

Clase 5

Curso Desarrollo en ANDROID V2.0

Emiliano Gonzalez

Pedro Coronel

Asunción – Paraguay

2013



Bases de Datos en Android

La plataforma Android proporciona dos herramientas principales para el almacenamiento y consulta de datos estructurados:

- Bases de Datos SQLite
- Content Providers

Nos centraremos en la primera opción, *SQLite*, que abarcará todas las tareas relacionadas con el almacenamiento de los datos propios de nuestra aplicación. El segundo de los mecanismos, los *Content Providers*, que trataremos más adelante, nos facilitarán la tarea de hacer visibles esos datos a otras aplicaciones y, de forma recíproca, de permitir la consulta de datos publicados por terceros desde nuestra aplicación.

SQLite es un motor de bases de datos muy popular en la actualidad por ofrecer características tan interesantes como su pequeño tamaño, no necesitar servidor, precisar poca configuración, ser transaccional y por supuesto ser de código libre.

Android incorpora de serie todas las herramientas necesarias para la creación y gestión de bases de datos SQLite, y entre ellas una completa API para llevar a cabo de manera sencilla todas las tareas necesarias.

En Android, la forma típica para crear, actualizar, y conectar con una base de datos SQLite será a través de una clase auxiliar llamada `SQLiteOpenHelper`, o para ser más exactos, de una clase propia que derive de ella y que debemos personalizar para adaptarnos a las necesidades concretas de nuestra aplicación.

La clase `SQLiteOpenHelper` tiene tan sólo un constructor, que normalmente no necesitaremos sobrescribir, y dos métodos abstractos, `onCreate()` y `onUpgrade()`, que deberemos personalizar con el código necesario para crear nuestra base de datos y para actualizar su estructura respectivamente.

Como ejemplo, nosotros vamos a crear una base de datos muy sencilla llamada `BDUsuarios`, con una sola tabla llamada `Usuarios` que contendrá sólo dos campos: `nombre` y `email`. Para ello, vamos a crear una clase derivada de `SQLiteOpenHelper` que llamaremos `UsuariosSQLiteHelper`, donde sobrescribiremos los métodos `onCreate()` y `onUpgrade()` para adaptarlos a la estructura de datos indicada:

```
1 package py.com.cursoandroid;  
2  
3 import android.content.Context;  
4 import android.database.sqlite.SQLiteDatabase;  
5 import android.database.sqlite.SQLiteDatabase.CursorFactory;
```

```

6  import android.database.sqlite.SQLiteOpenHelper;
7
8  public class UsuariosSQLiteHelper extends SQLiteOpenHelper {
9
10     //Sentencia SQL para crear la tabla de Usuarios
11     String sqlCreate = "CREATE TABLE Usuarios (codigo INTEGER, nombre TEXT)";
12
13     public UsuariosSQLiteHelper(Context contexto, String nombre,
14                                CursorFactory factory, int version) {
15         super(contexto, nombre, factory, version);
16     }
17
18     @Override
19     public void onCreate(SQLiteDatabase db) {
20         //Se ejecuta la sentencia SQL de creación de la tabla
21         db.execSQL(sqlCreate);
22     }
23
24     @Override
25     public void onUpgrade(SQLiteDatabase db, int versionAnterior, int versionNueva) {
26         //NOTA: Por simplicidad del ejemplo aquí utilizamos directamente la opción
27         //        eliminar la tabla anterior y crearla de nuevo vacía con el nuevo
28         //        Sin embargo lo normal será que haya que migrar datos de la tabla
29         //        a la nueva, por lo que este método debería ser más elaborado.
30
31         //Se elimina la versión anterior de la tabla
32         db.execSQL("DROP TABLE IF EXISTS Usuarios");
33
34         //Se crea la nueva versión de la tabla
35         db.execSQL(sqlCreate);
36     }
37 }

```

Lo primero que hacemos es definir una variable llamada `sqlCreate` donde almacenamos la sentencia SQL para crear una tabla llamada `Usuarios` con los campos alfanuméricos `nombre` y `email`. El método `onCreate()` será ejecutado automáticamente por nuestra clase `UsuariosDBHelper` cuando sea necesaria la creación de la base de datos, es decir, cuando aún no exista. Las tareas típicas que deben hacerse en este método serán la creación de todas las tablas necesarias y la inserción de los datos iniciales si son necesarios. En nuestro caso, sólo vamos a crear la tabla `Usuarios` descrita anteriormente. Para la creación de la tabla utilizaremos la sentencia SQL ya definida y la ejecutaremos contra la base de datos utilizando el método más sencillo de los disponibles en la API de SQLite proporcionada por Android, llamado `execSQL()`. Este método se limita a ejecutar directamente el código SQL que le pasemos como parámetro.

