2015

Bases de datos SQLite en Android

CREACIÓN Y UTILIZACIÓN DESDE LA APLICACIÓN ANDROID; CREAR Y COPIAR UNA BASE DE DATOS EXTERNA A LA APLICACIÓN ANDROID

SERGIO DIEZ GARCÍA-OLALLA

Índice

Introducción.

SQLite es un sistema gestor de base de datos relacional (RDBMS). Lo que hace único a SQLite es que se considera una solución embebida, no requiriendo ni instalarla ni administrarla.

La mayoría de los sistemas de gestión de bases de datos como Oracle, MySQL, y SQL Server son procesos de servidor autónomos que se ejecutan independientemente. SQLite es en realidad una librería que está enlazada dentro de las aplicaciones. Todas las operaciones de base de datos se manejan dentro de la aplicación mediante llamadas y funciones contenidas en la librería SQLite.

A su vez cada base de datos SQLite de las diferentes aplicaciones serán totalmente independientes (aporta seguridad) unas de otras. Sus datos sólo serán accesibles desde la aplicación que la creó y para poder compartir datos usaremos los Content Providers.

Como curiosidad indicar que SQLite está realmente escrito en C y está contenido en un "envoltorio" de Java que proporciona el SDK de Android.

Gracias a la API de Android solo se necesitará mediante consultas SQL dar las órdenes necesarias en función de tus datos para crear o cambiar la versión de tu base de datos ya que por detrás Android gestiona todo automáticamente.

Acceder a la base de datos implica acceder al sistema de archivos, eso puede hacer según el dispositivo que sea algo más lenta la conexión y consulta.

SQLite soporta los tipos de datos TEXT (similar al String de Java), INTEGER (similar al long) y REAL (similar al double). Todos los demás pueden ser convertidos a uno de estos antes de ser guardados en la base de datos. En sí,SQLite, no valida si los tipos de datos de las columnas son en realidad del tipo definido, por ejemplo, se puede escribir un número entero en una columna de String y viceversa.

Creación y utilización de una base de datos SQLite.

Creación.

En Android, la forma típica para crear, actualizar, y conectar con una base de datos SQLite será la codificación de una clase que derive de la clase **SQLiteOpenHelper** 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.

Una vez definida nuestra clase helper, abrir la base de datos desde nuestra aplicación será algo de lo más sencillo.

Lo primero será crear un objeto de la clase helper que hemos definido, a su constructor le pasaremos los siguientes parámetros:

1. El *contexto de la aplicación*.
2. El *nombre de la base de datos*.
3. Un *objeto CursorFactory* que normalmente no será necesario (en ese caso pasaremos el valor null).
   Se utiliza como clase auxiliar en la que se suelen implementar validaciones extra u operaciones sobre las consultas a la base de datos
4. Por último la *versión de la base de datos* que necesitamos.

La creación de este objeto puede tener varios efectos, según la situación previa de la aplicación:

- 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 llegados a este punto hemos conseguido conectar a nuestra base de datos. La manipulación de los datos la explico en el siguiente punto, antes de eso pongo la referencia a los métodos abstractos mencionados anteriormente.

Método onCreate(SQLiteDatabase db)

Será ejecutado automáticamente por nuestra clase helper cuando sea necesaria la creación de la base de datos, es decir, cuando aún no exista. En este método lo que se hará será crear todas las tablas necesarias y la inserción de los datos iniciales si son necesarios.

En nuestro caso, sólo vamos a crear la tabla descrita anteriormente. Para la creación de la tabla ejecutaremos sobre el objeto SQLiteDatabase el método execSQL() que debe recibir como parámetro el String con la sentencia SQL válida en SQLite.

Método onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)

Se lanzará automáticamente cuando sea necesaria una actualización de la estructura de la base de datos o una conversión de los datos.

Utilización o manipulación de datos.

Una vez tenemos una referencia al objeto helper que hemos codificado deberemos llamar a uno de los métodos que comentaré en breve para poder empezar a manipular los datos. Estos métodos nos devuelven un objeto de la clase SQLiteDatabase sobre el que ya podremos ejecutar sentencias SQL de consulta, inserción, borrado, etc.

También es importante indicar que una vez hayamos acabado de utilizar la base de datos es aconsejable cerrar la conexión con ella llamando al método close() del objeto.

Los métodos a utilizar sobre el objeto helper son:

getReadableDatabase() el cual usaremos si sólo necesitamos consultar los datos (Read) o, **getWritableDatabase()** si necesitamos consultar y realizar modificaciones (Read & Write).

Operaciones que no devuelven resultados.

Ahora vamos a pasar a las operaciones sobre la base de datos que no devuelven resultados entre ellas la inserción/actualización/eliminación de registros, existen la creación de tablas, de índices, etc). Me voy a centrar en los de manipulación de los datos sobre una tabla ya creada, para la creación de tablas, índices, vistas y otras opciones habría que consultar la sentencia SQL apropiada para SQLite y hacer que se ejecute en la base de datos con el método apropiado de existir o con **execSQL()** pasándole como parámetro la sentencia.

La API de SQLite de Android proporciona dos alternativas para realizar estas consultas:

Utilizando el método execSQL(String consulta)

Este método permite ejecutar cualquier sentencia SQL sobre la base de datos, siempre que ésta no devuelva resultados. Para ello, le indicaremos como parámetro de entrada la cadena de texto correspondiente con la sentencia SQL.

Podremos insertar un registro así:

```
objSQLDatabase.execSQL("INSERT INTO medicamentos (codigo, nombre, comprimidos)
VALUES (6,'Paracetamol', 20)");
```

Eliminarlo usando:

```
objSQLDatabase.execSQL("DELETE FROM medicamentos WHERE nombre='Paracetamol'");
```

Y actualizarlo:

```
objSQLDatabase.execSQL("UPDATE medicamentos SET comprimidos= 30 WHERE nombre =
'Paracetamol'");
```

Utilizando los métodos insert(), delete() y update() de la clase SQLiteDatabase

Estos métodos permiten realizar las tareas de inserción, actualización y eliminación de registros mediante el uso de parámetros separando en ellos los datos necesarios para su funcionamiento como son tablas, campos (como nombre o comprimidos) y condiciones (WHERE, HAVING, GROUP BY).

