

Persistencia de datos

Profesor: Ana Isabel Vegas

INDICE

1.- GESTION DE DATOS CON SQLITE	3
2.- LA API SQLITE	4
3.- CREAR UNA BASE DE DATOS	5
4.- CREACIÓN DE TABLAS	7
5.- INSERCIÓN DE REGISTROS.....	8
6.- ELIMINAR REGISTROS	9
7.- MODIFICACION DE REGISTROS.....	10
8.- CONSULTA DE REGISTROS	11
9.- DONDE ESTA LA BASE DE DATOS.....	13
ÍNDICE DE GRÁFICOS	15

1.- GESTION DE DATOS CON SQLITE

Las bases de datos son una herramienta de gran potencia en la creación de aplicaciones informáticas. Hasta hace muy poco resultaba muy costoso y complejo incorporar bases de datos a nuestras aplicaciones. Afortunadamente, Android incorpora la librería SQLite que nos permitirá utilizar bases de datos mediante el lenguaje SQL, de una forma sencilla y utilizando muy pocos recursos del sistema.

SQLite es un motor de base de datos relacional de código abierto y muy potente, eso hace que actualmente sea muy usado por los desarrolladores. Sus principales características son que precisa de poca configuración, no necesita ningún servidor ya que directamente lee y escribe en archivos de disco normales, ocupa muy poco tamaño en el almacenamiento y a parte es multiplataforma.

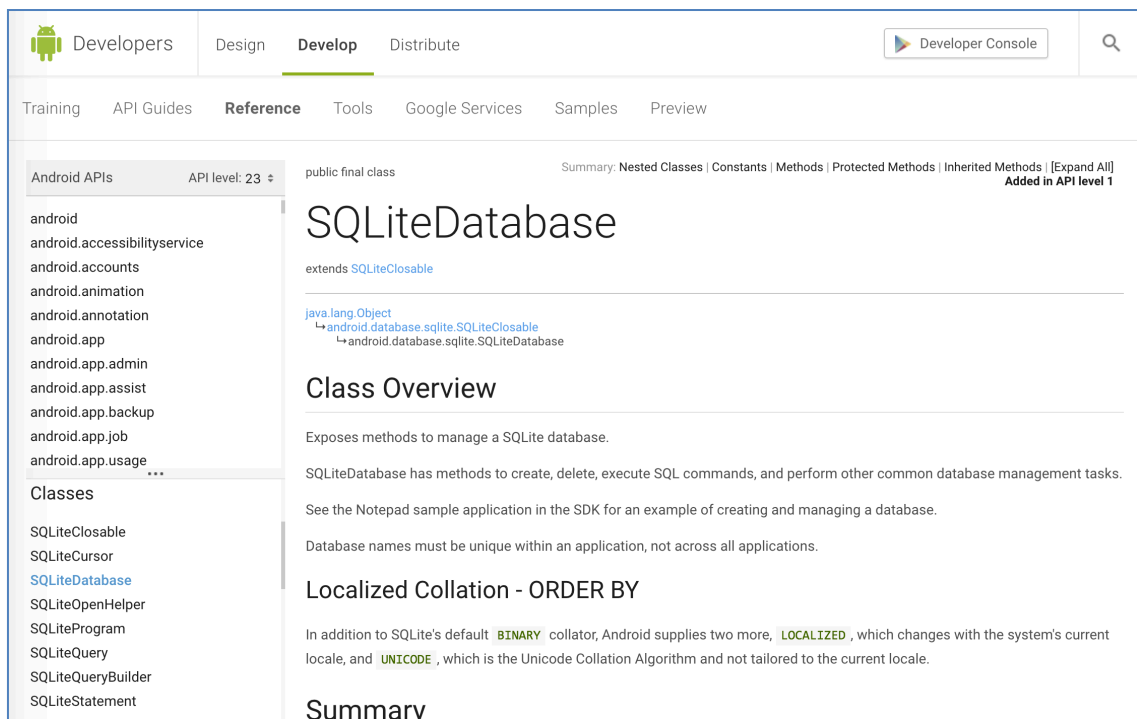
Android ofrece de serie soporte total para la creación y administración de base de datos SQLite a través del paquete "android.database.sqlite". Solo tendremos que definir las sentencias SQL para crear y gestionar la base de datos.