Por su parte, el método `onUpgrade()` se lanzará automáticamente cuando sea necesaria una actualización de la estructura de la base de datos o una conversión de los datos. Un

ejemplo práctico: imaginemos que publicamos una aplicación que utiliza una tabla con los campos `usuario` y `email` (llamémoslo versión 1 de la base de datos). Más adelante, ampliamos la funcionalidad de nuestra aplicación y necesitamos que la tabla también incluya un campo adicional por ejemplo con la `edad` del usuario (versión 2 de nuestra base de datos). Pues bien, para que todo funcione correctamente, la primera vez que ejecutemos la versión ampliada de la aplicación necesitaremos modificar la estructura de la tabla `Usuarios` para añadir el nuevo campo `edad`. Pues este tipo de cosas son las que se encargará de hacer automáticamente el método `onUpgrade()` cuando intentemos abrir una versión concreta de la base de datos que aún no exista. Para ello, como parámetros recibe la versión actual de la base de datos en el sistema, y la nueva versión a la que se quiere convertir. En función de esta pareja de datos necesitaremos realizar unas acciones u otras. En nuestro caso de ejemplo optamos por la opción más sencilla: borrar la tabla actual y volver a crearla con la nueva estructura, pero como se indica en los comentarios del código, lo habitual será que necesitemos algo más de lógica para convertir la base de datos de una versión a otra y por supuesto para conservar los datos registrados hasta el momento.

Una vez definida nuestra clase *helper*, la apertura de la base de datos desde nuestra aplicación resulta ser algo de lo más sencillo. Lo primero será crear un objeto de la clase `UsuariosSQLiteHelper` al que pasaremos el contexto de la aplicación (en el ejemplo una referencia a la actividad principal), el nombre de la base de datos, un objeto `CursorFactory` que típicamente no será necesario (en ese caso pasaremos el valor `null`), y por último la versión de la base de datos que necesitamos. La simple creación de este objeto puede tener varios efectos:

- Si la base de datos ya existe y su versión actual coincide con la solicitada simplemente se realizará la conexión con ella.
- Si la base de datos existe pero su versión actual es anterior a la solicitada, se llamará automáticamente al método `onUpgrade()` para convertir la base de datos a la nueva versión y se conectará con la base de datos convertida.
- Si la base de datos no existe, se llamará automáticamente al método `onCreate()` para crearla y se conectará con la base de datos creada.

Una vez tenemos una referencia al objeto `UsuariosSQLiteHelper`, llamaremos a su método `getReadableDatabase()` o `getWritableDatabase()` para obtener una referencia a la base de datos, dependiendo si sólo necesitamos consultar los datos o también necesitamos realizar modificaciones, respectivamente.

Ahora que ya hemos conseguido una referencia a la base de datos (objeto de tipo `SQLiteDatabase`) ya podemos realizar todas las acciones que queramos sobre ella. Para nuestro ejemplo nos limitaremos a insertar 5 registros de prueba, utilizando para ello el

método ya comentado `execSQL()` con las sentencias `INSERT` correspondientes. Por último cerramos la conexión con la base de datos llamando al método `close()`.

```
1 package py.com.cursoandroid;
2
3 import android.app.Activity;
4 import android.database.sqlite.SQLiteDatabase;
5 import android.os.Bundle;
6
7 public class AndroidBaseDatos extends Activity
8 {
9     @Override
10    public void onCreate(Bundle savedInstanceState)
11    {
12        super.onCreate(savedInstanceState);
13        setContentView(R.layout.main);
14
15        //Abrimos la base de datos 'DBUsuarios' en modo escritura
16        UsuariosSQLiteHelper usdbh =
17            new UsuariosSQLiteHelper(this, "DBUsuarios", null, 1);
18
19        SQLiteDatabase db = usdbh.getWritableDatabase();
20
21        //Si hemos abierto correctamente la base de datos
22        if(db != null)
23        {
24            //Insertamos 5 usuarios de ejemplo
25            for(int i=1; i<=5; i++)
26            {
27                //Generamos los datos
28                int codigo = i;
29                String nombre = "Usuario" + i;
30
31                //Insertamos los datos en la tabla Usuarios
32                db.execSQL("INSERT INTO Usuarios (codigo, nombre) " +
33                    "VALUES (" + codigo + ", '" + nombre + "')");
34            }
35
36            //Cerramos la base de datos
37            db.close();
38        }
39    }
40
```

¿y ahora qué? ¿dónde está la base de datos que acabamos de crear? ¿cómo podemos comprobar que todo ha ido bien y que los registros se han insertado correctamente? Vayamos por partes.