Veamos que necesita cada uno:

1. Método **insert(String table, String nullColumnHack, ContentValues values).**

   Este método recibe *tres parámetros*:

   El primero de ellos será el nombre de la tabla.

   El segundo sirve para poder insertar registros completamente vacíos, en cualquier otro caso pasaremos con valor null este segundo parámetro.

   El tercero serán los valores del registro a insertar, en el orden de los campos de la tabla.

   Ejemplo de uso:

   Los valores a insertar los pasaremos como elementos de una colección de tipo CotentValues.

```
ContentValues nuevoRegistro = new ContentValues();
nuevoRegistro.put("codigo", "6");
nuevoRegistro.put("nombre","Aspirina");
nuevoRegistro.put("comprimidos","20");
Y para insertar el registro en la base de datos
objSQLiteDatabase.insert("medicamentos", null, nuevoRegistro);
```

2. Método **delete(String table, String whereClause, String[] whereArgs)**

   El método delete() se utilizaría de forma análoga. El tercer parámetro es para proporcionar argumentos mediante un array de cadenas.

   Ejemplo sin argumentos:

```
objSQLiteDatabase.delete("medicamentos", "nombre = ’Paracetamol’", null);
```

   Ejemplo usando argumentos:

```
String[] args = new String[]{"’Paracetamol’", "’Aspirina’"};
objSQLiteDatabase.delete("medicamentos", "nombre =?", args);
```

3. Método **update(String table, ContentValues values, String whereClause, String[] whereArgs)**

Recibe un par de parámetros adicionales con la condición WHERE de la sentencia SQL y el último que se usa en el caso de usarse argumentos en la condición WHERE.

Ejemplo sin argumentos:

```
ContentValues valores = new ContentValues();
valores.put("comprimidos","30");

objSQLiteDatabase.update("Medicamentos", valores, "nombre = ’Paracetamol’", null);
```

Ejemplo con argumentos:

```
ContentValues valores = new ContentValues();
valores.put("comprimidos","30");

String[] args = new String[]{"’Paracetamol’", " ’Aspirina’"};

objSQLiteDatabase.update("medicamentos", valores, "nombre=? OR nombre=?", args);
```

Operaciones que devuelven resultados.

Este tipo de operaciones son las conocidas como consultas que nos permiten recuperar ciertos datos que nos puedan resultar de interés para alguna operación a realizar con ellos accediendo a un conjunto de ellos o a algún campo en concreto con el que realizar operaciones numéricas o de otros tipos.

Análogamente a lo visto previamente la API de Android nos permite realizar estas operaciones de dos formas:

Utilizando el método rawQuery(String sql, String[] selectionArgs)

Este método recibe directamente como parámetro un comando SQL completo, donde indicamos los campos a recuperar y los criterios de selección (WHERE, GROUP BY, o el que necesitemos). El segundo parámetro sería para los argumentos en caso de necesitar usarlos en la consulta o tener que recibir el dato del criterio de selección de una fuente externa (Tener que calcularlo o que sea el usuario quien lo introduzca).

El resultado de la consulta lo obtendremos en forma de cursor, que posteriormente podremos recorrer para procesar los registros recuperados.

En mi aplicación de ejemplo es usado de la siguiente forma:

```
String sqlMedicamento = "select codigo, nombre, comprimidos from medicamentos";

Cursor c = dataBase.rawQuery(sqlMedicamento, null);
```

El uso de argumentos es análogo a lo visto previamente.

A posteriori habrá que realizar operaciones para recorrer los resultados del cursor con el uso de sus métodos:

- **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 siempre que puedan posicionarse en el elemento del cursor, ya sea porque exista un resultado en la consulta o mientras siga habiendo registros.

Una vez posicionados en cada registro devuelto 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 (cero), 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.

getFloat (int columnIndex) Returns the value of the requested column as a float.
getInt (int columnIndex) Returns the value of the requested column as an int.
getLong (int columnIndex) Returns the value of the requested column as a long.
getShort (int columnIndex) Returns the value of the requested column as a short.
getString (int columnIndex) Returns the value of the requested column as a String.
getType (int columnIndex) Returns data type of the given column's value.

Utilizando el método query (String table, String[] columns, String where, String[] whereArgs, String groupBy, String having, String orderBy)

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 un ejemplo:

```
String[] campos = new String[] {"codigo", "nombre"};
String[] args = new String[] {"'Paracetamol'"};

Cursor c = db.query("medicamentos", campos, "nombre=?", args, null, null, null);
```

Acceso a base de datos desde el emulador

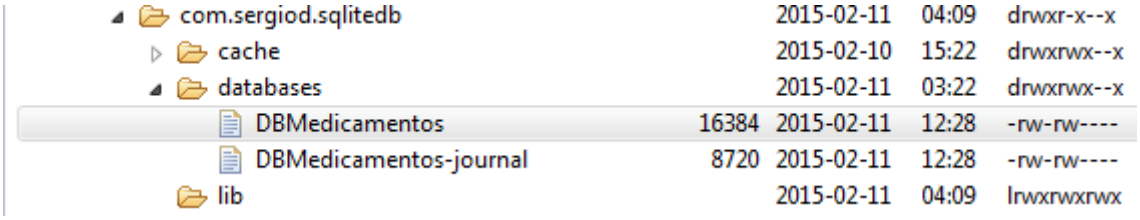
En primer lugar veamos dónde se ha creado nuestra base de datos. Todas las bases de datos SQLite creadas por aplicaciones Android utilizando este método se almacenan en la memoria del teléfono en un fichero con el mismo nombre de la base de datos situado en una ruta que sigue el siguiente patrón:

/data/data/<NOMBRE_PAQUETE_JAVA>/databases/<NOMBRE_BASE_DATOS>

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

/data/data/com.sergiod.sqlitedb/databases/DBMedicamentos

Para comprobar esto 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.

