PDM Avanzado Datos Bases de datos

Sumario

Introdución

Información adicional: http://developer.android.com/guide/topics/data/data-storage.html

Android permite manexar bases de datos SQLite.

Neste curso partimos de que o alumno coñece o que é unha base de datos relacional e todo o que leva consigo (claves primarias, atributos, nulos, sentenzas SQL,...)

Máis información: http://es.wikipedia.org/wiki/Base_de_datos_relacional

Creación dunha base de datos

O primeiro que temos que facer é crear unha clase que herde da clase abstracta SQLiteOpenHelper (http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html).

Nota: Podemos facelo nun paquete separado para dar maior claridade ao noso proxecto.

Ao facelo, o IDE, obrigaranos a crear un construtor e uns métodos asociados á clase abstracta **SQLiteOpenHelper**.

Ao final de todo teremos o seguinte código, partindo que a clase que creamos ten de nome BaseDatos:

```
1 import android.content.Context;
2 import android.database.sqlite.SQLiteDatabase;
3 import android.database.sqlite.SQLiteDatabase.CursorFactory;
4 import android.database.sqlite.SQLiteOpenHelper;
5
6 public class BaseDatos extends SQLiteOpenHelper{
```

```
 7
 8     public BaseDatos(Context context, String name, CursorFactory factory,
 9             int version) {
10         super(context, name, factory, version);
11         // TODO Auto-generated constructor stub
12     }
13
14     @Override
15     public void onCreate(SQLiteDatabase db) {
16         // TODO Auto-generated method stub
17
18     }
19
20     @Override
21     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
22         // TODO Auto-generated method stub
23
24     }
25 }
```

Analicemos o construtor.

```
1     public BaseDatos(Context context, String name, CursorFactory factory,
2             int version) {
3         super(context, name, factory, version);
4         // TODO Auto-generated constructor stub
5     }
```

Ten catro parámetros:

- **Context context**: O contexto da Activity.
- **String name**: O nome da base de datos a abrir.
- **CursorFactory factory**: Este valor normalmente leva **null** xa que é utilizado cando queremos facer consultas que devolvan Cursores (http://developer.android.com/reference/android/database/Cursor.html) que nos van permitir acceder ós datos dunha consulta de forma aleatoria.
- **int version**: Versión da base de datos. Verémolo despois.

Para facer máis sinxelo o construtor ímolo modificar e deixarémolo así:

```
1     public final static String NOME_BD="basedatos";
2     public final static int VERSION_BD=1;
3
4     public BaseDatos(Context context) {
5         super(context, NOME_BD, null, VERSION_BD);
6         // TODO Auto-generated constructor stub
7     }
```

Cando dende a nosa Activity cremos un obxecto desta clase e abrimos a base de datos, chamará ós métodos **onCreate** ou **onUpgrade** segundo os casos.

- Se a base de datos **non existe**, chamará ao método **onCreate**. Será dentro deste método onde se poñan as ordes para crear as diferentes táboas da nosa base de datos.
- Se a base de datos **xa existe**, e a versión da base de datos é diferente, chamará ao método **onUpgrade**. Será dentro deste método onde poñamos as ordes de modificación das táboas da nosa base de datos.

Poñamos un exemplo:

- Un usuario descarga a nosa aplicación do Market e empeza a traballar con ela. A primeira vez chamará ao método onCreate e xeraranse as táboas necesarias.
- Cando leva un tempo traballando coa aplicación (engadiu datos ás táboas), decide actualizar a nosa aplicación. Imaxinemos que na actualización, necesitamos engadir unha táboa nova.
  - Como a base de datos xa existe, non vai pasar polo método **onCreate**.
  - Como facemos para engadir a nova táboa?
    - Cambiamos o número de versión da base de datos e subindo a aplicación compilada ao Market.
    - Cando o usuario descargue a aplicación actualizada executará o método **onUpgrade** e é nese método onde teremos a creación da nova táboa.
    - Os datos almacenados non se perden.

Nota: Veremos máis adiante que cando se crea a base de datos, gárdase no cartafol **/data/data/paquete_aplicación/databases/**

Normalmente a base de datos xa a teremos feita (cunha ferramenta externa) e o que facemos é copiala a dito cartafol. Cando o usuario abra a base de datos xa non vai pasar polo método onCreate.

Para que se chamen a ditos procedementos (**onCreate / onUpgrade**) será necesario crear un obxecto da clase que deriva de SQLiteOpenHelper e chamar a un destes métodos:

- **getReadableDatabase()** (http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getReadableDatabase%28%29): Abre a base de datos para operacións de só lectura.
- **getWritableDatabase()** (http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html#getWritableDatabase%28%29): Abre a base de datos para operacións de lectura / escritura.

Manualmente

Debemos crear e sobrescribir o método **onCreate** para crear as táboas que conforman a base de datos.

Dito método (**onCreate**) trae como parámetro un obxecto da clase **SqLiteDatabase** (http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html) que posúe o método **execSQL** para lanzar as ordes SQL necesarias.

- **Nota:**
  - Máis información sobre a orde **Create Table**: https://www.sqlite.org/lang_createtable.html
  - Máis información sobre os tipos de datos en sqlite: https://www.sqlite.org/datatype3.html

Se necesitamos modificar as táboas ou crear novas deberemos sobrescribir o método **onUpgrade**.

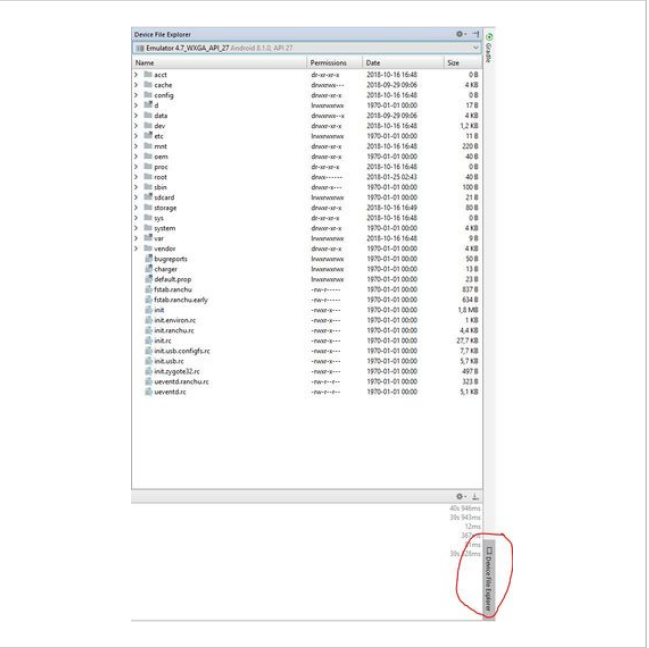
Manual: Caso práctico

Nesta práctica imos crear unha base de datos cunha táboa.

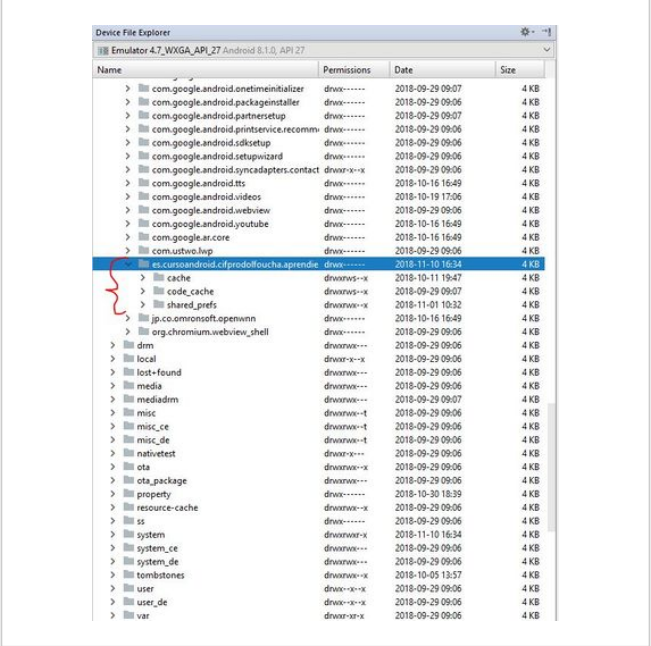
Para facelo teremos que abrir a base de datos en modo lectura ou lectura/escritura como xa comentamos. Neste exemplo vaise abrir en modo lectura/escritura.

Creando unha base de datos

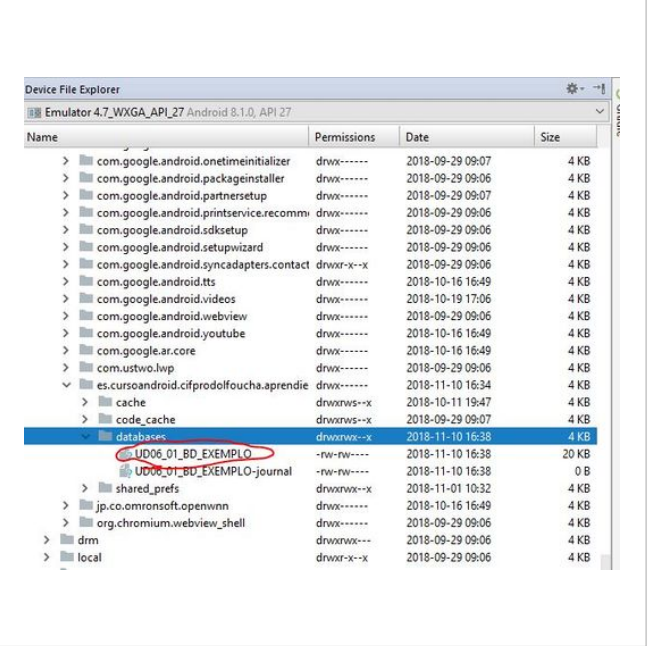
Aspecto que ten a aplicación. Consta dun único botón que o premer abrirá a base de datos en modo lectura/escritura.



Antes de executar a aplicación podemos comprobar na lapela DEVICE FILE EXPLORER do emulador...



Que o cartafol /databases/ non existe.



Unha vez prememos o botón podemos comprobar como a base de datos está creada no cartafol /databases/ dentro do paquete da aplicación.

Aviso Se o alumno está a probar a aplicación nun dispositivo real non hai opción de comprobar se e base de datos está creada a no ser que o dispositivo físico estea *rooteado*.

Creamos a clase que vai xestionar a BD

- Partimos que xa temos creado o proxecto inicial como xa indicamos anteriormente (http://wiki.cifprodolfoucha.es/index.php?title=PDM_Creando_proxecto_base).

  Se non o temos creado antes, crearemos un novo paquete de nome: **Persistencia** como un subpaquete do teu paquete principal.

Se non o temos creado antes, crearemos un novo paquete de nome: **BasesDatos** como un subpaquete do paquete anterior.

- Neste exemplo imos crear unha clase de nome **UD05_01_BaseDatos** que derive da clase SQLiteOpenHelper como explicamos antes. Esta clase estará creada dentro do paquete BasesDatos

- O nome da base de datos será **UD05_01_BD_EXEMPLO**
- Faremos que no método **onCreate** se cre a seguinte táboa:
  - Táboa: AULAS (_id integer autonumérico clave primaria, nome varchar(50))
- A modo de exemplo, faremos que no método **onUpgrade** se borren todas as táboas e se volvan crear.