En primer lugar veamos dónde se ha creado nuestra base de datos. Todas las bases de datos SQLite creadas por aplicaciones Android se almacenan en la memoria del teléfono

en un fichero con el mismo nombre de la base de datos situado en una ruta que sigue el siguiente patrón:

```
/data/data/paquete.java.de.la.aplicacion/databases/nombre_base_datos
```

En el caso de nuestro ejemplo, la base de datos se almacenaría por tanto en la ruta siguiente:

```
/data/data/py.com.cursoandroid/databases/DBUsuarios
```

Para comprobar esto podemos hacer lo siguiente. Una vez ejecutada por primera vez desde Eclipse la aplicación de ejemplo sobre el emulador de Android (y por supuesto antes de cerrarlo) podemos ir a la perspectiva “DDMS” (*Dalvik Debug Monitor Server*) de Eclipse y en la solapa “*File Explorer*” podremos acceder al sistema de archivos del emulador, donde podremos buscar la ruta indicada de la base de datos.

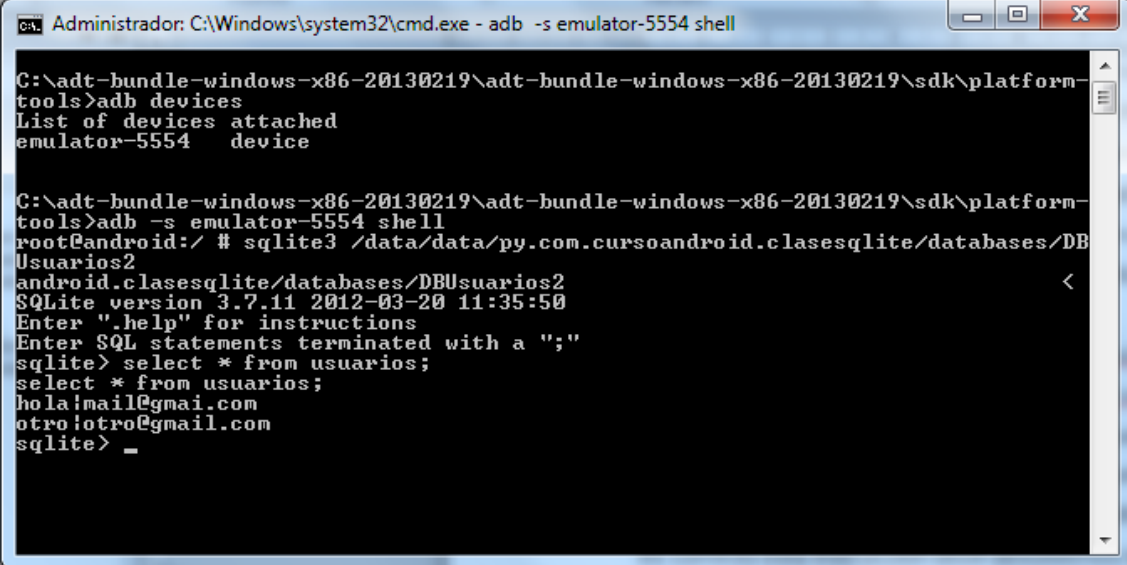
Con esto ya comprobamos al menos que el fichero de nuestra base de datos se ha creado en la ruta correcta. Ya sólo nos queda comprobar que tanto las tablas creadas como los datos insertados también se han incluido correctamente en la base de datos. Para ello podemos recurrir a dos posibles métodos:

1. Transferir la base de datos a nuestro PC y consultarla con cualquier administrador de bases de datos SQLite.
2. Acceder directamente a la consola de comandos del emulador de Android y utilizar los comandos existentes para acceder y consultar la base de datos SQLite.