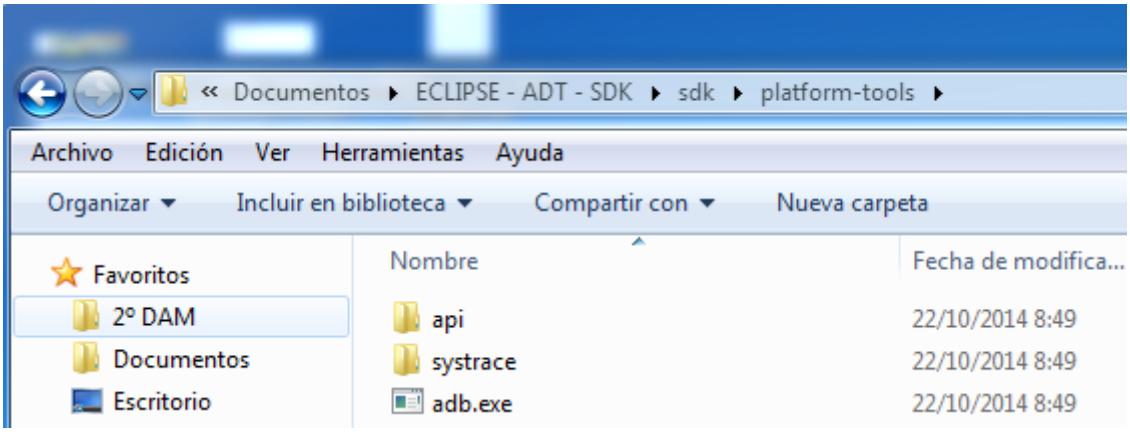


Con esto vemos que se ha creado pero no si los datos estan guardándose en ella correctamente, para comprobar esto último podemos usar varios métodos yo voy a hacerlo desde la consola de comandos (o shell) del emulador de Android y utilizar los comandos existentes para acceder y consultar la base de datos SQLite.

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.

ATENCIÓN: En el caso de nuestros equipos usamos un eclipse con ADT y SDK portable que cada uno habrá puesto en una carpeta de su PC por lo que yo lo voy a hacer para el mío pero en vuestros casos para probar esto deberéis usar la ruta del vuestro. Mi ruta es:

D:\Documentos\ECLIPSE - ADT - SDK\sdk\platform-tools



Por lo que abro el CMD y me situo en esa carpeta mediante el comando cd + ruta (previamente cambio de partición poniendo "D:" + INTRO)

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 se llama "emulator-5554".

```
D:\Documentos\ECLIPSE - ADT - SDK\sdk\platform-tools>adb.exe devices
List of devices attached
emulator-5554   device

D:\Documentos\ECLIPSE - ADT - SDK\sdk\platform-tools>
```

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/com.sergiod.sqlitedb/databases/DBMedicamentos".

Si todo ha ido bien, debe aparecernos el prompt de SQLite "sqlite>"

```
D:\Documentos\ECLIPSE - ADT - SDK\sdk\platform-tools>adb -s emulator-5554 shell
root@generic:/ # sqlite3 /data/data/com.sergiod.sqlitedb/databases/DBMedicamento
s
sqlitedb/databases/DBMedicamentos                                          <
SQLite version 3.7.11 2012-03-20 11:35:50
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

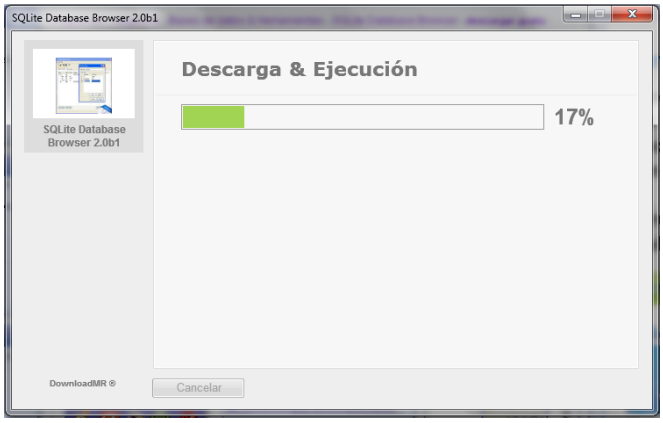
De ser así ya podemos escribir las consultas SQL necesarias sobre nuestra base de datos. Nosotros vamos a comprobar que existe la tabla medicamentos y que tiene insertados registros, para ello haremos la siguiente consulta: "select * from medicamentos;".