Creamos a clase UD05_01_BaseDatos:

Código da clase UD05_01_BaseDatos

Obxectivo: Clase que vai executar as ordes para a creación das táboas e operacións contra a base de datos (SELECT, UPDATE, DELETE).

```
1 package es.cursoandroid.cifprodolfoucha.aprendiendo.Persistencia.BasesDatos;
2
3 import android.content.Context;
4 import android.database.sqlite.SQLiteDatabase;
5 import android.database.sqlite.SQLiteOpenHelper;
6
7 public class UD05_01_BaseDatos extends SQLiteOpenHelper{
8     public final static String NOME_BD="UD05_01_BD_EXEMPLO";
9     public final static int VERSION_BD=1;
10
11     private String CREAR_TABOA_AULAS ="CREATE TABLE AULAS ( " +
12             "_id  INTEGER PRIMARY KEY AUTOINCREMENT," +
13             "nome VARCHAR( 50 )  NOT NULL)";
14
15
16     public UD05_01_BaseDatos(Context context) {
17         super(context, NOME_BD, null, VERSION_BD);
18         // TODO Auto-generated constructor stub
19     }
20
21     @Override
22     public void onCreate(SQLiteDatabase db) {
23         // TODO Auto-generated method stub
24         db.execSQL(CREAR_TABOA_AULAS);
25
26     }
27
28     @Override
29     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
30         // TODO Auto-generated method stub
31
32         db.execSQL("DROP TABLE IF EXISTS AULAS");
33         onCreate(db);
34     }
35
36 }
```

Se executamos máis veces a aplicación non vai volver a executar o método **onCreate** xa que a base de datos está creada.
Teríamos que desinstalar a aplicación ou cambiar o número de versión da base de datos (iría ao método **onUpgrade**).

Creamos a Activity Principal

- Dentro do paquete **BasesDatos** (creado previamente no paso anterior) crear unha nova 'Empty Activity' de nome: **UD05_01_DatosPersistentes_BD** de tipo Launcher e sen compatibilidade.

Modifica o arquivo **AndroidManifiest.xml** e engade unha label á activity como xa vimos na creación do proxecto base (http://wiki.cifprodolfoucha.es/index.php?title=PDM_Creando_proxecto_base).

Código xml do layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Persistencia.BasesDatos.UD05_01_A1_DatosPersistentes_BD">

    <Button
        android:id="@+id/btnCrearBD_UD05_01_A1_BaseDatos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="CREAR BASE DE DATOS"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/btnPecharBD_UD05_01_A1_BaseDatos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="8dp"
        android:text="PECHAR BASE DE DATOS"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.347" />

</android.support.constraint.ConstraintLayout>
```

Código da clase UD05_01_DatosPersistentes_BD

Obxectivo: Crear unha base de datos.

```java
package es.cursoandroid.cifprodolfoucha.aprendiendo.Persistencia.BasesDatos;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

import es.cursoandroid.cifprodolfoucha.aprendiendo.R;

public class UD05_01_DatosPersistentes_BD extends Activity {
    private UD05_01_BaseDatos baseDatos;

    private void xestionarEventos(){

        findViewById(R.id.btnCrearBD_UD05_01_A1_BaseDatos).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (baseDatos==null){
                    baseDatos = new UD05_01_BaseDatos(getApplicationContext());
                    baseDatos.getWritableDatabase();

                    Toast.makeText(getApplicationContext(), "BD ABERTA PARA LECTURA/ESCRITURA", Toast.LENGTH_LONG).show();
                }

            }
        });
        findViewById(R.id.btnPecharBD_UD05_01_A1_BaseDatos)
                .setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {
                        if (baseDatos!=null){
                            baseDatos.close();
                            baseDatos = null;
                            Toast.makeText(getApplicationContext(),"BD PECHADA",Toast.LENGTH_SHORT).show();
                        }
                    }
                });


    }
```

```
41
42      @Override
43      protected void onCreate(Bundle savedInstanceState) {
44          super.onCreate(savedInstanceState);
45          setContentView(R.layout.activity_UD05_01__datos_persistentes__bd);
46
47          xestionarEventos();
48      }
49  }
```

- Liña 11: Creamos unha propiedade que vai servir para poder realizar operacións contra a base de datos.
- Liña 19: Instanciamos a propiedade.
- Liña 20: Abrimos a base de datos para operacións de escritura / lectura. É agora cando, se a base de datos non existe, chamará o método onCreate.
- Liña 32: Pechamos a base de datos ao premer no botón.

- **NOTA IMPORTANTE:** Fixarse que aínda que premamos moitas veces no botón de crear base de datos, o método onCreate soamente se chama no caso de que non exista a base de datos. Se xa existe non se chama.
- Podedes probar a poñer un número de versión diferente á base de datos e comprobar cun Log como pasa polo método onUpgrade.

- Agora temos que resolver o 'problema' de onde se van a realizar as operacións contra a base de datos...

Se o fixéramos dende a Activity (é dicir, na activity irían as ordes SQL (INSERT, UPDATE, DELETE, SELECT)) soamente teríamos que gardar a referencia ao obxecto SQLiteDatabase que devolve o método getWritableDatabase() ou o método getReadableDatabase() da seguinte forma:

```
1  public class UD05_01_BD_EXEMPLO extends AppCompatActivity {
2
3      private UD05_01_BaseDatos baseDatos;
4      private SQLiteDatabase operacionsBD;
5
6      private void xestionarEventos(){
7          findViewById(R.id.btnCrearBD_UD05_01_A1_BaseDatos)
8                  .setOnClickListener(new View.OnClickListener() {
9                      @Override
10                     public void onClick(View v) {
11                         if (baseDatos==null){
12                             baseDatos = new UD05_01_BaseDatos(getApplicationContext());
13                             operacionsBD = baseDatos.getReadableDatabase();
14                             Toast.makeText(getApplicationContext(),"BD ABERTA PARA LECTURA/ESCRITURA",Toast.LENGTH_SHORT).show();
15                         }
16                     }
17                 });
18     ...........
```

A partires de entón, cando se queira facer unha operación contra a base de datos teríamos que facer:

```
1 ...
2 operacionsBD.rawQuery("select _id,nome from AULAS", null);
3 ...
```

Veremos máis adainte o que fai dita orden. O que tedes que ter claro é que calquera operación SQL se ten que facer empregando o obxeto SQLiteDatabase que devolve o método getReadableDatabase() ou getWritableDatabase().

- No noso caso, imos facer todas as operación dentro da clase UD05_01_BaseDatos (a que derive de SQLiteOpenHelper) e polo tanto necesitamos gardar dentro de dita clase unha referencia ao obxecto operacionsBD.

A forma 'óptima' de facelo é empregando o patrón Singleton (https://es.wikipedia.org/wiki/Singleton) que veremos máis adiante, neste exemplo imos ver unha solución sinxela...
O único que temos que facer é modificar a **clase UD05_01_BaseDatos**:

```
1  public class UD05_01_BaseDatos extends SQLiteOpenHelper {
2
3      public final static String NOME_BD = "basedatos.db";
4      public final static int VERSION_BD=2;
5
6      private SQLiteDatabase operacionsBD;
```

```
 7
 8      private String CREAR_TABOA_AULAS ="CREATE TABLE AULAS ( " +
 9              "_id  INTEGER PRIMARY KEY AUTOINCREMENT," +
10             "nome VARCHAR( 50 )  NOT NULL)";
11
12      public UD05_01_BaseDatos(Context context) {
13          super(context, NOME_BD, null, VERSION_BD);
14      }
15
16      public void asigarSQLiteDatabase(SQLiteDatabase operacionsBD){
17          this.operacionsBD = operacionsBD;
18      }
19
20      .....
```

- E agora dende a AppCompactActivity **UD05_01_BD_EXEMPLO** gardamos a referencia:

```
 1      ...........
 2      private void xestionarEventos(){
 3          findViewById(R.id.btnCrearBD_UD05_01_A1_BaseDatos)
 4                  .setOnClickListener(new View.OnClickListener() {
 5                      @Override
 6                      public void onClick(View v) {
 7                          if (baseDatos==null){
 8                              baseDatos = new UD05_01_BaseDatos(getApplicationContext());
 9                              SQLiteDatabase operacionsBD = baseDatos.getWritableDatabase();
10                              baseDatos.asigarSQLiteDatabase(operacionsBD);
11                              Toast.makeText(getApplicationContext(),"BD ABERTA PARA LECTURA/ESCRITURA",Toast.LENGTH_SHORT).show();
12                          }
13                      }
14                  });
15
16       ............
```

- Agora imos poder facer todas as operacións dentro da clase UD05_01_BaseDatos facendo uso da referencia ao obxecto operacionsBD.

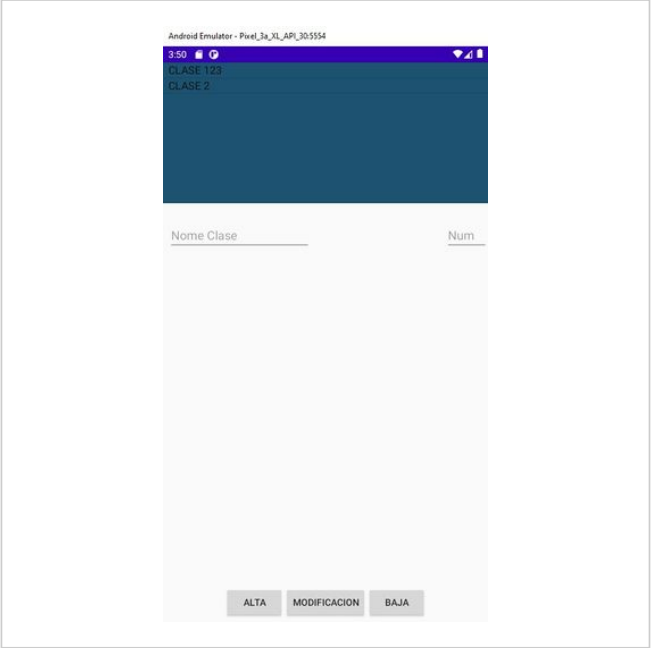
- **IMPORTANTE:** Loxicamente nunha aplicación 'normal' abriremos, nas activities necearias, a base de datos no método onCreate e a pecharemos no método onDestroy.

Ferramenta interna: Database Inspector

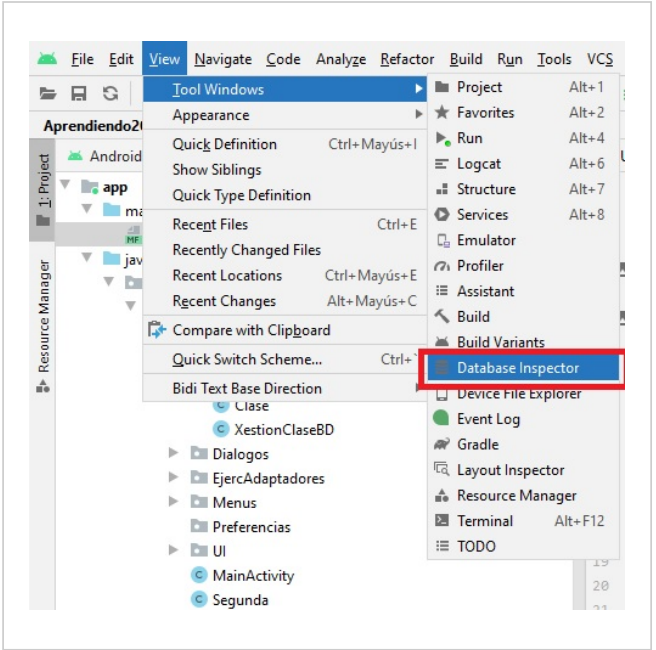
- A partires do **Android Studio versión 4.1.1** é posible acceder e visualizar os datos dunha base de datos asociada a unha aplicación mentres se está a executar.

Para iso é necesario que a versión do emulador sexa maior ou igual á API 26.

Accedendo ao Database Inspector

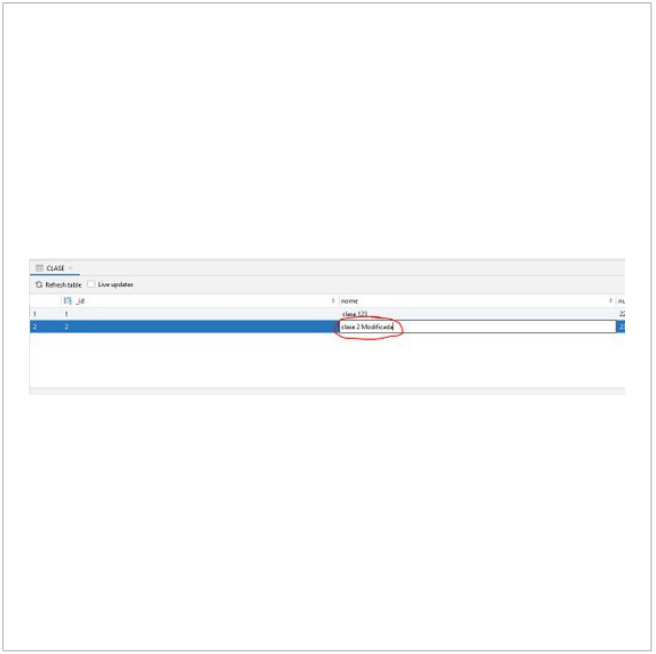


Executamos a aplicación que abre a base de datos e vai facer operacións sobre ela.

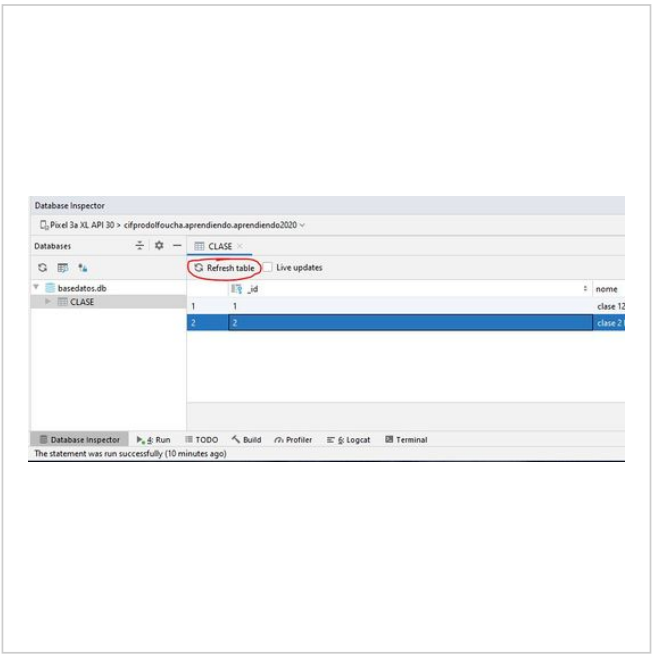


Abrimos a ferramenta Database Inspector.

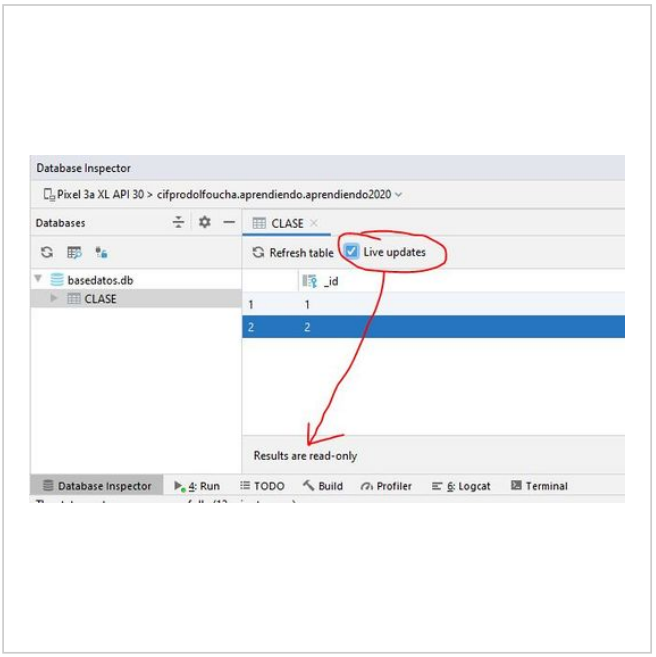
Na parte baixa aparecerá outra lapela có nome 'Database inspector'. Debemos asegurarnos que estean seleccionados o emulador e o proceso da aplicación que está correndo no emulador. Ao premer dúas veces sobre unha táboa, aparecerá en forma de grid os datos da mesma.



Se prememos dúas veces sobre un dato, podemos editalo e modificar o seu valor.

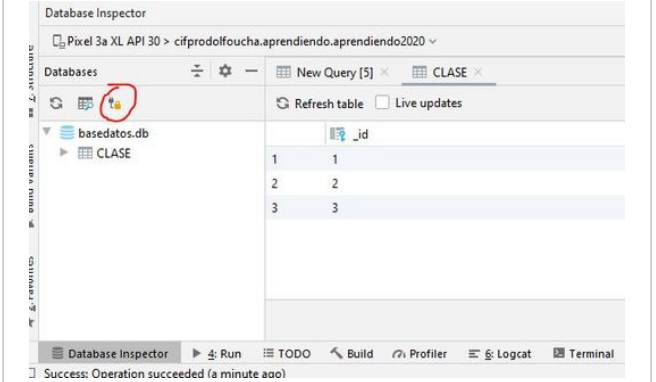


Por defecto, os datos non está sincronizados coa aplicación, polo que se modificamos os datos a través da aplicación será necesario refrescar os datos da táboa premendo no botón **Refresh**.



Se queremos que estean sincronizados e calquera modificación dos datos a través do emulador se reflictan automaticamente no Database Inspector, debemos marcar a opción **Live Update**. Ao facelo xa non podemos modificar os datos dende o Android Studio.

Podemos crear as nosas propias consultas premendo na opción **Open New Query Tap**. Abrirase unha nova lapela onde podemos escribir calquera sentenza SQL (INSERT, UPDATE, DELETE, SELECT). Ao escribila debemos premer no botón **RUN**, aparecendo os resultados na parte inferior.



Se marcamos a opción **Keep Database Connection Open** poderemos acceder á base de datos aínda que se peche a conexión dende a aplicación do emulador.

Ferramenta externa: SqliteStudio

Normalmente, cando creamos a nosa aplicación e vai utilizar unha base de datos, non imos creala no método onCreate mediante ordes CREATE TABLE.

O lóxico será tela creada cuns datos iniciais.

Esta base de datos se atopará nun cartafol (por exemplo **/assets/**) e copiaremos a base de datos dende /assets/ ao cartafol onde Android busca a base de datos.

Para crear a base de datos temos moitas ferramentas gratuítas para facelo.

Por exemplo: http://sqlitestudio.pl/

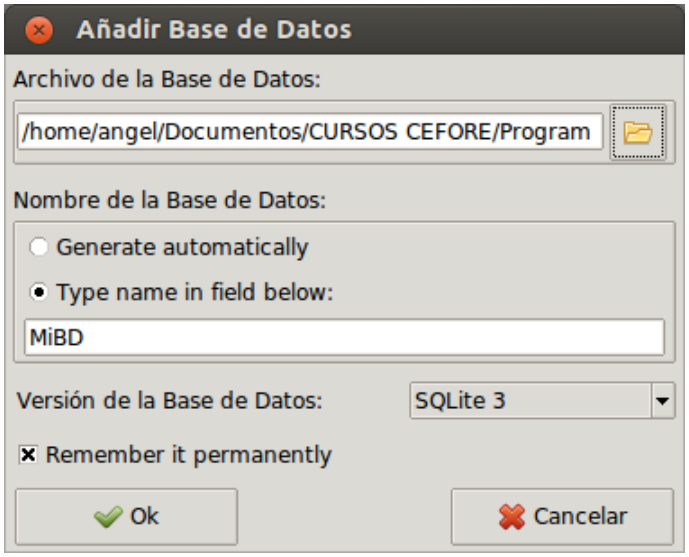
Nota: Se estamos a utilizar Linux deberemos darlle permiso de execución ao arquivo baixado coa orde:

```
chmod 755 sqlitestudio-2.1.4.bin
```

e despois executala:

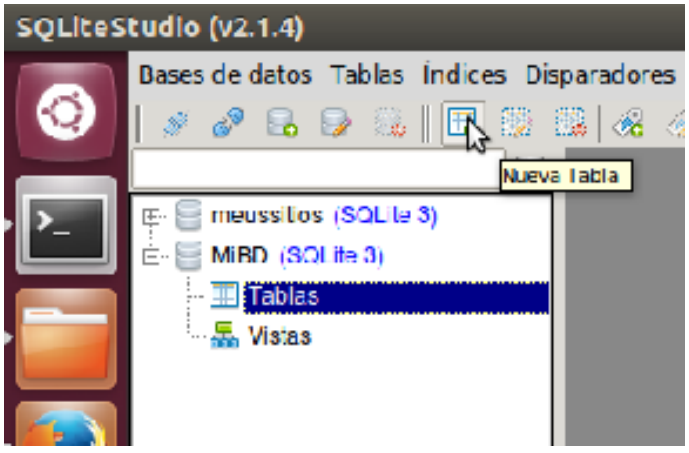
```
./sqlitestudio-2.1.4.bin
```

- Unha vez no programa podemos crear unha base de datos nova indo o menú **Base de datos => Engadir Base de datos**.



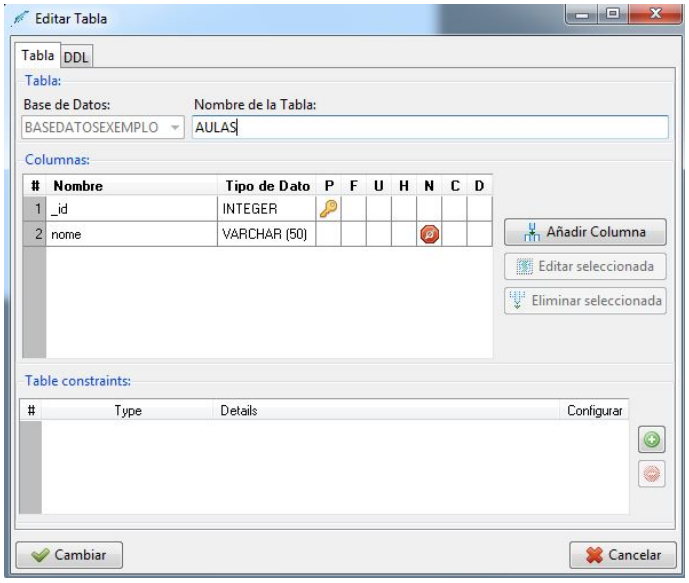
Aquí temos que indicar o sitio físico onde se vai gardar / abrir a base de datos e o nome da mesma.
A versión poñeremos sqlite3.
Debemos premer o botón de 'Ok'.

- Agora debemos de escoller a opción de **Tablas** e unha vez escollida premer a opción **Nueva Tabla** da parte superior.

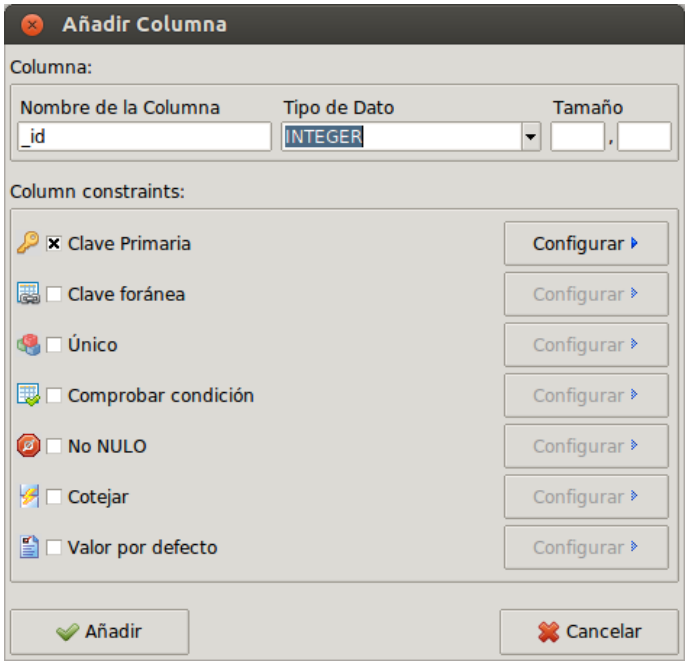


- Esta pantalla é bastante intuitiva.

- Damos un nome á táboa (no exemplo AULAS) e prememos o botón 'Añadir Columna'.



- Agora só temos que engadir as columnas co tipo de dato adecuado:

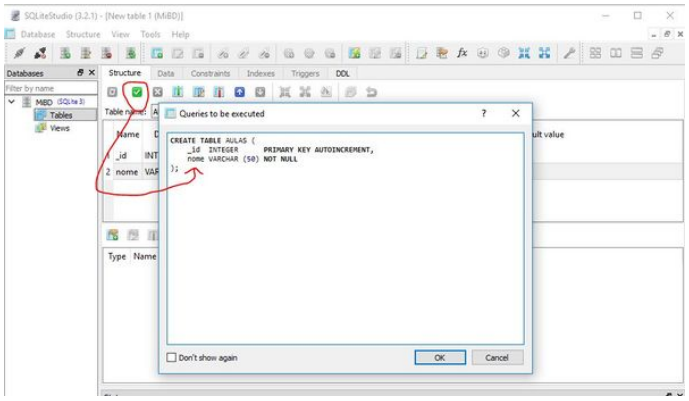


Se prememos na opción configurar podemos facer varias cousas dependendo do que teñamos escollido.

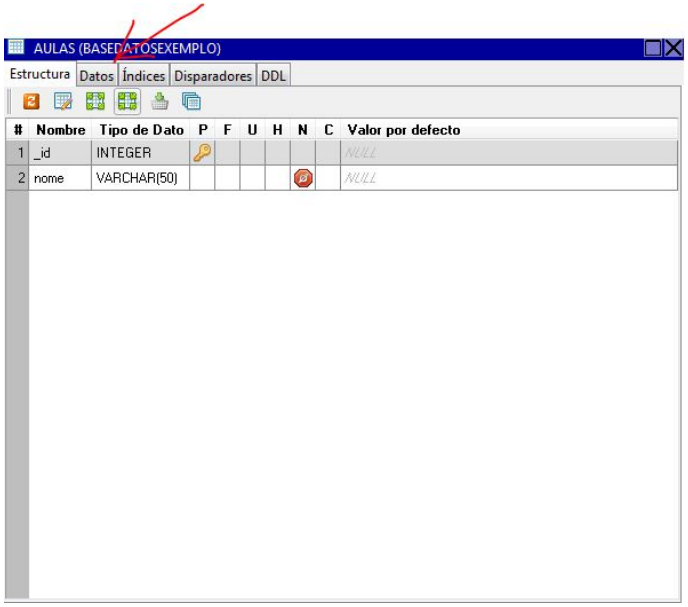
Por exemplo:

- Na clave primaria: podemos facer que o campo sexa autonumérico. Quere isto dicir que cando engadimos unha nova fila non é necesario darlle un valor a dita columna xa que vai xerar automaticamente un valor.
- No valor por defecto: podemos especificar o valor que terá dita columna na fila se non enviamos ningún.

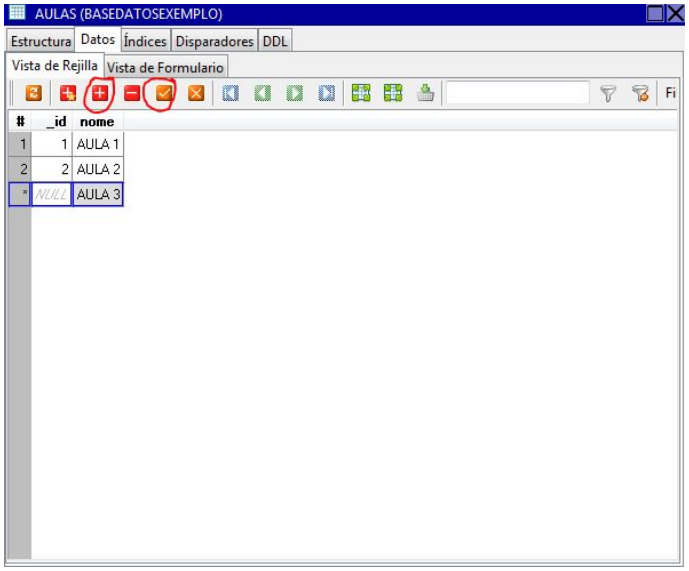
Unha vez temos todos os campo debemos premer o botón de **Crear**. Aparecerá unha nova ventá coa orde SQL para crear a táboa. Poderíamos copiala e gardala xa que podemos necesitar ter as ordes para crear a base de datos:



- Unha vez creada a táboa xa podemos engadir filas ou modificar os campos premendo sobre a pestana de datos:



- Temos que premer sobre o botón **Máis**, engadir os datos e ao rematar de engadir as filas, premer sobre o botón **Commit**:



Copiando a base de datos dende /assets/ a .../databases/

Como comentamos antes, a base de datos podemos tela xa creada e con datos preparada para ser utilizada pola nosa aplicación.

Para que esta poida utilizala teremos que copiala ao cartafol **/data/data/paquete_aplicación/databases/**

A forma de copiala xa a vimos cando vimos o apartado de xestión de arquivos (http://wiki.cifprodolfoucha.es/index.php?title=PDM_Avanzado_Datos_Persistentes_Arquivos).

Normalmente a copia debería facerse nun fío separado do fío principal. O uso dos fíos xa verémolo na unidade 7 (http://wiki.cifprodolfoucha.es/index.php?title=Programaci%C3%B3n_de_dispositivos_m%C3%B3biles#UNIDADE_7:_Threads_e_AsyncTask).

O único que temos que ter coidado é que se copiamos a base de datos, o cartafol **.../databases/** non está creado e teremos que crealo nos.

Fisicamente as bases de datos se crean no cartafol **/data/data/nome do paquete/databases/NOME BD**.

Se queremos ter unha referencia ao obxecto **File** que apunte á nosa base de datos podemos usar a chamada:

```
getDatabasePath(NOME_BD)
```

ou ben ter unha referencia ao seu path e partires del obter un obxecto da clase File:

```
String pathbd = "/data/data/" + getApplicationContext().getPackageName()+"/databases/";
```

Como comentamos antes o normal é ter a base de datos creada no cartafol /assets/ e copiala o cartafol /databases/.

- Se o facemos así será necesario crear previamente dito cartafol /databases/:

```
1 String pathbd = "/data/data/" + contexto.getPackageName()+"/databases/";
2 File ruta = new File(pathbd);
3 ruta.mkdirs();
```

- Unha vez creado o cartafol xa podemos copiar a base de datos. Neste exemplo non se usa un fío de execución.

  Comprobamos se existe a base de datos previamente, xa que se non, cada vez que iniciáramos a aplicación borraríamos a base de datos.

```
1         String bddestino = "/data/data/" + getPackageName() + "/databases/"
2                 + UD5_05_BASEDATOS.NOME_BD;
3         File file = new File(bddestino);
4         if (file.exists())
5             return; // XA EXISTE A BASE DE DATOS
```

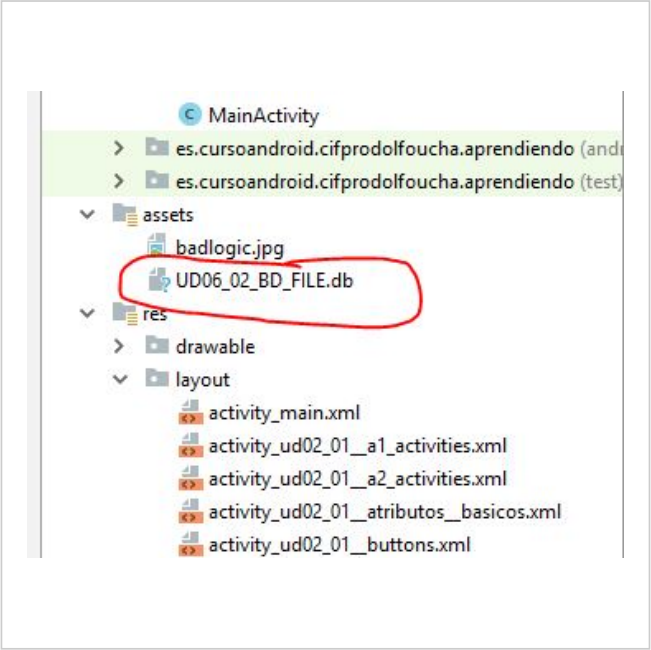
Copiamos a base de datos.