El primero de los métodos es sencillo. El fichero de la base de datos podemos transferirlo a nuestra PC utilizando el botón de descarga situado en la esquina superior derecha del explorador de archivos (remarcado en rojo en la imagen anterior). Junto a este botón aparecen otros dos para hacer la operación contraria (copiar un fichero local al sistema de archivos del emulador) y para eliminar ficheros del emulador. Una vez descargado el fichero a nuestro sistema local, podemos utilizar cualquier administrador de SQLite para abrir y consultar la base de datos, por ejemplo SQLite Administrator (freeware).

El segundo método utiliza una estrategia diferente. En vez de descargar la base de datos a nuestro sistema local, somos nosotros los que accedemos de forma remota al emulador a través de su consola de comandos (*shell*). Para ello, con el emulador de Android aún abierto, debemos abrir una consola de MS-DOS y utilizar la utilidad `adb.exe` (*Android Debug Bridge*) situada en la carpeta `platform-tools` del SDK de Android (en mi caso: `c:\android-sdk-windows\platform-tools\`). En primer lugar consultaremos los identificadores de todos los emuladores en ejecución mediante el comando “`adb devices`”. Esto nos debe devolver una única instancia si sólo tenemos un emulador abierto, que en mi caso particular se llama “`emulator-5554`”.

Tras conocer el identificador de nuestro emulador, vamos a acceder a su shell mediante el comando `adb -s identificador-del-emulador shell`. Una vez conectados, ya podemos acceder a nuestra base de datos utilizando el comando `sqlite3` pasándole la ruta del fichero, para nuestro ejemplo `sqlite3 /data/data/py.com.cursoandroid/databases/DBUsuarios`. Si todo ha ido bien, debe aparecernos el *prompt* de SQLite `sqlite>`, lo que nos indicará que ya podemos escribir las consultas SQL necesarias sobre nuestra base de datos. Nosotros vamos a comprobar que existe la tabla `Usuarios` y que se han insertado los cinco registros de ejemplo. Para ello haremos la siguiente consulta: `"SELECT * FROM Usuarios;"`. Si todo es correcto esta instrucción debe devolvernos los cinco usuarios existentes en la tabla. En la imagen siguiente se muestra todo el proceso descrito:



```
Administrator: C:\Windows\system32\cmd.exe - adb -s emulator-5554 shell

C:\adt-bundle-windows-x86-20130219\adt-bundle-windows-x86-20130219\sdk\platform-tools>adb devices
List of devices attached
emulator-5554    device

C:\adt-bundle-windows-x86-20130219\adt-bundle-windows-x86-20130219\sdk\platform-tools>adb -s emulator-5554 shell
root@android:/ # sqlite3 /data/data/py.com.cursoandroid.clasesqlite/databases/DBUsuarios2
android.clasesqlite/databases/DBUsuarios2
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> select * from usuarios;
select * from usuarios;
hola!mail@gmail.com
otro!otro@gmail.com
sqlite> _
```

Con esto ya hemos comprobado que nuestra base de datos se ha creado correctamente, que se han insertado todos los registros de ejemplo y que todo funciona según se espera.

Insertar/Actualizar/Eliminar

La API de SQLite de Android proporciona dos alternativas para realizar operaciones sobre la base de datos que no devuelven resultados (entre ellas la inserción/actualización/eliminación de registros, pero también la creación de tablas, de índices, etc).

El primero de ellos, que ya comentamos brevemente en el artículo anterior, es el método `execSQL()` de la clase `SQLiteDatabase`. Este método permite ejecutar cualquier sentencia SQL sobre la base de datos, siempre que ésta no devuelva resultados. Para ello, simplemente aportaremos como parámetro de entrada de este método la cadena de texto correspondiente con la sentencia SQL. Cuando creamos la base de datos en el anterior ejemplo ya vimos algunas formas para insertar los registros de prueba. Otros ejemplos podrían ser los siguientes:

```

2  db.execSQL("INSERT INTO Usuarios (usuario,email) VALUES ('usu1','usu1@email.com')
3
4  //Eliminar un registro
5  db.execSQL("DELETE FROM Usuarios WHERE usuario='usu1' ");
6
7  //Actualizar un registro
8  db.execSQL("UPDATE Usuarios SET email='nuevo@email.com' WHERE usuario='usu1' ");

```

La segunda de las alternativas disponibles en la API de Android es utilizar los métodos `insert()`, `update()` y `delete()` proporcionados también con la clase `SQLiteDatabase`. Estos métodos permiten realizar las tareas de inserción, actualización y eliminación de registros de una forma algo más paramétrica que `execSQL()`, separando tablas, valores y condiciones en parámetros independientes de estos métodos.