```
sqlite> select * from medicamentos;
select * from medicamentos;
1|Medicamento 1|16
2|Medicamento 2|22
3|Medicamento 3|15
4|Medicamento 4|16
5|Medicamento 5|37
sqlite>
```

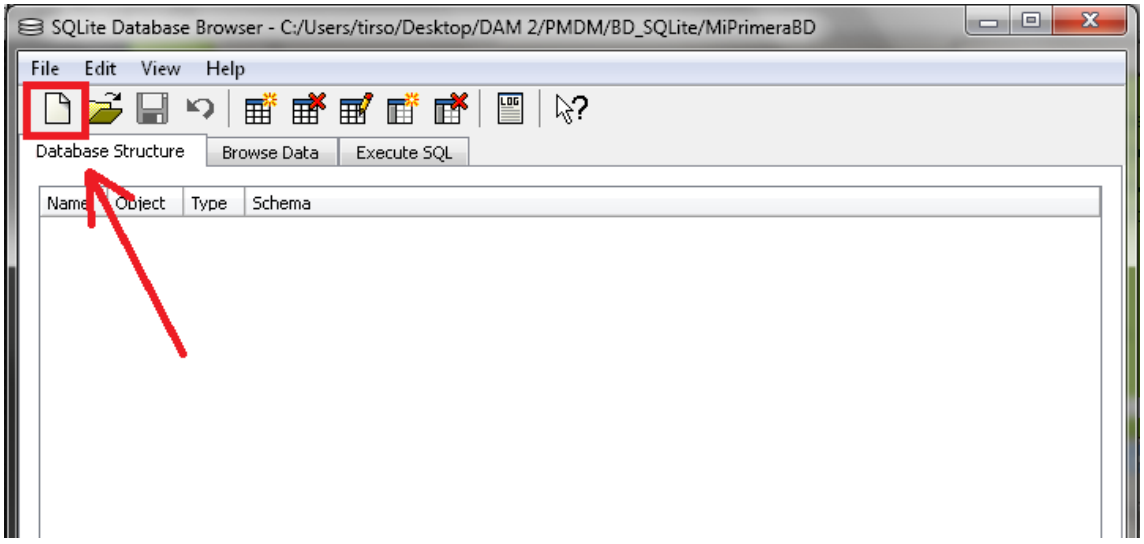
Creación de una base de datos externa y traslado para su uso en SQLite.

Creación.

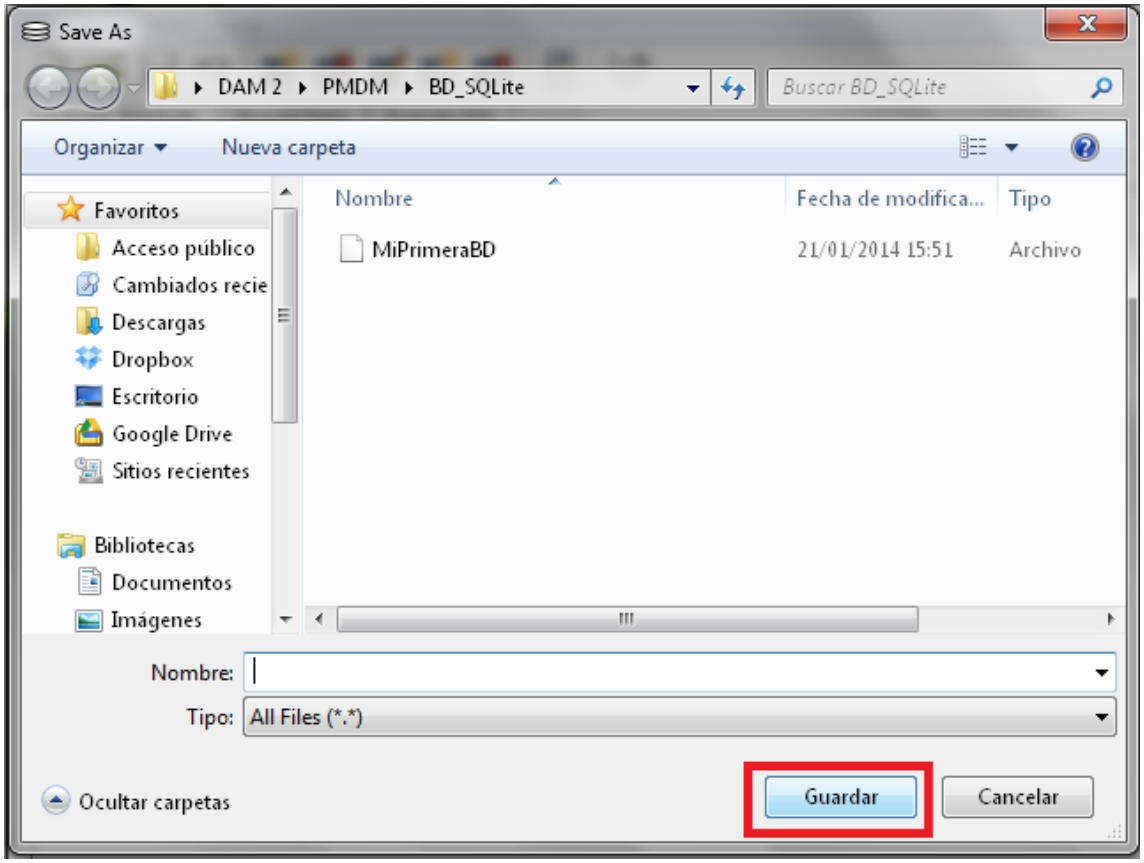
Lo primero que necesitamos es un software de código abierto SQLite Database Browser (disponible para Windows y Linux).



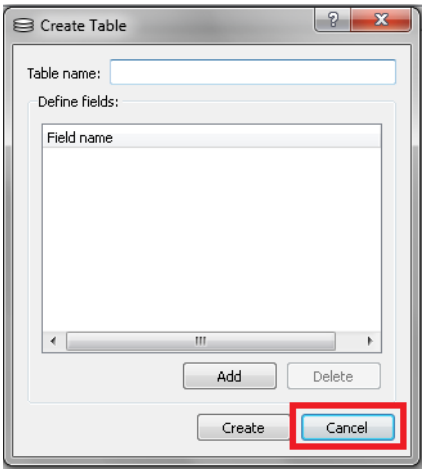
Terminada la descarga, ejecutamos el fichero SQLite Database Browser.



Pulsa sobre nuevo, te preguntara donde quieres guardar la base de datos, selecciona tu ruta (una que te sea sencilla acceder, porque luego tendremos que copiarla a nuestra aplicación de Android en Eclipse) y pulsa sobre Guardar.



Saldrá un cuadro de diálogo preguntándote si quieres crear una tabla, pulsa Cancelar.

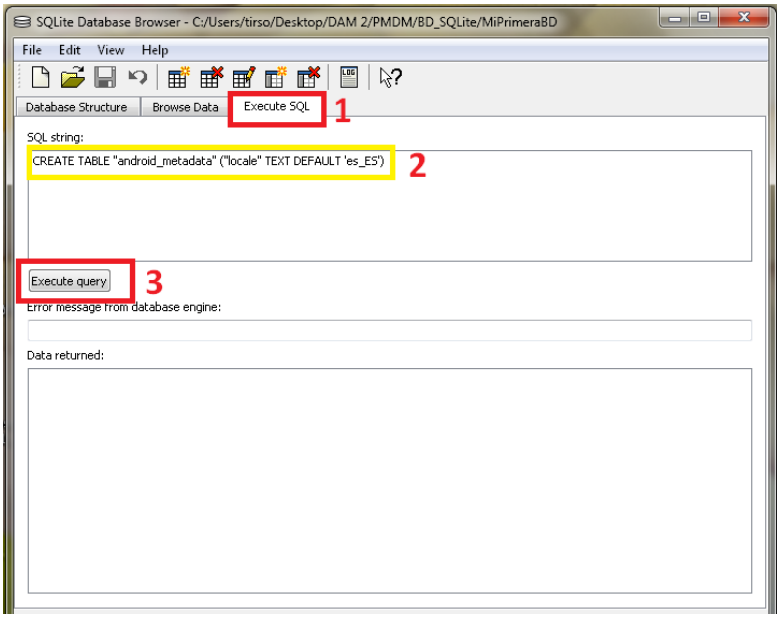


Nuestra base de datos SQLite para Android tiene que cumplir unos requisitos para que nuestra aplicación la pueda utilizar.

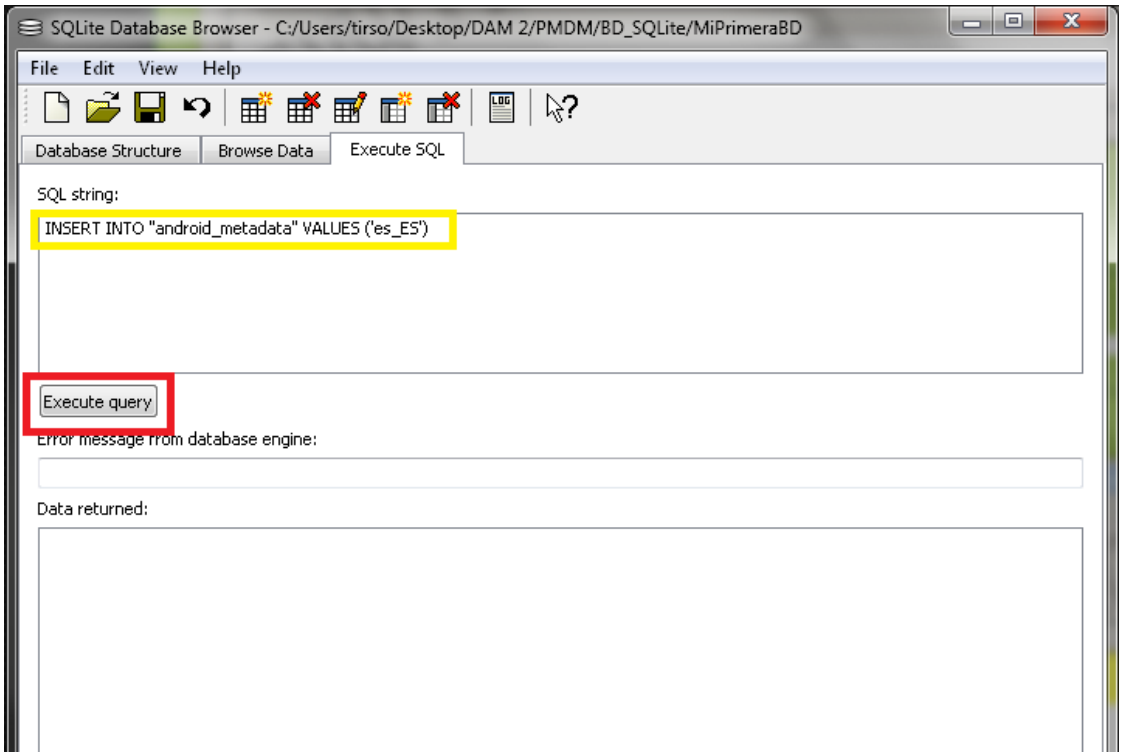
La base de datos debe tener una tabla llamada android_metadata que tenga una sola columna "locale" de tipo TEXT. Dicha tabla debe tener al menos un valor, por ejemplo es_ES.