```
 1         String pathbd = "/data/data/" + getPackageName()
 2                 + "/databases/";
 3         File filepathdb = new File(pathbd);
 4         filepathdb.mkdirs();
 5
 6         InputStream inputstream;
 7         try {
 8             inputstream = getAssets().open(UD5_05_BASEDATOS.NOME_BD);
 9             OutputStream outputstream = new FileOutputStream(bddestino);
10
11             int tamread;
12             byte[] buffer = new byte[2048];
13
14             while ((tamread = inputstream.read(buffer)) > 0) {
15                 outputstream.write(buffer, 0, tamread);
16             }
17
18             inputstream.close();
19             outputstream.flush();
20             outputstream.close();
21         } catch (IOException e) {
22             // TODO Auto-generated catch block
23             e.printStackTrace();
24         }
```

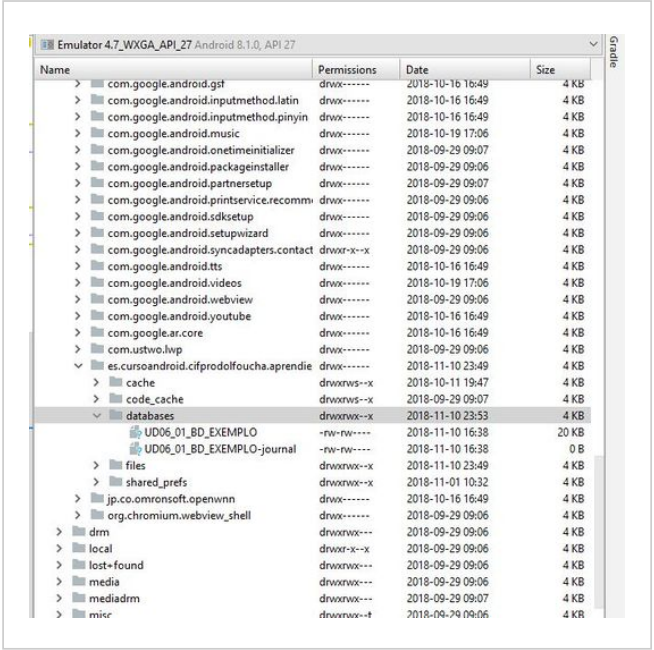
Caso práctico

Imos copiar unha base de datos feita coa ferramenta anterior dende o cartafol /assets/ ao cartafol /databases/.

Copiando unha base de datos existente

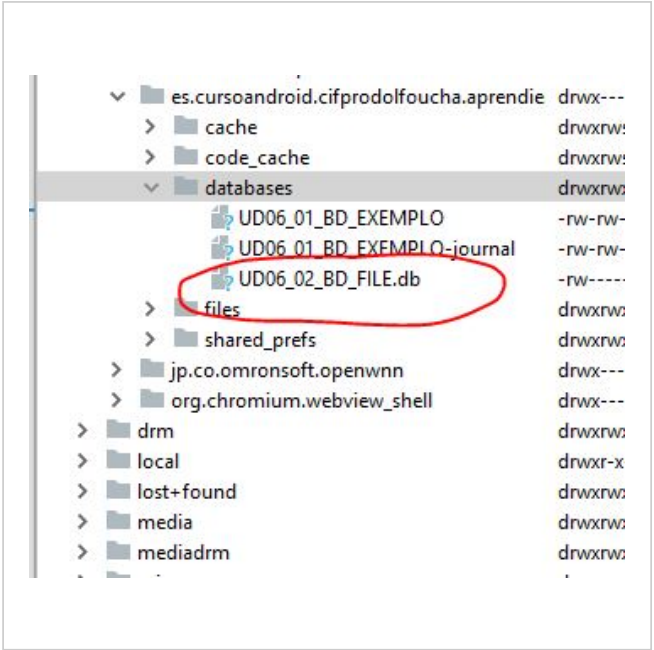


Temos a base de datos a copia no cartafol /assets/

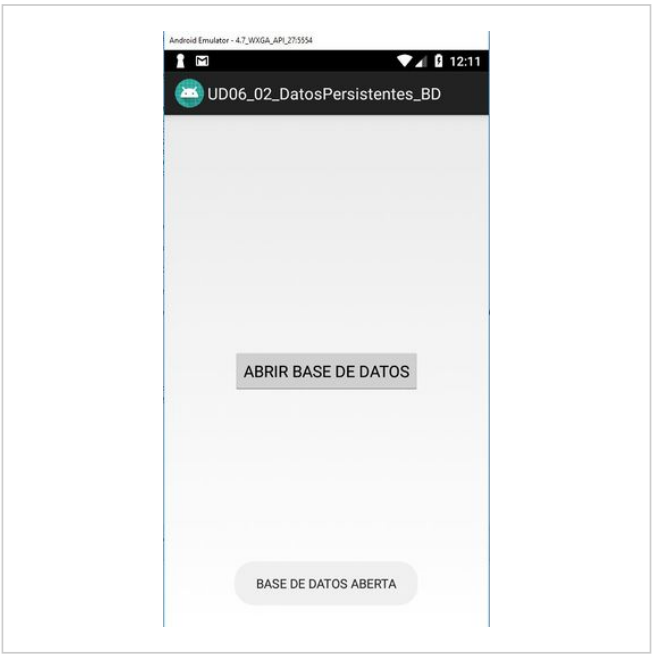


Podemos comprobar como a base de datos deste exercicio (a número 02) non existe.

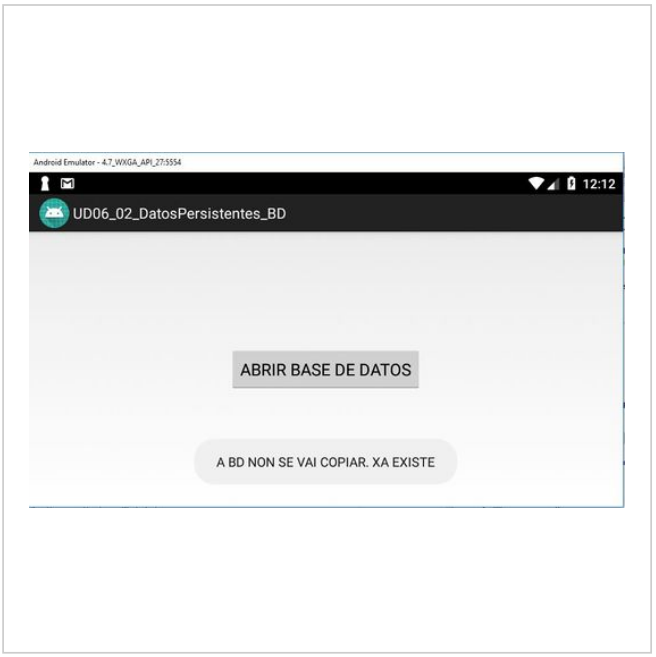
Executamos a aplicación e como a base de datos non está copiada, a copia dende /assets/ ao cartafol 'databases/'.



Agora podemos comprobar como a base de datos xa está copiada.



Podemos premer o botón de abrir. Se a base de datos non existira, crearía unha baleira segundo o establecido no método onCreate.



Se volvemos a executar a aplicación ou rotamos a pantalla, como a base de datos xa foi copiada, non a volver a copiar.

Preparación

Hai que descomprimir e copiar o seguinte arquivo ao cartafol /assets/ do voso proxecto.

Media:EX_02_BD_FILE.zip

O cartafol /assets/ xa o deberíamos ter creado dunha práctica anterior (http://wiki.cifprodolfoucha.es/index.php?title=PDM_Avanzado_Datos_Persistentes_Arquivos#Preparaci.C3.B3n).

Nota: O alumno pode utilizar a súa propia base de datos xerada coa ferramenta anterior.

Creamos a Activity

- Dentro do paquete **BasesDatos** (creado previamente no paso anterior) crear unha nova 'Empty Activity' de nome: **UD05_02_DatosPersistentes_BD** de tipo Launcher e sen compatibilidade.

  Modifica o arquivo **AndroidManifiest.xml** e engade unha label á activity como xa vimos na creación do proxecto base (http://wiki.cifprodolfoucha.es/index.php?title=PDM_Creando_proxecto_base).

Código da clase UD05_02_BaseDatos

Esta clase é a que vai xestionar a base de datos. É igual ao código anterior pero cambiando o nome da base de datos.
Como neste exercicio o que imos facer é copiar unha base de datos con táboas poderíamos non poñer o código no método onCreate e método onUpgrade, pero lembrar que se fixéramos unha actualización da nosa aplicación, teríamos que poñer neses método o código necesario para adaptar a base de datos aos novos requerimentos (podería ser necesario borrar toda a base de datos e volver a creala)

- Como podemos ter varias referencias a esta clase (que fagan uso dela diferentes fragments, por exemplo), imos asegurarnos que soamente existe unha instancia da mesma, mediante o uso de patrón Singleton (https://es.wikipedia.org/wiki/Singleton).

  Isto se implementa da forma seguinte:

```
public class MiClase {
    private static MiClase sInstance;



    public static synchronized MiClase getInstance() {
        if (sInstance == null) {
            sInstance = new MiClase ();
        }
        return sInstance;
    }

}
```

Agora se quero facer uso desta clase non fago: MiClase miClase = new MiClase();
Terei que poñer: **MiClase miClase = MiClase.getInstance();**

Se o fago á vez dende diferentes clase sempre estarei empregando a mesma instancia.
Da outra forma teríamos un obxecto diferente.

- Aplicado ao noso caso:

```
package es.cursoandroid.cifprodolfoucha.aprendiendo.Persistencia.BasesDatos;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class UD05_02_BaseDatos extends SQLiteOpenHelper{
    public final static String NOME_BD="EX_02_BD_FILE.db";
    public final static int VERSION_BD=1;
    private static UD05_02_BaseDatos sInstance;

    public static synchronized UD05_02_BaseDatos getInstance(Context context) {
        // Use the application context, which will ensure that you
        // don't accidentally leak an Activity's context.
        // See this article for more information: http://bit.ly/6LRzfx
        if (sInstance == null) {
            sInstance = new UD05_02_BaseDatos(context.getApplicationContext());
        }
        return sInstance;
    }

    private String CREAR_TABOA_AULAS ="CREATE TABLE AULAS ( " +
            "_id  INTEGER PRIMARY KEY AUTOINCREMENT," +
            "nome VARCHAR( 50 )  NOT NULL)";


    public UD05_02_BaseDatos(Context context) {
        super(context, NOME_BD, null, VERSION_BD);
        // TODO Auto-generated constructor stub
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // TODO Auto-generated method stub
       // db.execSQL(CREAR_TABOA_AULAS);

    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // TODO Auto-generated method stub

        // db.execSQL("DROP TABLE IF EXISTS AULAS");
        // onCreate(db);
    }

}
```

Código do layout xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Persistencia.BasesDatos.UD05_02_DatosPersistentes_BD">

```

```
    <Button
        android:id="@+id/btnAbrirBD_UD05_02_BaseDatos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:text="ABRIR BASE DE DATOS"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Código da clase UD05_02_DatosPersistentes_BD

Obxectivo: Copiar unha base de datos dende /assets/ ao cartafol /databases/.

```
package es.cursoandroid.cifprodolfoucha.aprendiendo.Persistencia.BasesDatos;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import es.cursoandroid.cifprodolfoucha.aprendiendo.R;

public class UD05_02_DatosPersistentes_BD extends Activity {
    private UD05_02_BaseDatos baseDatos;

    private void copiarBD() {
        String bddestino = "/data/data/" + getPackageName() + "/databases/"
                + UD05_02_BaseDatos.NOME_BD;
        File file = new File(bddestino);
        if (file.exists()) {
            Toast.makeText(getApplicationContext(), "A BD NON SE VAI COPIAR. XA EXISTE", Toast.LENGTH_LONG).show();
            return; // XA EXISTE A BASE DE DATOS
        }

        // En caso de que non exista creamos o cartafol /databases/
        String pathbd = "/data/data/" + getPackageName()
                + "/databases";
        File filepathdb = new File(pathbd);
        if (!filepathdb.exists()) {
            filepathdb.mkdirs();
        }

        InputStream inputstream;
        try {
            inputstream = getAssets().open(UD05_02_BaseDatos.NOME_BD);
            OutputStream outputstream = new FileOutputStream(bddestino);

            int tamread;
            byte[] buffer = new byte[2048];

            while ((tamread = inputstream.read(buffer)) > 0) {
                outputstream.write(buffer, 0, tamread);
            }

            inputstream.close();
            outputstream.flush();
            outputstream.close();
            Toast.makeText(getApplicationContext(), "BASE DE DATOS COPIADA", Toast.LENGTH_LONG).show();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    private void xestionarEventos(){

        findViewById(R.id.btnAbrirBD_UD05_02_BaseDatos).setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                baseDatos = UD05_02_BaseDatos.getInstance(getApplicationContext());
                baseDatos.getWritableDatabase();

                Toast.makeText(getApplicationContext(), "BASE DE DATOS ABERTA", Toast.LENGTH_LONG).show();
            }
        });
    }