2.- LA API SQLITE

El sitio web donde poder consultar su API y descargar documentación adicional es: <https://www.sqlite.org/>

Desde la pagina de Android Developer es la mejor opción para consultar su API:

<http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>



The screenshot shows the Android Studio interface with the API reference for `SQLiteDatabase` selected. The top navigation bar includes 'Design', 'Develop' (active), and 'Distribute'. Below it, a secondary bar shows 'Training', 'API Guides', 'Reference' (active), 'Tools', 'Google Services', 'Samples', and 'Preview'. The left sidebar lists various Android APIs, with 'SQLiteDatabase' highlighted under the 'Classes' section. The main content area displays the class details for `SQLiteDatabase`, including its inheritance hierarchy (extending `SQLiteClosable` and `java.lang.Object`), a 'Class Overview' section describing its purpose and usage, a 'Localized Collation - ORDER BY' section, and a 'Summary' section. The page also indicates it was 'Added in API level 1'.

Gráfico 1. Consulta API SQLite

3. - CREAR UNA BASE DE DATOS

Para crear nuestra base de datos nos basaremos en la clase SQLiteOpenHelper que nos facilita tanto la creación de la base de datos, como el trabajar con futuras versiones de esta base de datos. Para crear una subclase hay que implementar los métodos onCreate() y onUpgrade() y opcionalmente onOpen().

La gran ventaja de utilizar esta clase es que ella se preocupará de abrir la base de datos si existe o de crearla si no existe. Incluso de actualizar la versión si decidimos crear una nueva estructura de la base de datos.

Adicionalmente esta Clase tiene dos métodos: getReadableDatabase() y getWritableDatabase() que abren la base de datos en modo solo lectura o lectura y escritura. En caso de todavía no existir estos métodos se encargarán de crearla.

```
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        try {
            db.execSQL(DATABASE_CREATE);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
            + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS contacts");
        onCreate(db);
    }
}
```

Grafico 2. Métodos onCreate() y onUpgrade()

En la siguiente imagen vemos como abrir y cerrar la base de datos

```
// ---opens the database---  
public DBAdapter open() throws SQLException {  
    db = DBHelper.getWritableDatabase();  
    return this;  
}  
  
// ---closes the database---  
public void close() {  
    DBHelper.close();  
}
```

Gráfico 3. Métodos open() y close()

4.- CREACIÓN DE TABLAS

Una vez que tenemos la base de datos creada ya podemos crear la tabla.

Necesitamos una clase que herede de la clase SQLiteOpenHelper y en ella sobrescribir los métodos onCreate y onUpgrade:

El método onCreate se utiliza para crear la tabla por primera vez.

```
@Override
public void onCreate(SQLiteDatabase db) {
    // Crear la tabla de Usuarios con los campos Usuario y password
    db.execSQL("CREATE TABLE Usuarios (usuario TEXT, password TEXT);");
}
```

Gráfico 4. Crear la tabla por primera vez

Mientras que el método onUpgrade se utiliza para realizar modificaciones sobre la estructura de la tabla.

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Entendemos que aquí se haría previamente una migración
    // de los datos de la tabla antigua a la nueva
    db.execSQL("DROP TABLE IF EXISTS Usuarios");
    db.execSQL("CREATE TABLE Usuarios (usuario TEXT, password TEXT);");
}
```

Gráfico 5. Modificar la estructura de la tabla

5.- INSERCIÓN DE REGISTROS

Para insertar un registro podemos lanzar una query SQL a través del método `execSQL` del objeto que tiene acceso a la base de datos

```
db.execSQL("INSERT INTO Usuarios (usuario,password) VALUES('" + nombre + "','" + password + "')");
```

Gráfico 6. Query SQL de inserción

Otra opción es utilizar el método `insert`. Primero creamos un objeto de tipo `ContentValues` donde agregamos los datos a insertar.