Pulsa sobre la pestaña "Execute SQL", escribe la siguiente instrucción SQL: CREATE TABLE "android_metadata" ("locale" TEXT DEFAULT 'es_ES') y después pulsa sobre el botón "Execute query".

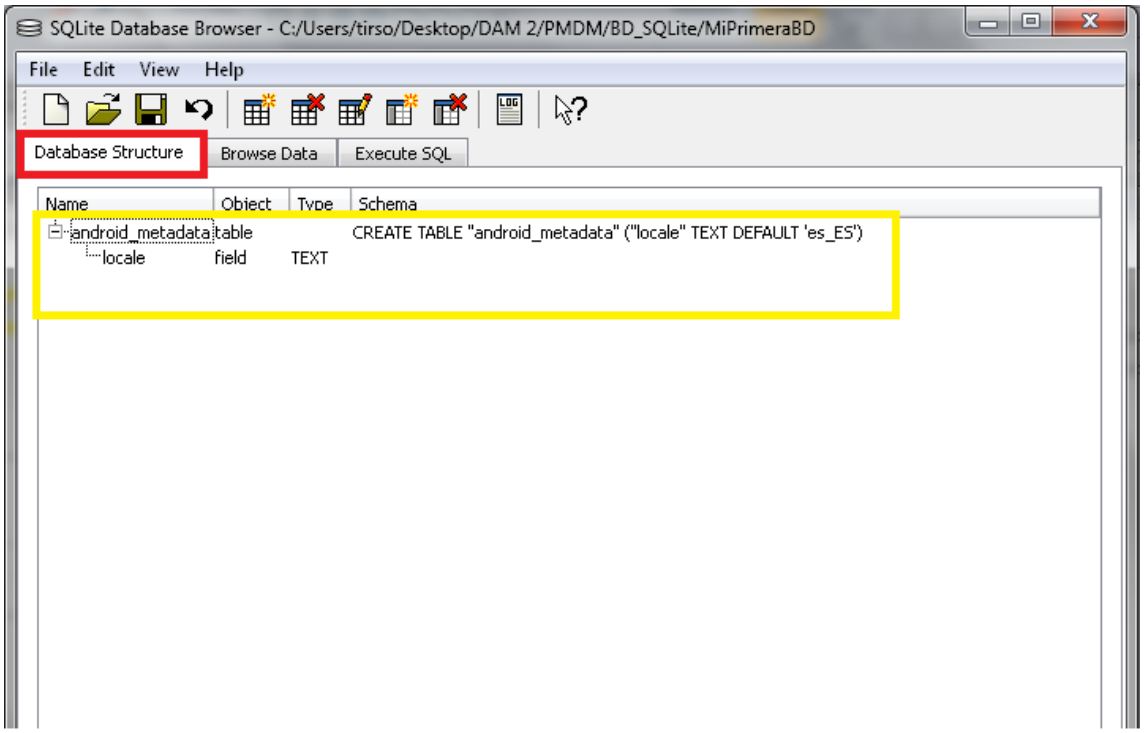
Si todo ha salido bien, verás un mensaje de “No error”.



Ahora escribe INSERT INTO "android_metadata" VALUES ('es_ES') para crear un registro con el dato ‘es_ES’ en la tabla recién creada.



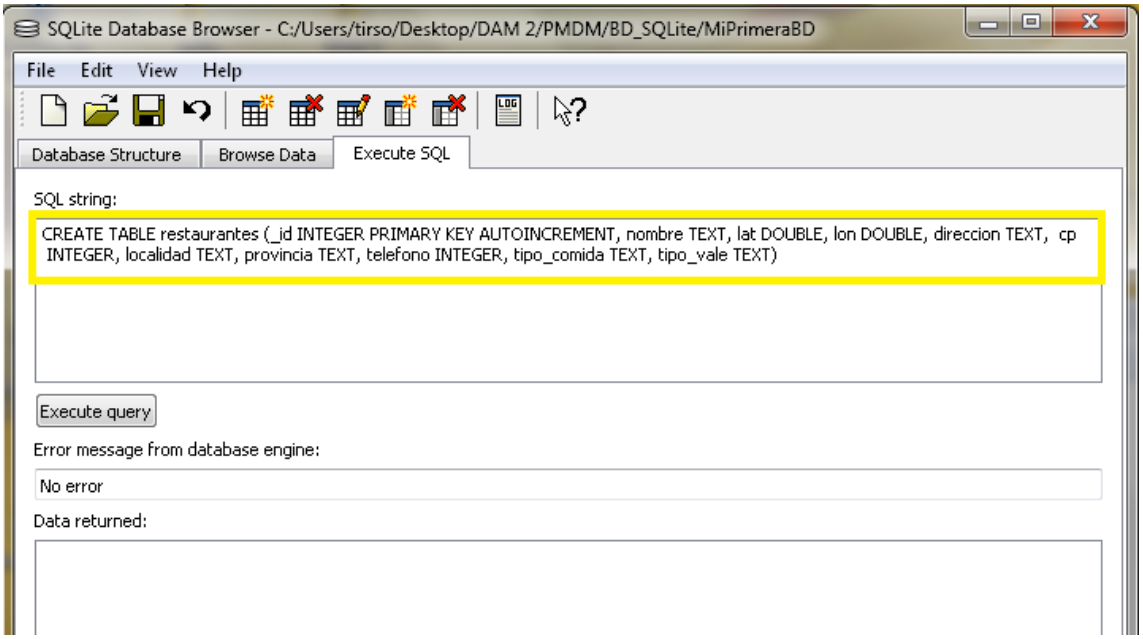
Vuelve a la pestaña de “Database Structure” para comprobar que este la tabla, fíjate que aquí aparecen las tablas, y si le damos al + aparecen los campos.



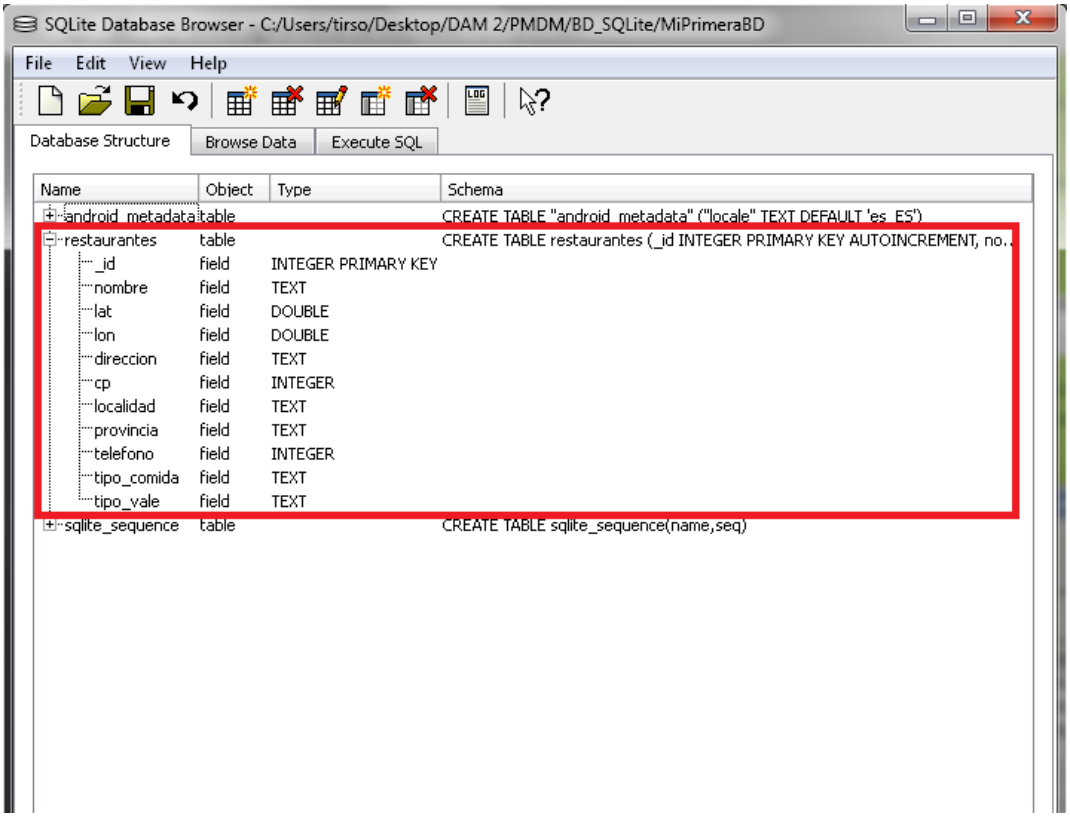
Ahora debes de crear otra tabla con el nombre que quieras para introducir los datos que necesites, para esto puedes usar otra sentencia de SQL o utilizar el asistente que tiene la aplicación (los iconos superiores de la aplicación).

Todas las tablas que crees en la BD (a excepción de android_metadata) deben tener la misma clave primaria:

- Nombre del campo: _id
- Tipo de campo: INTEGER PRIMARY KEY AUTOINCREMENT



Es posible que la propia aplicación te añada una tercera tabla llamada “sqlite_sequence”, en ella se guardan las transacciones que se van haciendo en la BD. NO LA BORRES.



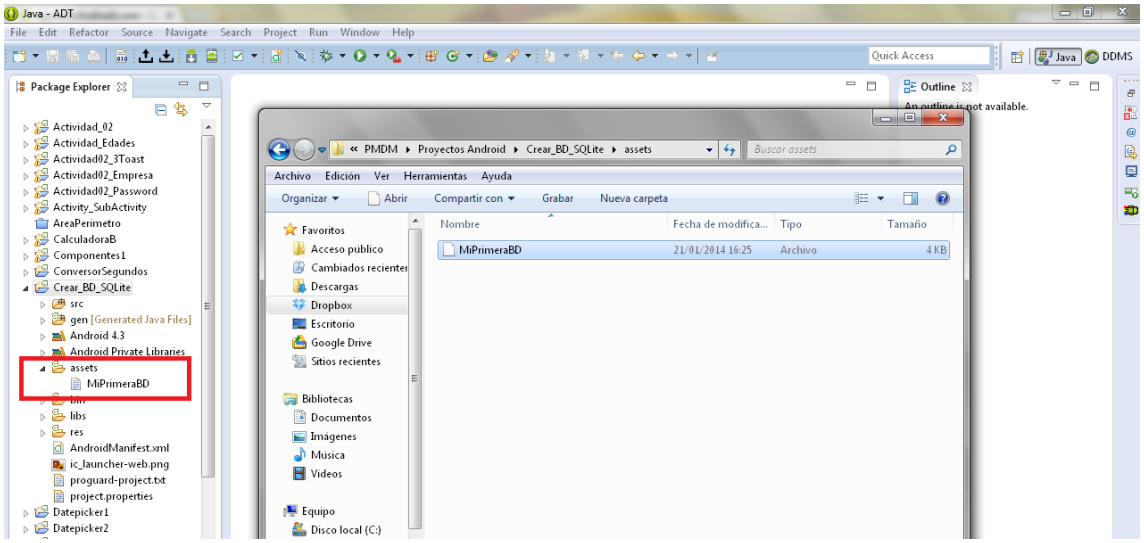
Con esto habrás creado la base de datos, ahora toca pasarla a nuestra aplicación.

Traslado de datos a SQLite.