    @Override
```

```
73      protected void onCreate(Bundle savedInstanceState) {
74          super.onCreate(savedInstanceState);
75          setContentView(R.layout.activity_UD05_02__datos_persistentes__bd);
76
77          xestionarEventos();
78          copiarBD();
79      }
80  }
```

- Liñas 19-55: Procedemento para copiar a base de datos. Xa o vimos na sección de Arquivos (http://wiki.cifprodolfoucha.es/index.php?title=PDM_Avanzado_Datos_Persistentes_Arquivos)].
    - Liñas 23-26: Miramos se a base de datos existe. Neste exemplo non a copiamos se existe. Nunha aplicación real poderíamos enviar un parámetro para indicar se queremos sobreesribila xa que pode ser necesario se queremos 'inicializar' a aplicación.
- Liñas 59-70: Xestionamos o evento de click sobre o botón de abrir. Como a base de datos xa está copiada podemos utilizala.
    - Liña 64: Fixarse como empregamos o patrón 'Singleton' para obter unha instancia da clase que xestiona a base de datos.
    - Liña 65: Fixarse que abrimos a base de datos para ler/escribir información nela.

Xestionar a base de datos dende consola

Se estamos a utilizar o emulador ou podemos ser root no dispositivo real, podemos conectarnos á base de datos dende consola.

O proceso é o seguinte:

- Abrimos un terminal (linux) ou consola (windows).
- Facemos un cd do caratafol onde está instalado o SDK e dentro deste ir ao cartafol /sdk/plataform-tools/.

  Se listades os arquivo vos debe aparecer un executable de nome 'adb'.
- Dende a consola deberedes escribir:
    - WINDOWS:
        - adb root
        - adb shell
    - LINUX
        - ./adb root
        - ./adb shell
- Unha vez no shell, conectamos coa base de datos escribindo:
    - sqlite3 /data/data/o_voso_paquete/databases/NOME_BASEDATOS
- Agora estamos conectados a SQLIte e podemos executar ordes SQL (deben acabar en ;).

  Por exemplo:
    - .tables : lista as táboas.
    - SELECT * FROM NOME_TABOA; (amosa os datos)
- Para saír do sqlite poñeremos:

  .exit
- Para saír do adb shell:

  exit

Nota:

- Coidado cas maiúsculas / minúsculas.
- Tedes a lista completa de comandos neste enlace (https://www.sqlite.org/cli.html).

Operacións sobre unha base de datos

Introdución

As operacións que imos facer sobre a base de datos son:

- **SELECT**: Selección.
- **UPDATE**: Actualización.
- **DELETE**: Borrado.
- **INSERT**: Engadir.

Nota: Neste curso consideramos que os alumnos teñen coñecementos para entender as ordes SQL que dan soporte a estas operacións.

Para poder realizar as operacións anteriormente comentadas imos necesitar facer uso dun obxecto da clase SQLiteDatabase (http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html).

Podemos obtelo de dúas formas:

- Dito obxecto o temos como parámetro no método **onCreate** e no método **onUpgrade** da clase que deriva de **SQLiteOpenHelper**:

```
1     @Override
2     public void onCreate(SQLiteDatabase db) {
3         // TODO Auto-generated method stub
4
5     }
6
7     @Override
8     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
9         // TODO Auto-generated method stub
10
11    }
```

  Polo tanto podemos facer uso do **obxecto db** para realizar as operacións SQL. Nestes métodos normalmente utilizaranse cando *non* se fai o proceso de copia da base de datos e estamos a crear a base de datos manualmente con ordes 'create table'.

- Cando abrimos a base de datos para escribir ou para ler:

```
1 SQLiteDatabase sqlLiteDB = ud4_05_baseDatos.getWritableDatabase();
```

  Nota: Código baseado a partires do exemplo anterior.

Agora con dito obxecto podemos facer as operacións SQL.

INSERT

Engadir unha nova fila.

Máis información en: http://www.w3schools.com/sql/sql_insert.asp

Temos dúas formas de engadir datos:

- Utilizando a orden INSERT de SQL:

```
1 sqlLiteDB.execSQL("INSERT INTO NOME_TABOA (campo1,campo2,....) VALUES ('valor1','valor2',....)");
```

Nota: Os valores numéricos non levan comillas.

- A segunda forma é utilizando un obxecto da clase ContentValues (http://developer.android.com/reference/android/content/ContentValues.html), o cal garda pares da forma clave-valor, e no noso caso, serán pares nome_columna-valor.

Chamaremos despois aos métodos **insert()**, **update()** e **delete()** da clase SQLiteDatabase.

Por exemplo:

```
1         ContentValues datosexemplo = new ContentValues();
2         datosexemplo.put("NOME_COL1", "Valor col1_a");
  3         datosexemplo.put("NOME_COL2", "Valor_col2_a");
4         long idFila1 = sqlLiteDB.insert(NOME_TABOA, null, datosexemplo);
5
6                 datosexemplo.clear();
7         datosexemplo.put("NOME_COL1", "Valor col1_b");
8         datosexemplo.put("NOME_COL2", "Valor_col2_b");
9         long idFila2 = sqlLiteDB.insert(NOME_TABOA, null, datosexemplo);
```

- O segundo parámetro leva de valor null, xa que só se usa en contadas ocasións, coma por exemplo, para engadir rexistros baleiros.
- Neste exemplo supoñemos que a táboa onde se van engadir os rexistros ten de clave primaria un campo autonumérico.
- O método insert devolve o id da fila engadida (a clave principal é autonumérica) ou -1 se hai un erro.

UPDATE

Actualiza datos.

Máis información en: http://www.w3schools.com/sql/sql_update.asp

O método update() leva como terceiro parámetro a condición que teñen que cumprir as filas.

```
1 ContentValues datosexemplo = new ContentValues();
2 datosexemplo.put("COLUMNA_MODIFICAR","TEXTO NOVO");
3 String condicionwhere = "COL1=?";
4 String[] parametros = new String[]{"Texto a buscar"};
5 int rexistrosafectados = sqlLiteDB.update(NOME_TABOA,datosexemplo,condicionwhere,parametros);
```

Se queremos unha condición composta (por exemplo con AND):

```
1 ContentValues datosexemplo = new ContentValues();
2 datosexemplo.put("COLUMNA_MODIFICAR","TEXTO NOVO");
3 String condicionwhere = "COL1=? AND COL2=?";
4 String[] parametros = new String[]{"Texto1","Texto2"};
5 int rexistrosafectados = sqlLiteDB.update(NOME_TABOA,datosexemplo,condicionwhere,parametros);
```

Nun exemplo concreto:

TÁBOA AULAS (_id, nome):

Esta consulta sql:

```
1 UPDATE AULAS
2 SET nome='Novo nome'
3 WHERE _id=5;
```

Pasará a ser:

```
1 ContentValues datos = new ContentValues();
2 datos.put("nome","Novo nome");
3 String condicionwhere = "_id=?";
4 String[] parametros = new String[]{String.valueOf(5)};
5 int rexistrosafectados = sqlLiteDB.update("AULAS",datos,condicionwhere,parametros);
```

Aquí hai que sinalar varios aspectos:

- Se queremos poñer unha condición a un campo de tipo texto NON teremos que poñer o parámetro entre comiñas:
  - NOME + "=?" **(Correcto)**
  - NOME + "='?'" (Incorrecto)
- Se o facemos directamente na orde SQL entón si que as levaría:

  String CONSULTA="SELECT id,nome FROM CLASES WHERE nome = 'AULA 1'";
- Cada '?' se corresponde cun dato do array de parámetros. No noso exemplo enviamos o ID que teremos que converter a String:

  String[] parametros = new String[]{String.valueOf(5)};

- O método update devolve o número de filas afectadas ou -1 en caso de erro.

DELETE

Borra filas.

Máis información: http://www.w3schools.com/sql/sql_delete.asp

O método delete() é parecido ao anterior.

Pásase como primeiro parámetro a táboa e como segundo a condición que teñen que cumprir as filas para ser borradas:

```
1 String condicionwhere = "COL1=?";
2 String[] parametros = new String[]{'Valor a buscar'};
3 int rexistrosafectados = sqlLiteDB.delete(NOME_TABOA,condicionwhere,parametros);
```

- O método delete devolve o número de filas afectadas ou -1 en caso de erro.
- Os conceptos da cláusula **Where** son os mesmos que no caso do **Update**.

Nun exemplo concreto:

TÁBOA AULAS (_id, nome):

Esta consulta sql:

```
1 DELETE FROM AULAS
2 WHERE _id=5;
```

Pasará a ser:

```
1 String condicionwhere = "_id=?";
2 String[] parametros = new String[]{String.valueOf(5)};
3 int rexistrosafectados = sqlLiteDB.delete("AULAS",condicionwhere,parametros);
```

SELECT

A selección de filas leva consigo que o resultado vai poder traer de volta moitas filas.

Veremos máis adiante como devolver esas filas como se foran un Array de obxectos.

Para facer consultas á base de datos podemos utilizar dous métodos da clase **SQLiteQueryBuilder** : rawQuery e query

- **rawQuery**

  A diferenza vai estar na forma de facer a chamada, xa que rawQuery deixa enviar a orde SQL e query utiliza outra estrutura de chamada.

  Por exemplo:

  ```
  1   Cursor cursor = sqlLiteDB.rawQuery("select _id,nome from AULAS where _id = ?", new String[] { String.valueOf(5) });
  ```

  - O segundo parámetro é un array de Strings no caso de que a consulta leve parámetros (símbolo ?).

- **query**

  ```
  1 Cursor cursor = sqlLiteDB.query(DATABASE_TABLE,
  2   new String[] { col1,col2,... },
  3   null, null, null, null, null);
  ```

  Os parámetros en query son:

  - DATABASE_TABLE: Nome da táboa.
  - Lista de columnas, nun array de String. Se poñemos null devolve todas.

- Cláusula where
- Array de string no caso de que a cláusula where leve parámetros da forma 'id=?', é dicir, co caracter ?.
- Array de String onde irán os nomes das columnas para facer group by.
- Array de String onde irán os nomes das columnas para facer having.
- Array de String onde irán os nomes das columnas para facer order by.

Por exemplo, a orde SQL:

SELECT nome FROM AULAS

sería:

```
1 Cursor cursor = sqlLiteDB.query("AULAS",
2   new String[] { nome },
3   null, null, null, null, null);
```

Nos imos a usar a primeira opción.

Os dous tipos de consultas (rawquery e query) devolven un obxecto Cursor. Para saber o nº de filas devoltas temos o método **getCount()** e dispoñemos de diferentes métodos para movernos polas filas do cursor.

O proceso normalmente será o de percorrer todo o cursor e devolver un **ArrayList** dun tipo determinado (verase despois).