Empecemos por el método `insert()` para insertar nuevos registros en la base de datos. Este método recibe tres parámetros, el primero de ellos será el nombre de la tabla, el tercero serán los valores del registro a insertar, y el segundo lo obviaremos por el momento ya que tan sólo se hace necesario en casos muy puntuales (por ejemplo para poder insertar registros completamente vacíos), en cualquier otro caso pasaremos con valor `null` este segundo parámetro.

Los valores a insertar los pasaremos como elementos de una colección de tipo `ContentValues`. Esta colección es de tipo *diccionario*, donde almacenaremos parejas de *clave-valor*, donde la *clave* será el nombre de cada campo y el *valor* será el dato correspondiente a insertar en dicho campo. Veamos un ejemplo:

```

1  //Creamos el registro a insertar como objeto ContentValues
2  ContentValues nuevoRegistro = new ContentValues();
3  nuevoRegistro.put("usuario", "usu10");
4  nuevoRegistro.put("email", "usu10@email.com");
5
6  //Insertamos el registro en la base de datos
7  db.insert("Usuarios", null, nuevoRegistro);

```

Los métodos `update()` y `delete()` se utilizarán de forma muy parecida a ésta, con la salvedad de que recibirán un parámetro adicional con la condición *WHERE* de la sentencia SQL. Por ejemplo, para actualizar el email del usuario de nombre 'usu1' haríamos lo siguiente:

```

1  //Establecemos los campos-valores a actualizar
2  ContentValues valores = new ContentValues();
3  valores.put("email", "usu1_nuevo@email.com");
4
5  //Actualizamos el registro en la base de datos
6  db.update("Usuarios", valores, "usuario='usu1'");

```

Como podemos ver, como tercer parámetro del método `update()` pasamos directamente la condición del *UPDATE* tal como lo haríamos en la cláusula *WHERE* en una sentencia SQL normal.

El método `delete()` se utilizaría de forma análoga. Por ejemplo para eliminar el registro del usuario 'usu2' haríamos lo siguiente:

```
1 //Eliminamos el registro del usuario 'usu2'
2 db.delete("Usuarios", "usuario='usu2'");
```

Como vemos, volvemos a pasar como primer parámetro el nombre de la tabla y en segundo lugar la condición *WHERE*. Por supuesto, si no necesitáramos ninguna condición, podríamos dejar como `null` en este parámetro.

Un último detalle sobre estos métodos. Tanto en el caso de `execSQL()` como en los casos de `update()` o `delete()` podemos utilizar argumentos dentro de las condiciones de la sentencia SQL. Esto no son más que partes *variables* de la sentencia SQL que aportaremos en un array de valores aparte, lo que nos evitará pasar por la situación típica en la que tenemos que construir una sentencia SQL concatenando cadenas de texto y variables para formar el comando SQL final. Estos argumentos SQL se indicarán con el símbolo '?', y los valores de dichos argumentos deben pasarse en el array en el mismo orden que aparecen en la sentencia SQL. Así, por ejemplo, podemos escribir instrucciones como la siguiente:

```
1 //Eliminar un registro con execSQL(), utilizando argumentos
2 String[] args = new String[]{"usu1"};
3 db.execSQL("DELETE FROM Usuarios WHERE usuario=?", args);
4
5 //Actualizar dos registros con update(), utilizando argumentos
6 ContentValues valores = new ContentValues();
7 valores.put("email", "usu1_nuevo@email.com");
8
9 String[] args = new String[]{"usu1", "usu2"};
10 db.update("Usuarios", valores, "usuario=? OR usuario=?", args);
```

Esta forma de pasar a la sentencia SQL determinados datos variables puede ayudarnos además a escribir código más limpio y evitar posibles errores.

Consultar/Recuperar registros

De forma análoga a lo que vimos para las sentencias de modificación de datos, vamos a tener dos opciones principales para recuperar registros de una base de datos SQLite en Android. La primera de ellas utilizando directamente un comando de selección SQL, y como segunda opción utilizando un método específico donde *parametrizaremos* la consulta a la base de datos.