Lo primero es copiar el fichero resultante de crear nuestra BD a nuestro proyecto de Eclipse. Coge el fichero y arrástralo desde el explorador de Windows hasta la carpeta “assets” de tu proyecto. Si no existe esta carpeta la tendrás que crear primero. Has de asegurarte que tu BD no ocupa más de 1Mb, ya que Android tiene una limitación para copiar ficheros de más de 1Mb desde el directorio “assets” o “raw”. Si la BD es de más peso, hay que “partir” el fichero resultante en fragmentos de 1Mb para luego recomponerla en nuestra aplicación.

Distinguiremos los 2 casos si es inferior a 1 Mb (directamente se guarda en assests) y si es superior (sus partes irán al directorio raw) y que desarrollaremos a continuación.

Base de datos inferior a 1 Mb



Para copiar una BD SQLite se necesita una clase auxiliar llamada que derive de SQLiteOpenHelper y que debemos personalizar para adaptarnos a las necesidades concretas de nuestra aplicación.

Lo primero que hacemos es crear dos variables:

- **DB_PATH**: Es la ruta que tienen las bases de datos. Todas las bases de datos de cada aplicación se guardan en esa ruta.

private static final String DB_PATH = "/data/data/nombreDelPaquete/databases/";

- **DB_NAME**: Es el nombre de la BD de tu aplicación, es decir, el nombre que utilizarás en tu aplicación para acceder a la BD.

private static final String DB_NAME = “nombreBD.db”;