```
1   Cursor cursor = sqlLiteDB.rawQuery("select _id,nome from AULAS", null);
2   while (cursor.moveToNext()) {     // Quédase no bucle ata que remata de percorrer o cursor
3         long id =cursor.getLong(0);
4         String nome = cursor.getString(1);
5   }
```

Dentro do bucle temos acceso ós datos de cada fila.
Cada columna da consulta está referenciada por un número, empezando en 0 (a columna _id correspóndese coa número 0 e a columna nome coa número 1). Isto é asi porque no select están nesa orde.

MOI IMPORTANTE: Cando rematemos de procesar a fila teremos que pasar á seguinte dentro do cursor e temos que chamar ó método **moveToNext()**. Se non o facemos quedaremos nun bucle infinito. No exemplo o moveToNext() xa vai no bucle, pero poderíamos poñer como condición while(!cursor.isAferLast()) e ter o cursor.moveNext() dentro do bucle.

Neste exemplo non estamos a facer nada con cada fila devolta. Poderíamos ter una Array e ir engadindo a dito Array os valores que devolve a consulta, pero veremos a continuación que é mellor facelo utilizando obxectos.

Aclaración: Cando creamos a táboa AULAS puxemos como clave primaria **_id**. Por que utilizar un guión baixo e ese nome ?
Porque unha consulta á base de datos pode devolver directamente o Cursor con todos os datos e 'cargar' ditos datos nun adaptador especial denominado SimpleCursorAdapter (http://developer.android.com/reference/android/widget/SimpleCursorAdapter.html)] e asociar dito Adaptador a un elemento gráfico como unha lista.

Por exemplo:

```
1 SimpleCursorAdapter mAdapter = new SimpleCursorAdapter(this,R.layout.list_layout_creado, cursor, new String[] { "nome" },new
int[] { android.R.id.text1});
```

Sendo:

- 'list_layout_creado': O layout que vai amosar a ListView.
- cursor: Os datos devoltos pola base de datos de todas as aulas.
- nome: O nome da columna que queremos amosar da táboa da base de datos.
- android.R.id.text1: Un TextView definido dentro do layout 'list_layout_creado'.

- Para que funcione ten que existir unha columna '_id' na táboa onde se fai a consulta.

> Lembrar que xa vimos o uso de adaptadores na Unidade 4 desta Wiki (http://wiki.cifprodolfoucha.es/index.php?title=Spinner_a_trav%C3%A9s_de_adaptador).

Onde facer as operacións

Neste curso imos facer as operacións contra a base de datos na clase que deriva da clase **abstracta SQLiteOpenHelper** (onde se atopan os métodos onCreate e onUpgrade).

Poderíamos facelo na activity a partires de obter o obxecto sqlLiteDB, pero desta forma quedará moito máis claro.

Polo tanto imos ter que gardar o obxecto sqlLiteDB na propia clase para poder ter acceso a el e poder facer as operacións.

Unha forma de facelo é a seguinte:

- Definimos unha propiedade public dentro da clase SQLiteOpenHelper.

```
public class UD6_03_BaseDatos extends SQLiteOpenHelper{

    private SQLiteDatabase sqlLiteDB;
        ..............
```

- Creamos un método **abrirBD()** y **pecharBD()**:

```
 1 public class UD6_03_BaseDatos extends SQLiteOpenHelper{
 2     public final static String NOME_BD="UD05_03_BD_FILE.db";
 3     public final static int VERSION_BD=1;
 4     private static UD05_03_BaseDatos sInstance;
 5
 6     private SQLiteDatabase dbsqlLiteDB;
 7
 8     public static synchronized UD05_03_BaseDatos getInstance(Context context) {
 9         // Use the application context, which will ensure that you
10         // don't accidentally leak an Activity's context.
11         // See this article for more information: http://bit.ly/6LRzfx
12         if (sInstance == null) {
13             sInstance = new UD05_03_BaseDatos(context.getApplicationContext());
14         }
15         return sInstance;
16     }
17         ..............
18     public void abrirBD(){
19         if (sqlLiteDB==null || !sqlLiteDB.isOpen()){
20           sqlLiteDB = sInstance.getWritableDatabase();
21         }
22     }
23
24     public void pecharBD(){
25         if (sqlLiteDB!=null || sqlLiteDB.isOpen()){
26           sqlLiteDB.close();
27         }
28     }
```

- Agora definimos os método que necesitemos para dar soporte á nosa aplicación.

Xuntando todo cos Modelos

Normalmente os datos dunha base de datos os imos 'modelizar' en forma de clases.

Así, se se ten unha táboa AULAS poderase modelizar utilizando unha clase AULAS que teña como propiedades as columnas da táboa.

Non pretendemos entrar a explicar o UML (http://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado) e o Diagrama de Clases (http://es.wikipedia.org/wiki/Diagrama_de_clases).

Tampouco imos traballar con capas intermedias. Atoparedes manuais nos que se crea unha clase 'Adaptadora' que é a que traballa con **ContentValues' *e*** *Cursors*. Por enriba desta clase atoparíase unha clase 'de xestión' que sería a que exporía os métodos para facer operacións dende a interface do usuario e que convertía un obxecto da clase Aula a ContentValues, para ser engadida á base de datos (por exemplo) e tamén o proceso contrario, pasar de, por exemplo, un Cursor a un array de obxectos da clase Aulas.

Imos velo cun exemplo concreto:

- TÁBOA AULAS: (_id, nome)
- Definimos a clase que vai gardar esta información: **Aulas**

Creamos esta clase dentro do paquete 'BaseDatos' (nas prácticas teredes que crear un paquete cun nome concreto)

```
1 public class Aulas {
2
3     private long _id;
4     private String nome;
5
6     public Aulas (long id, String nome){
7         this._id=id;
8         this.nome=nome;
9     }
10
11    public long get_id() {
12        return _id;
13    }
14    public void set_id(long _id) {
15        this._id = _id;
16    }
17    public String getNome() {
18        return nome;
19    }
20    public void setNome(String nome) {
21        this.nome = nome;
22    }
23
24    public String toString(){
25        return nome;
26    }
27 }
```

NOTA IMPORTANTE: Definimos o método toString() xa que imos 'cargar' ós elementos gráficos (como as listas=> ListView) con array de obxectos. Neses casos o elemento gráfico (a lista) amosa o que devolva o método toString().

NOTA:: Dentro de Android hai artigos que indican que facer métodos get e set para acceder as propiedades degradan o rendemento da aplicación e que sería conveniente acceder directamente ás propiedades (facéndoas public). En todas as aplicacións que levo feitas non atopei ese 'rendemento inferior' polo que supoño que só será apreciable en aquelas aplicacións que fagan un uso moi intensivo de datos.

- Agora que temos definida a clase podemos definir os métodos que imos necesitar.

Nota: Lembrar que as operacións contra a base de datos se definirán na clase que deriva de SQLiteOpenHelper (onde se atopan os métodos onCreate e onUpgrade).

Por exemplo:

- Método: engadirAula(Aulas aula_engadir).

Engade unha aula á base de datos.
Leva como parámetro a aula a engadir.

```
1     public long engadirAula(Aulas aula_engadir){
2         ContentValues valores = new ContentValues();
3         valores.put("nome", aula_engadir.getNome());
4         long id = sqlLiteDB.insert("AULAS",null,valores);
5
6         return id;
7     }
```

Como vemos devolve o id da aula engadida. O id é xerado automaticamente pola BD cando engadimos unha fila (lembrar que o ID é autonumérico e non se envía).
Nota:Poderíamos devolver un obxecto da clase Aula co novo id xa posto.

- Método: ArrayList<Aulas> obterAulas().

  Devolve as aulas da táboa AULAS.
  A forma de devolver moitos obxectos dunha clase é utilizando un ArrayList (http://developer.android.com/reference/java/util/ArrayList.html).

```
1         private final String CONSULTAR_AULAS ="SELECT _id,nome FROM AULAS order by nome";
2
3     public ArrayList<Aulas> obterAulas() {
4         ArrayList<Aulas> aulas_devolver = new ArrayList<Aulas>();
5
6         Cursor datosConsulta = sqlLiteDB.rawQuery(CONSULTAR_AULAS, null);
7         if (datosConsulta.moveToFirst()) {
8             Aulas aula;
9             while (!datosConsulta.isAfterLast()) {
10                aula = new Aulas(datosConsulta.getLong(0),
11                        datosConsulta.getString(1));
12                aulas_devolver.add(aula);
13                datosConsulta.moveToNext();
14            }
15        }
16        return aulas_devolver;
17    }
```

Caso práctico

Neste exemplo imos copiar unha base de datos dende /assets/ ao cartafol .../databases/.

Dita BD consta dunha única táboa de nome **AULAS** cos seguintes campos:

- **_id**: clave autonumérica.
- **nome**: nome da aula.

A aplicación consta dunha lista (ListView) unha caixa de texto (EditText) e tres botóns para realizar operacións de Alta-Baixa-Modificación sobre as aulas.

Cabe sinalar que por motivos de tempo non están contempladas todas as condicións que teríamos que ter en conta para facer as operacións.

Así, por exemplo, cando damos de alta unha aula, non se comproba que tiñamos escrito algo na caixa de texto.

Operacións sobre unha base de datos

Aspecto da aplicación ao iniciarse.

Exemplo de aula modificada.

Exemplo de aula dada de baixa.

Preparación

- Deberemos de ter no cartafol /assets/ a base de datos. Descomprimir o arquivo e copiar a base de datos a dito cartafol:

Media:EX_03_BD_FILE.zip

Creamos a clase que xestiona a base de datos

Nome da clase: UD05_03_BaseDatos

Código da clase que xestiona a base de datos.

```
 1 package es.cursoandroid.cifprodolfoucha.aprendiendo.Persistencia.BasesDatos;
 2
 3 import android.content.ContentValues;
 4 import android.content.Context;
 5 import android.database.Cursor;
 6 import android.database.sqlite.SQLiteDatabase;
 7 import android.database.sqlite.SQLiteOpenHelper;
 8
 9 import java.util.ArrayList;
10
11 public class UD05_03_BaseDatos extends SQLiteOpenHelper{
12     public final static String NOME_BD="EX_03_BD_FILE.db";
13     public final static int VERSION_BD=1;
14     private static UD05_03_BaseDatos sInstance;
15     private SQLiteDatabase sqlLiteDB;
```

```
16
17      /* SENTENZAS SQL */
18      private final String TABOA_AULAS="AULAS";
19      private final String CONSULTAR_AULAS ="SELECT _id,nome FROM AULAS order by nome";
20
21
22      public static synchronized UD05_03_BaseDatos getInstance(Context context) {
23          // Use the application context, which will ensure that you
24          // don't accidentally leak an Activity's context.
25          // See this article for more information: http://bit.ly/6LRzfx
26          if (sInstance == null) {
27              sInstance = new UD05_03_BaseDatos(context.getApplicationContext());
28          }
29          return sInstance;
30      }
31
32      private String CREAR_TABOA_AULAS ="CREATE TABLE AULAS ( " +
33              "_id  INTEGER PRIMARY KEY AUTOINCREMENT," +
34              "nome VARCHAR( 50 )  NOT NULL)";
35
36      public long engadirAula(Aulas aula_engadir){
37          ContentValues valores = new ContentValues();
38          valores.put("nome", aula_engadir.getNome());
39          long id = sqlLiteDB.insert("AULAS",null,valores);
40
41          return id;
42      }
43
44      public int borrarAula(Aulas aula){
45          String condicionwhere = "_id=?";
46          String[] parametros = new String[]{String.valueOf(aula.get_id())};
47          int rexistrosafectados = sqlLiteDB.delete(TABOA_AULAS,condicionwhere,parametros);
48
49          return rexistrosafectados;
50
51      }
52
53      public int modificarAula(Aulas aula_modificar){
54          ContentValues datos = new ContentValues();
55          datos.put("nome", aula_modificar.getNome());
56
57          String where = "_id=?";
58          String[] params = new String[]{String.valueOf(aula_modificar.get_id())};
59
60          int rexistrosModificados = sqlLiteDB.update(TABOA_AULAS, datos, where, params);
61
62          return rexistrosModificados;
63      }
64
65      public ArrayList<Aulas> obterAulas() {
66          ArrayList<Aulas> aulas_devolver = new ArrayList<Aulas>();
67
68          Cursor datosConsulta = sqlLiteDB.rawQuery(CONSULTAR_AULAS, null);
69          Aulas aula;
70          while (datosConsulta.moveToNext()) {
71             aula = new Aulas(datosConsulta.getLong(0),
72                              datosConsulta.getString(1));
73             aulas_devolver.add(aula);
74          }
75          return aulas_devolver;
76      }
77
78      public UD05_03_BaseDatos(Context context) {
79          super(context, NOME_BD, null, VERSION_BD);
80          // TODO Auto-generated constructor stub
81      }
82
83      public void abrirBD(){
84          if (sqlLiteDB==null || !sqlLiteDB.isOpen()){
85            sqlLiteDB = sInstance.getWritableDatabase();
86          }
87      }
88
89      public void pecharBD(){
90          if (sqlLiteDB!=null && sqlLiteDB.isOpen()){
91            sqlLiteDB.close();
92          }
93      }
94
95      @Override
96      public void onCreate(SQLiteDatabase db) {
97          // TODO Auto-generated method stub
98         // db.execSQL(CREAR_TABOA_AULAS);
99
100     }
101
102     @Override
103     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
104         // TODO Auto-generated method stub
105
106         //db.execSQL("DROP TABLE IF EXISTS AULAS");
107         //onCreate(db);
108     }
```