Para la primera opción utilizaremos el método `rawQuery()` de la clase `SQLiteDatabase`. Este método recibe directamente como parámetro un comando SQL completo, donde indicamos los campos a recuperar y los criterios de selección. El resultado de la consulta lo obtendremos en forma de cursor, que posteriormente podremos

recorrer para procesar los registros recuperados. Sirva la siguiente consulta a modo de ejemplo:

```
1 Cursor c = db.rawQuery(" SELECT usuario,email FROM Usuarios
2 WHERE usuario='usul' ");
```

Como en el caso de los métodos de modificación de datos, también podemos añadir a este método una lista de argumentos variables que hayamos indicado en el comando SQL con el símbolo '?', por ejemplo así:

```
1 String[] args = new String[] {"usul"};
2 Cursor c = db.rawQuery(" SELECT usuario,email FROM Usuarios
3 WHERE usuario=? ", args);
```

Más adelante veremos cómo podemos manipular el objeto `Cursor` para recuperar los datos obtenidos.

Como segunda opción para recuperar datos podemos utilizar el método `query()` de la clase `SQLiteDatabase`. Este método recibe varios parámetros: el nombre de la tabla, un array con los nombre de campos a recuperar, la cláusula *WHERE*, un array con los argumentos variables incluidos en el *WHERE* (si los hay, `null` en caso contrario), la cláusula *GROUP BY* si existe, la cláusula *HAVING* si existe, y por último la cláusula *ORDER BY* si existe. Opcionalmente, se puede incluir un parámetro al final más indicando el número máximo de registros que queremos que nos devuelva la consulta. Veamos el mismo ejemplo anterior utilizando el método `query()`:

```
1 String[] campos = new String[] {"usuario", "email"};
2 String[] args = new String[] {"usul"};
3
4 Cursor c = db.query("Usuarios", campos, "usuario=?", args, null, null, null);
```

Como vemos, los resultados se devuelven nuevamente en un objeto `Cursor` que deberemos recorrer para procesar los datos obtenidos.

Para recorrer y manipular el cursor devuelto por cualquiera de los dos métodos mencionados tenemos a nuestra disposición varios métodos de la clase `Cursor`, entre los que destacamos dos de los dedicados a recorrer el cursor de forma secuencial y en orden natural:

- `moveToFirst()`: mueve el puntero del cursor al primer registro devuelto.
- `moveToNext()`: mueve el puntero del cursor al siguiente registro devuelto.

Los métodos `moveToFirst()` y `moveToNext()` devuelven `TRUE` en caso de haber realizado el movimiento correspondiente del puntero sin errores, es decir, siempre que exista un primer registro o un registro siguiente, respectivamente.

Una vez posicionados en cada registro podremos utilizar cualquiera de los métodos `getXXX(índice_columna)` existentes para cada tipo de dato para recuperar el dato de cada campo del registro actual del cursor. Así, si queremos recuperar por ejemplo la segunda columna del registro actual, y ésta contiene un campo alfanumérico, haremos la

llamada `getString(1)` [NOTA: los índices comienzan por 0, por lo que la segunda columna tiene índice 1], en caso de contener un dato de tipo real llamaríamos a `getDouble(1)`, y de forma análoga para todos los tipos de datos existentes. Con todo esto en cuenta, veamos cómo podríamos recorrer el cursor devuelto por el ejemplo anterior:

```
1  String[] campos = new String[] {"usuario", "email"};
2  String[] args = new String[] {"usu1"};
3
4  Cursor c = db.query("Usuarios", campos, "usuario=?", args, null, null, null);
5
6  //Nos aseguramos de que existe al menos un registro
7  if (c.moveToFirst()) {
8      //Recorremos el cursor hasta que no haya más registros
9      do {
10         String usuario = c.getString(0);
11         String email = c.getString(1);
12     } while(c.moveToNext());
13 }
```

Además de los métodos comentados de la clase `Cursor` existen muchos más que nos pueden ser útiles en muchas ocasiones. Por ejemplo, `getCount()` te dirá el número total de registros devueltos en el cursor, `getColumnName(i)` devuelve el nombre de la columna con índice `i`, `moveToPosition(i)` mueve el puntero del cursor al registro con índice `i`, etc. Se puede consultar la lista completa de métodos disponibles en la clase `Cursor` en la documentación oficial de Android.