```
1  public class DBHelper extends SQLiteOpenHelper{
2
3  //Ruta por defecto de las bases de datos en el sistema Android
4  private static String DB_PATH = "/data/data/TU_PAQUETE/databases/";
5
6  private static String DB_NAME = "filename.db";
7
8  private SQLiteDatabase myDataBase;
9
10 private final Context myContext;
11
12 /**
13 * Constructor
14 * Toma referencia hacia el contexto de la aplicación que lo invoca para poder acc
15 * Crea un objeto DBOpenHelper que nos permitirá controlar la apertura de la base
16 * @param context
17 */
18 public DBHelper(Context context) {
19
20 super(context, DB_NAME, null, 1);
21 this.myContext = context;
22
23 }
```

Para copiar el contenido de la BD externa, primero hay que crear una BD vacía en el DB_PATH que hemos proporcionado al principio. Mediante una variable de tipo boolean se comprueba si ya existe esa BD. En caso de que no exista, se llama al método getReadableDatabase() que crea la BD vacía, para sobrescribirla con la BD externa mediante el método copyDataBase(), definido más abajo.

```
/**
* Crea una base de datos vacía en el sistema y la reescribe con nuestro fichero d
* */
public void createDataBase() throws IOException{

boolean dbExist = checkDataBase();

if(dbExist){
//la base de datos existe y no hacemos nada.
}else{
//Llamando a este método se crea la base de datos vacía en la ruta por defecto de
//de nuestra aplicación por lo que podremos sobreescribirla con nuestra base de d
this.getReadableDatabase();

try {

copyDataBase();

} catch (IOException e) {
throw new Error("Error copiando Base de Datos");
}
}

}

/**
* Comprueba si la base de datos existe para evitar copiar siempre el fichero cada
* @return true si existe, false si no existe
*/
private boolean checkDataBase(){

SQLiteDatabase checkDB = null;

try{

String myPath = DB_PATH + DB_NAME;
checkDB = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.OPEN_READONLY)

}catch(SQLiteException e){

//si llegamos aqui es porque la base de datos no existe todavía.