```
ContentValues nuevoRegistro = new ContentValues();  
nuevoRegistro.put("usuario", "usuario6");  
nuevoRegistro.put("password", "pw6");  
db.insert("Usuarios", null, nuevoRegistro);
```

Gráfico 7. Inserción de registros

6.- ELIMINAR REGISTROS

Igual que antes podemos lanzar la query SQL:

```
// Borrar un registro  
db.execSQL("DELETE FROM Usuarios WHERE usuario='usuario3' ");
```

Gráfico 8. Query SQL de eliminación

O utilizar el método delete:

```
// Otra forma de borrar registros  
db.delete("Usuarios", "usuario='usuario2'", null);
```

Gráfico 9. Borrado de registros

7.- MODIFICACION DE REGISTROS

```
// Modificar un registro  
db.execSQL("UPDATE Usuarios SET password='nueva' WHERE usuario='usuario1' ");
```

Gráfico 10. Consulta SQL de modificación

```
// Otra forma de modificar registros  
ContentValues modificar = new ContentValues();  
modificar.put("password", "pwOtra");  
db.update("Usuarios", modificar, "usuario='usuario1'", null);
```

Gráfico 11. Método update

8.- CONSULTA DE REGISTROS

Consultamos todos los registros obteniendo el resultado en un objeto de tipo Cursor.

```
// Consultar todos los registros  
Cursor c1 = db.rawQuery("SELECT * FROM Usuarios", null);
```

Gráfico 12. Consultar todos los registros

Si queremos lanzar una consulta con parámetros lo podemos hacer de esta forma:

```
// Consulta con argumentos  
String[] args1 = new String[]{"usuario5"};  
Cursor c2 = db.rawQuery("SELECT * FROM Usuarios WHERE usuario=?", args1);
```

Gráfico 13. Consulta con parámetros

Aunque esta otra también es posible, utilizando el método query:

```
// Otra forma de generar consultas  
String[] args2 = new String[]{"usuario6"};  
String[] campos = new String[]{"usuario", "password"};  
Cursor c3 = db.query("Usuarios", campos, "usuario=?", args2, null, null, null);
```

Gráfico 14. Consultas con el método query

Una vez obtenidos los resultados de la consulta debemos recorrer el objeto Cursor para obtener cada registro.

```

// Recorrer los resultados obtenidos
// Posicionar el cursor en el primer registro
if (c1.moveToFirst()){

