Bases de datos (1)

Cada dispositivo Android cuenta con una base de datos SQLite que se puede utilizar desde nuestras aplicaciones para almacenar información.

SQLite es una BD relacional *open source* con una sintaxis casi idéntica a la de SQL y con los mismos tipos (INTEGER, TEXT, etc.), pero con la ventaja de que consume muy pocos recursos.

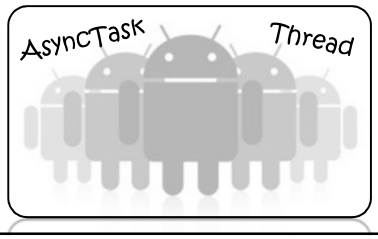
24

Bases de datos (2)

Una BD SQLite es privada, sólo se puede acceder a ella directamente desde la aplicación desde la que fue creada, y se almacena en la siguiente ubicación:

data/data/paquete_app/databases

Las operaciones de lectura/escritura pueden suponer cierto tiempo de espera, por lo que deben ser ejecutadas en segundo plano utilizando otro hilo (***Thread***) o una tarea asíncrona (***AsyncTask***).



25

Bases de datos (3)

- Acceder a la BD: Método 1

El método más sencillo para utilizar una BD SQLite consiste en utilizar un objeto *SQLiteDatabase* que obtendremos invocando al método estático *openOrCreateDatabase*, al que pasaremos como parámetros:

- Nombre de la BD (*String*)
- Modo: normalmente *Context.MODE_PRIVATE*
- *CursorFactory*: podemos dejarlo a *null*

```
SQLiteDatabase db;
db = openOrCreateDatabase("MisContactos", Context.MODE_PRIVATE, null);
```

26

Bases de datos (4)

- Acceder a la BD: Método 1

Una vez que tenemos el objeto *SQLiteDatabase*, ya podremos ejecutar sentencias SQL sobre él utilizando el método *execSQL* que recibirá como parámetro un *String* con la sentencia SQL a ejecutar.

- Crear tabla:

```
String sql = "CREATE TABLE IF NOT EXISTS Contactos (Nom VARCHAR, Ape VARCHAR);"
db.execSQL(sql);
```

- Insertar un registro:

```
String sql = "INSERT INTO Contactos VALUES ('"+nombre+"','"+apellido+"');"
db.execSQL(sql);
```

27

Bases de datos (5)

- Acceder a la BD: Método 1

Para realizar una consulta utilizaremos el método *rawQuery* que recibe como párametro un *String* con la consulta a ejecutar (*Select*) y devuelve un cursor con el que podremos acceder a los datos.

Utilizaremos el cursor para recorrer los registros obtenidos y al finalizar lo cerraremos mediante *close*.

```
Cursor cur = db.rawQuery("SELECT * FROM Contactos", null);
if (cur.getCount() == 0) {
  // No obtenemos ningún resultado
} else {
  while (cur.moveToNext()) {
    String nombre = cur.getString(0);
    String apellido = cur.getString(1);
    ...
  }
  cur.close();
}
```

Bases de datos (6)

- Acceder a la BD: Método 1

Una vez que hayamos terminado con la BD tendremos que cerrar la conexión para liberar los recursos utilizando el método *close* sobre el objeto *SQLiteDatabase*.

```
// Cerrar la conexión con la BD
db.close();
```

29

Bases de datos (7)

- Acceder a la BD: Método 2

En el método anterior es sencillo obtener y añadir datos a nuestra BD SQLite, pero no ofrece ningún mecanismo para comprobar la estructura de la BD, de manera que deberemos realizarlo manualmente, para evitar accesos erróneos si la estructura cambia en tiempo de ejecución.

Para evitarlo se aconseja crear una clase que extienda a *SQLiteOpenHelper*, de modo que se encapsule toda la lógica de acceso a la BD, y al mismo tiempo, se disponga de todas las utilidades ofrecidas por Android para su manejo.

30

Bases de datos (8)

- Acceder a la BD: Método 2

Esta clase que extiende a ***SQLiteOpenHelper*** dispone de los siguientes métodos:

- ***onCreate***: se encarga de la creación de la BD.
- ***onUpgrade***: mantenimiento de la estructura (añadir, eliminar o modificar tablas) para pasar de una versión a otra de la BD.
- ***getReadableDatabase***: devuelve un objeto *SQLiteDatabase* sobre el que realizar operaciones de consulta.
- ***getWritableDatabase***: también devuelve un objeto *SQLiteDatabase*, pero permite escribir datos.

31

Bases de datos (9)

- Acceder a la BD: Método 2

```
public class ContactoHelper extends SQLiteOpenHelper {
  private static final String TABLA_CONTACTOS = "TABLA_CONTACTOS";
  private static final String COL_ID = "ID";
  private static final String COL_NOMBRE = "NOMBRE";
  private static final String COL_TLF = "TELEFONO";
  private static final String CREATE_BD = "CREATE TABLE " + TABLA_CONTACTOS
  + " ("+ COL_ID +" INTEGER PRIMARY KEY, "+ COL_NOMBRE
  + " TEXT NOT NULL, " + COL_TLF + " TEXT NOT NULL);";

  public ContactoHelper(Context c,String name,CursorFactory cf,int version) {
    super(context, name, factory, version);
  }

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

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

Bases de datos (10)

- Acceder a la BD: Método 2

Toda tabla *SQLITE* incluye de forma automática un campo *ROWID* que funciona como clave primaria numérica que se autoincrementa.

Cuando se define un campo de tipo *INTEGER PRIMARY KEY*, funciona como alias de *ROWID*, de manera que si insertamos un registro y no establecemos el valor de este campo, éste se asignará de forma automática (autoincremento).

No se recomienda el uso de la clausula *AUTOINCREMENT* porque aumenta de forma considerable el consumo de recursos.

Leer más: https://www.sqlite.org/autoinc.html

33

Bases de datos (11)

- Acceder a la BD: Método 2

A continuación, creamos una clase que se encargará de gestionar los datos de la tabla: inserción, consulta, borrado, etc.