}
if(checkDB != null){

checkDB.close();

}
return checkDB != null ? true : false;
}
```

Método para sobrescribir la BD vacía que acabamos de crear:

```
/**
* Copia nuestra base de datos desde la carpeta assets a la recién creada
* base de datos en la carpeta de sistema, desde dónde podremos acceder a ella.
* Esto se hace con bytestream.
* */
private void copyDataBase() throws IOException{

//Abrimos el fichero de base de datos como entrada
InputStream myInput = myContext.getAssets().open(DB_NAME);

//Ruta a la base de datos vacía recién creada
String outFileName = DB_PATH + DB_NAME;

//Abrimos la base de datos vacía como salida
OutputStream myOutput = new FileOutputStream(outFileName);

//Transferimos los bytes desde el fichero de entrada al de salida
byte[] buffer = new byte[1024];
int length;
while ((length = myInput.read(buffer))>0){
myOutput.write(buffer, 0, length);
}

//Liberamos los streams
myOutput.flush();
myOutput.close();
myInput.close();

}
```

Método para abrir la BD:

```
public void open() throws SQLException{

//Abre la base de datos
try {
createDataBase();
} catch (IOException e) {
throw new Error("Ha sido imposible crear la Base de Datos");
}

String myPath = DB_PATH + DB_NAME;
myDataBase = SQLiteDatabase.openDatabase(myPath, null, SQLiteDatabase.OPEN_READONLY);

}
```

Una vez tenemos creada la clase DBHelper, basta con crear un objeto DBHelper y llamar al método open() desde la Activity. Este código deberás ponerlo solo la primera vez que intentes acceder a la base de datos por primera vez, yo te aconsejo que lo pongas seguido al OnCreate() para que te asegures de que inicializas la base de datos antes de usarla más adelante cuando te interese.

Base de datos superior a 1 Mb

El proceso de partir o dividir una base de datos es “complicado", tendremos que usar un comando "Split" que solo está en sistemas operativos Linux.

Requerimientos:

- VMware Player. Es el software que usaremos para la virtualización. (http://www.vmware.com/products/player/overview.html)

- Ubuntu Linux. Un ISO que usaremos para la instalación de Linux.

(http://www.ubuntu.com/download/desktop)

- Un PenDrive USB. Para intercambiar ficheros entre nuestro ordenador físico, que seguramente correrá bajo Windows y nuestra máquina virtual que corre bajo Linux.

Una vez tengas copiada la Base de datos en la memoria USB, ve a la barra de menú el VMware, pulsa sobre "Removable Devices" y busca en la lista tu dispositivo USB, después selecciona la opción "Connect (Disconect from host)".

Como ves, se te agregara la memoria USB en tu máquina virtual (Linux), y ya podremos dividir la base de datos, si instalas las WMware Tools (te aparecerá un aviso debajo de la pantalla), podrás arrastrar archivos entre la maquina física y la máquina virtual, y no necesitaras PenDrive, esto ya como tú quieras, el resultado.

Desde el PenDrive arrastro y copio la BD en la carpeta "Home", de esta forma en el Terminal no tendremos que movernos entre carpetas, por defecto, cuando entramos en el Terminal estaremos en esa carpeta.

En el terminal, ejecutas la sentencia: split -b 1m “nombreBD” “nombreNuevo_”

El resultado serán varios archivos de 1Mb con un sufijo: nombreNuevo_aa, nombreNuevo_ab, nombreNuevo_ac…

Para recomponer el archivo de BD en la aplicación de Android en la clase extendida de SQLiteOpenHelper, es necesario copiar los ficheros resultantes del comando “Split” a la carpeta “raw”, que está dentro de la carpeta “res”.

El código para este caso, es casi similar al anterior, solamente cambia el contenido del método copyDataBase().

```
private void copyDataBase() throws IOException {

    OutputStream databaseOutputStream = new
    // FileOutputStream("/data/data/es.epinanab.calculadoraticket/databases/db_calc
    FileOutputStream("" + DB_PATH + DB_NAME);
    InputStream databaseInputStream;

    byte[] buffer = new byte[1024];
    int length;

    //Comenzamos a recomponer el fichero
    databaseInputStream = myContext.getResources().openRawResource(
            R.raw.db_calc_aa);
    while ((length = databaseInputStream.read(buffer)) > 0) {
        databaseOutputStream.write(buffer);
    }
    databaseInputStream.close();

    databaseInputStream = myContext.getResources().openRawResource(
            R.raw.db_calc_ab);
    while ((length = databaseInputStream.read(buffer)) > 0) {
        databaseOutputStream.write(buffer);
    }
    databaseInputStream.close();

    databaseInputStream = myContext.getResources().openRawResource(
            R.raw.db_calc_ac);
    while ((length = databaseInputStream.read(buffer)) > 0) {
        databaseOutputStream.write(buffer);
    }
    databaseInputStream.close();

    databaseInputStream = myContext.getResources().openRawResource(
            R.raw.db_calc_ad);
    while ((length = databaseInputStream.read(buffer)) > 0) {
        databaseOutputStream.write(buffer);
    }
    databaseInputStream.close();

    databaseInputStream = myContext.getResources().openRawResource(
            R.raw.db_calc_ae);
    while ((length = databaseInputStream.read(buffer)) > 0) {
        databaseOutputStream.write(buffer);
    }
    databaseInputStream.close();

    databaseInputStream = myContext.getResources().openRawResource(
            R.raw.db_calc_af);
    while ((length = databaseInputStream.read(buffer)) > 0) {
        databaseOutputStream.write(buffer);
    }
    databaseInputStream.close();

    //cerramos el fichero
    databaseOutputStream.flush();
    databaseOutputStream.close();
}
```

Bibliografía

http://www.sgoliver.net/blog/bases-de-datos-en-android-i-primeros-pasos/

http://www.sgoliver.net/blog/bases-de-datos-en-android-ii-insertaractualizareliminar/

http://www.sgoliver.net/blog/bases-de-datos-en-android-iii-consultarrecuperar-registros/

http://es.slideshare.net/mejiaff/ejemplo-base-de-datos-sqlite-android

http://developer.android.com/reference/android/database/sqlite/package-summary.html

http://www.tutorialspoint.com/android/android_sqlite_database.htm

http://blog.netrunners.es/usar-nuestra-propia-base-de-datos-sqlite-en-android/

http://www.aprendeandroid.com/l5/sql5.htm