas existentes
    cargarTareas();

    // Botón agregar
    btnAgregar.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        agregarTarea();
      }
    });
    // Long click para eliminar
    listViewTareas.setOnItemLongClickListener(new
AdapterView.OnItemLongClickListener() {
      @Override
```

```
        public boolean onItemLongClick(AdapterView<?> parent, View view, int
position, long id) {
            eliminarTarea(position);
            return true;
        }
    });
  }
  private void cargarTareas() {
    listaTareas = dbHelper.obtenerTareas();
    adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1,
listaTareas);
    listViewTareas.setAdapter(adapter);
  }
  private void agregarTarea() {
    String tarea = editTextTarea.getText().toString().trim();
    if (!tarea.isEmpty()) {
      dbHelper.agregarTarea(tarea);
      editTextTarea.setText("");
      cargarTareas();
      Toast.makeText(this, "Tarea agregada", Toast.LENGTH_SHORT).show();
    } else {
      Toast.makeText(this, "Escribe una tarea", Toast.LENGTH_SHORT).show();
    }
  }
  private void eliminarTarea(int position) {
    String tarea = listaTareas.get(position);
    dbHelper.eliminarTarea(tarea);
    cargarTareas();
    Toast.makeText(this, "Tarea eliminada", Toast.LENGTH_SHORT).show();
  }
  @Override
  protected void onDestroy() {
    dbHelper.close();
    super.onDestroy();
  }

}
```

Paso 4: Configurar AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.ejemplo.tareas">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">

    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

  </application>
</manifest>
```

Bibliografía

- García, J. (2022). *Desarrollo de aplicaciones Android con Java y SQLite*. Editorial Alfaomega.
- López, M. (2021). *Bases de datos SQLite para Android: Guía práctica para principiantes*. Ediciones Marcombo.
- Ortega, R. (2023). *Programación Android: Uso de SQLite y almacenamiento local*. Ediciones RA-MA.
- Sánchez, L. (2020). *Android Studio: Desarrollo de aplicaciones móviles paso a paso*. Editorial Anaya Multimedia.
- Google Developers. (2024). *Guía oficial de almacenamiento con SQLite en Android*. Recuperado de: https://developer.android.com/training/data-storage/sqlite?hl=es-419
- Sitio Oficial SQLite. (2024). *Documentación de SQLite en español*. Recuperado de: https://www.sqlite.org/spanish.html
- Ramos, P. (2022). "Implementación de bases de datos SQLite en aplicaciones móviles Android". *Revista Iberoamericana de Tecnología Educativa y Móvil*, 18(2), 45-58.
- Díaz, A. (2021). *Introducción al desarrollo móvil con SQLite en Android*. Universidad Tecnológica de México (UNITEC).