```
public class ContactoBD {
  private static final int VERSION = 1;
  private static final String BD_NOMBRE = "contactos.db";
  private static final String TABLA_CONTACTOS = "TABLA_CONTACTOS";
  private static final String COL_ID = "ID";
  private static final int NUM_COL_ID = 0;
  private static final String COL_NOMBRE = "NOMBRE";
  private static final int NUM_COL_NOMBRE = 1;
  private static final String COL_TLF = "TELEFONO";
  private static final int NUM_COL_TLF = 2;

  private SQLiteDatabase bd;
  private ContactoHelper contactos;
  // Métodos para gestionar los datos
}
```

34

Bases de datos (12)

- Acceder a la BD: Método 2

```
// Métodos para gestionar los datos
public ContactoBD (Context context) {
    contactos = new ContactoHelper(context, BD_NOMBRE, null, VERSION);  }

  public void openForWrite() {
    bd = contactos.getWritableDatabase(); }

  public void openForRead() {
    bd = contactos.getReadableDatabase(); }

  public SQLiteDatabase getBD() {
    return bd; }

  public long insertarContacto (Contacto contacto) {
    ContentValues content = new ContentValues();
    content.put(COL_NOMBRE, contacto.getNombre());
    content.put(COL_TLF, contacto.getTelefono());
    return  bd.insert(TABLA_CONTACTOS, null, content);
  }

  /* Resto de métodos */
}
```

Bases de datos (13)

- Consultas con filtro

Independientemente del método utilizado, es frecuente que a la hora de obtener datos de una BD, necesitemos filtrar los resultados (cláusula WHERE).

SQLite nos ofrece varios métodos para realizar el filtrado (*rawQuery*, *query*) y pueden utilizarse con y sin parámetros.

- ***rawQuery*** *(String query, String[] selecArgs)*
- ***query*** *(String table, String[] columns, String selec, String[] selecArgs, String groupBy, String having, String orderBy, String limit)*

36

Bases de datos (13)

- Consultas con filtro: rawQuery

- **Sin parámetros**

```
String nombre = "Juan";
String sql = "SELECT * FROM contactos WHERE nombre ='" + nombre + "'";
Cursor cur = bd.rawQuery(sql, null);
```

- **Con parámetros**

```
String [] valores = {"Juan"};
String sql = "SELECT * FROM contactos WHERE nombre = ?";
Cursor cur = bd.rawQuery(sql, valores);
```

37

Bases de datos (14)

- Consultas con filtro: query

- **Sin parámetros**

```
String nombre = "Ana";
String campos [] = new String[] {COL_NOMBRE, COL_APE1, COL_APE2, COL_TLF};
String where = COL_NOMBRE + "='" + nombre + "'";
Cursor cur = bd.query(TABLA_CONTACTOS,campos,where,null,null,null,null);
```

- **Con parámetros**

```
String campos [] = new String[] {COL_NOMBRE, COL_APE1, COL_APE2, COL_TLF};
String where = COL_NOMBRE + "=?";
String [] valores = {"Ana"};
Cursor cur = bd.query(TABLA_CONTACTOS,campos,where,valores,null,null,null);
```

38

Bases de datos (15)

- Comprobar las bases de datos

Android Studio nos proporciona las herramientas **adb** y **sqlite3** para acceder a nuestras bases de datos.

La herramienta **adb** se encuentra en nuestra carpeta:

android_sdk/platform-tools

Podemos comprobar los dispositivos disponibles mediante ***adb devices***

```
C:\Android\SDK\platform-tools>adb devices
List of devices attached
emulator-5554   device
```

Y conectarnos al dispositivo mediante ***adb shell***

```
C:\Android\SDK\platform-tools>adb shell
root@generic_x86:/ #
```

39

Bases de datos (16)

- Comprobar las bases de datos

Una vez dentro del dispositivo accederemos a la carpeta que contiene la base de datos:

cd /data/data/paquete_app/databases

```
C:\Android\SDK\platform-tools>adb shell
root@generic_x86:/ # cd /data/data/com.example.oscar.exbasedatos02/databases
root@generic_x86:/data/data/com.example.oscar.exbasedatos02/databases #
```

Y nos conectamos a nuestra BD: ***sqlite3 nombreBD***

```
127|root@generic_x86:/data/data/com.example.oscar.exbasedatos02/databases #
ls
contactos.db
contactos.db-journal
root@generic_x86:/data/data/com.example.oscar.exbasedatos02/databases #
sqlite3 contactos.db
SQLite version 3.8.6.1 2015-05-21 17:24:32
Enter ".help" for usage hints.
```

40

Bases de datos (17)

- Comprobar las bases de datos

Tras establecer la conexión con nuestra base de datos, podemos ejecutar sentencias SQL.

Por ejemplo: select * from TABLA_CONTACTOS

```
sqlite> select * from TABLA_CONTACTOS;
6|Oscar|Hernando|Tejedor|654123123
7|Ana|Mart├¡n|L├¡pez|678253785
8|Javier|Sanz|Campos|612651672
9|Teresa|├ülvarez|Iglesias|655778123
```

Más información:

- adb:
  https://developer.android.com/studio/command-line/adb.html?hl=es-419
- sqlite:
  https://www.imaginanet.com/blog/primeros-pasos-con-sqlite3-comandos-basicos.html

41