    // recorro todos los registros de la query
    do{
        String usuarioText = c1.getString(0);
        String pwText = c1.getString(1);
        etiqueta.setText(etiqueta.getText() +
            "User: " + usuarioText + " Contraseña: " + pwText + "\n");
    }while(c1.moveToNext());
}

```

Gráfico 15. Procesar datos obtenidos

9.- DONDE ESTA LA BASE DE DATOS

1.- Abrir Android Device Monitor:

Tools/ Android / Android Device Monitor

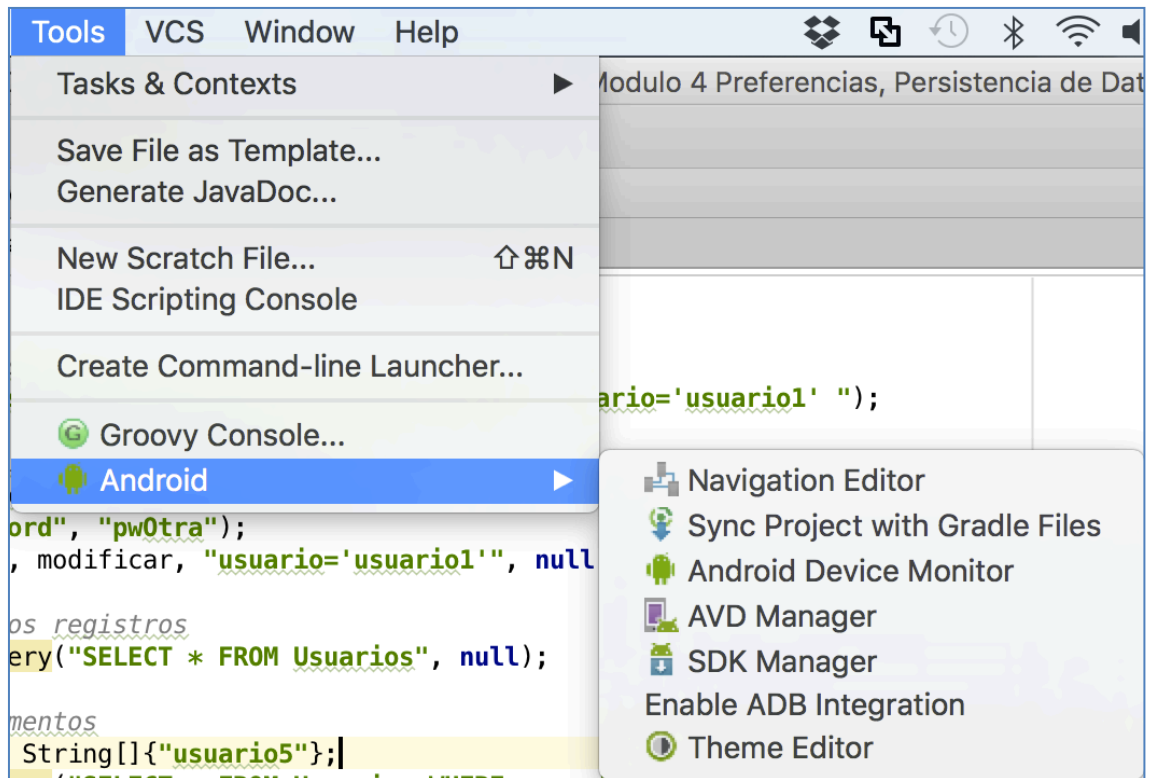


Gráfico 16. Android Device Monitor

2.- Elegimos pestaña File Explorer en la parte derecha de la ventana

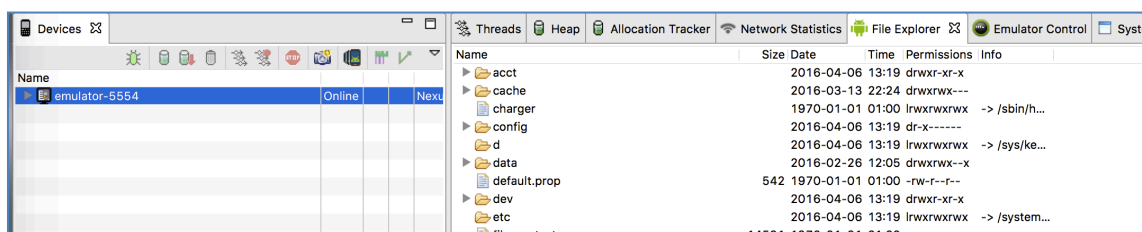


Gráfico 17. File Explorer

3.- La ruta donde la encontraremos es:

data/data/paquete de la aplicacion / databases

▶ com.example.android.livecubes	2016-02-26 12:08	drwxr-x--x
▶ com.example.android.softkeyboard	2016-02-26 12:08	drwxr-x--x
▼ com.example.bdusuarios	2016-04-06 11:58	drwxr-x--x
▶ cache	2016-04-06 11:58	drwxrwx--x
▶ code_cache	2016-04-06 11:58	drwxrwx--x
▼ databases	2016-04-06 11:58	drwxrwx--x
DBUsuarios	16384 2016-04-06 11:58	-rw-rw----
DBUsuarios-journal	8720 2016-04-06 11:58	-rw-----
▶ com.example.ejemplobbdd	2016-04-06 11:56	drwxr-x--x
▶ com.example.ejemplobotones	2016-03-10 16:15	drwxr-x--x
▶ com.example.ejemplocheckbox	2016-03-13 22:13	drwxr-x--x

Gráfico 18. Ubicación de la BBDD

ÍNDICE DE GRÁFICOS

Gráfico 1. Consulta API SQLite	4
Grafico 2. Métodos onCreate() y onUpgrade()	5
Gráfico 3. Métodos open() y close().....	6
Gráfico 4. Crear la tabla por primera vez.....	7
Gráfico 5. Modificar la estructura de la tabla	7
Gráfico 6. Query SQL de inserción	8
Gráfico 7. Inserción de registros.....	8
Gráfico 8. Query SQL de eliminación	9
Gráfico 9. Borrado de registros.....	9
Gráfico 10. Consulta SQL de modificación	10
Gráfico 11. Método update.....	10
Gráfico 12. Consultar todos los registros.....	11
Gráfico 13. Consulta con parámetros	11
Gráfico 14. Consultas con el método query	11
Gráfico 15. Procesar datos obtenidos	12
Gráfico 16. Android Device Monitor	13
Gráfico 17. File Explorer	13
Gráfico 18. Ubicación de la BBDD.....	14