```
109
110 }
```

Creamos a activity

- Dentro do paquete **BasesDatos** (creado previamente no paso anterior) crear unha nova 'Empty Activity' de nome: **UD05_03_DatosPersistentes_BD** de tipo Launcher e sen compatibilidade.

  Modifica o arquivo **AndroidManifiest.xml** e engade unha label á activity como xa vimos na creación do proxecto base (http://wiki.cifprodolfoucha.es/index.php?title=PDM_Creando_proxecto_base).

Código do layout xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".Persistencia.BasesDatos.UD05_03_DatosPersistentes_BD">

    <ListView
        android:id="@+id/lstAulas_UD05_03_BaseDatos"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintBottom_toTopOf="@+id/guideline3"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <android.support.constraint.Guideline
        android:id="@+id/guideline3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_percent=".45" />

    <EditText
        android:id="@+id/etAula_UD05_03_BaseDatos"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:hint="Introduce un aula"
        android:inputType="textCapCharacters"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/guideline3" />

    <Button
        android:id="@+id/btnAlta_UD05_03_BaseDatos"
        android:layout_width="100sp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginStart="8dp"
        android:text="Alta"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <Button
        android:id="@+id/btnModificar_UD05_03_BaseDatos"
        android:layout_width="110sp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:text="Modificar"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/btnBaixa_UD05_03_BaseDatos"
        app:layout_constraintStart_toEndOf="@+id/btnAlta_UD05_03_BaseDatos" />
64
```

```
    <Button
        android:id="@+id/btnBaixa_UD05_03_BaseDatos"
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:text="Baixa"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Código da clase UD6_03_DatosPersistentes_BD

Obxectivo: Realizar operacións contra unha base de datos.

```
package es.cursoandroid.cifprodolfoucha.aprendiendo.Persistencia.BasesDatos;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.ArrayList;

import es.cursoandroid.cifprodolfoucha.aprendiendo.R;

public class UD05_03_DatosPersistentes_BD extends Activity {
    private UD05_03_BaseDatos baseDatos;
    private Aulas aula_seleccionada=null;   // Garda a aula seleccionada da lista

    private void copiarBD() {
        String bddestino = "/data/data/" + getPackageName() + "/databases/"
                + UD05_03_BaseDatos.NOME_BD;
        File file = new File(bddestino);
        if (file.exists()) {
            Toast.makeText(getApplicationContext(), "A BD NON SE VAI COPIAR. XA EXISTE", Toast.LENGTH_LONG).show();
            return; // XA EXISTE A BASE DE DATOS
        }

        String pathbd = "/data/data/" + getPackageName()
                + "/databases/";
        File filepathdb = new File(pathbd);
        filepathdb.mkdirs();

        InputStream inputstream;
        try {
            inputstream = getAssets().open(UD05_03_BaseDatos.NOME_BD);
            OutputStream outputstream = new FileOutputStream(bddestino);

            int tamread;
            byte[] buffer = new byte[2048];

            while ((tamread = inputstream.read(buffer)) > 0) {
                outputstream.write(buffer, 0, tamread);
            }

            inputstream.close();
            outputstream.flush();
            outputstream.close();
            Toast.makeText(getApplicationContext(), "BASE DE DATOS COPIADA", Toast.LENGTH_LONG).show();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

    private void xestionarEventos(){

        Button btnAltaAula = (Button)findViewById(R.id.btnAlta_UD05_03_BaseDatos);
        btnAltaAula.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                EditText editAula = (EditText) findViewById(R.id.etAula_UD05_03_BaseDatos);
                // Habería que comprobar se hai algún dato na caixa de texto, pero neste exemplo centrámonos no funcionamento da
base de datos.
```

```
73
74              Aulas aula = new Aulas(0, editAula.getText().toString());
75              long id = baseDatos.engadirAula(aula); // Obtemos o id. Neste exemplo non faremos nada con el.
76              aula.set_id(id);
77
78              // Poderíamos engadir a nova aula ó adaptador pero neste exemplo
79              // recargamos a lista
80              cargarLista();
81              editAula.setText("");
82
83              Toast.makeText(getApplicationContext(), "AULA ENGADIDA",Toast.LENGTH_LONG).show();
84          }
85      });
86
87      Button btnBaixaAula = (Button)findViewById(R.id.btnBaixa_UD05_03_BaseDatos);
88      btnBaixaAula.setOnClickListener(new View.OnClickListener() {
89
90          @Override
91          public void onClick(View v) {
92              // TODO Auto-generated method stub
93
94              baseDatos.borrarAula(aula_seleccionada);
95              EditText editAula = (EditText)findViewById(R.id.etAula_UD05_03_BaseDatos);
96              editAula.setText("");
97
98              // Poderíamos borrar a aula do adaptador e non cargar a lista novamente
99              cargarLista();
100             Toast.makeText(getApplicationContext(), "AULA BORRADA",Toast.LENGTH_LONG).show();
101         }
102     });
103
104     Button btnModificaraAula = (Button)findViewById(R.id.btnModificar_UD05_03_BaseDatos);
105     btnModificaraAula.setOnClickListener(new View.OnClickListener() {
106
107         @Override
108         public void onClick(View v) {
109             // TODO Auto-generated method stub
110
111             EditText editAula = (EditText)findViewById(R.id.etAula_UD05_03_BaseDatos);
112
113             Aulas aula_modificar = new Aulas(aula_seleccionada.get_id(),editAula.getText().toString());
114             baseDatos.modificarAula(aula_modificar);
115
116             // Poderíamos borrar a aula do adaptador e non cargar a lista novamente
117             cargarLista();
118             Toast.makeText(getApplicationContext(), "AULA MODIFICADA",Toast.LENGTH_LONG).show();
119         }
120     });
121
122
123
124     ListView lista = (ListView)findViewById(R.id.lstAulas_UD05_03_BaseDatos);
125     lista.setOnItemClickListener(new AdapterView.OnItemClickListener() {
126
127         @Override
128         public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
129                                 long arg3) {
130             // TODO Auto-generated method stub
131             // Obtemos o obxecto aula seleccionado
132             ArrayAdapter<Aulas> adaptador = (ArrayAdapter<Aulas>) arg0.getAdapter();
133             aula_seleccionada = adaptador.getItem(arg2);
134             EditText editAula = (EditText)findViewById(R.id.etAula_UD05_03_BaseDatos);
135             editAula.setText(aula_seleccionada.getNome());
136
137         }
138     });
139 }
140
141 /**
142  * Accede á base de datos para obter as aulas e as asina á lista.
143  */
144 private void cargarLista(){
145
146     ListView lista = (ListView)findViewById(R.id.lstAulas_UD05_03_BaseDatos);
147
148     ArrayList<Aulas> aulas = baseDatos.obterAulas();
149     ArrayAdapter<Aulas> adaptador = new ArrayAdapter<Aulas>(getApplicationContext(),
150             android.R.layout.simple_list_item_1,aulas);
151     lista.setAdapter(adaptador);
152
153 }
154
155
156 @Override
157 public void onStart(){
158     super.onStart();
159
160     if (baseDatos==null) {   // Abrimos a base de datos para escritura
161         baseDatos = UD05_03_BaseDatos.getInstance(getApplicationContext());
162         baseDatos.abrirBD();
163
164         cargarLista();
165     }
166 }
```

```
167
168     @Override
169     public void onStop(){
170         super.onStop();
171
172         if (baseDatos!=null){    // Pechamos a base de datos.
173             baseDatos.pecharBD();
174             baseDatos=null;
175         }
176
177     }
178
179     @Override
180     protected void onCreate(Bundle savedInstanceState) {
181         super.onCreate(savedInstanceState);
182         setContentView(R.layout.activity_UD05_03__datos_persistentes__bd);
183
184         copiarBD();
185         xestionarEventos();
186     }
187 }
```

- Liñas 66-85: Xestionamos o evento Click sobre o botón de alta.
    - Liña 74: Creamos un obxecto da clase Aula co contido do EditText. O id non vai utilizarse xa que cando damos de alta o id non se utiliza.
    - Liña 75: Chamamos á base de datos có aula a dar de alta. Devolve o id (autonumérico).
    - Liña 80: Chamamos a o método que volve chamar á base de datos para cargar as aulas novamente.
- Liñas 87-102: Xestionamos o evento Click sobre o botón de baixa.
    - Liña 94: Chamamos á base de datos para dar de baixa a aula seleccionada. Máis adiante está explicado que cando prememos sobre a lista gardamos en aula_seleccionada a aula seleccionada da lista.
    - Liña 99: Chamamos a o método que volve chamar á base de datos para cargar as aulas novamente.
- Liñas 104-120: Xestionamos o evento Click sobre o botón de modificar.
    - Liña 113: Creamos o obxecto aula que vai a enviarse á base de datos. O id é o id da aula seleccionada na lista e o texto é o do EditText.
    - Liña 114: Chamamos á base de datos para modificar o nome da aula seleccionada.
    - Liña 117: Chamamos a o método que volve chamar á base de datos para cargar as aulas novamente.
- Liñas 125-138: Xestionamos o evento Click sobre a lista.
    - Liña 133: Gardamos na propiedade 'aula_seleccionada' a aula seleccionada.
    - Liña 135: Cambiamos o texto do EditText polo seleccionado.
- Liñas 144-153: Cargamos as aulas dende a base de datos á lista.
- Liñas 161-164: Cando a aplicación empeza abrimos a base de datos e cargamos a lista coas aulas.
- Liñas 173-174: Liberamos os recursos.

Aclaración sobre el patrón Singlenton

- No exemplo anterior empregamos o patrón Singleton (https://es.wikipedia.org/wiki/Singleton) como forma para obter unha única referencia ao obxecto da clase que se vai encargar de conectar á base de datos e facer as operacións.

```
1 public class MiClase {
2     private static MiClase sInstance;
3
4
5
6     public static synchronized MiClase getInstance() {
7         if (sInstance == null) {
8             sInstance = new MiClase ();
9         }
10        return sInstance;
11    }
12
13 }
```

- Se empregamos esta forma temos que ter coidado xa que non imos poder 'abrir' á base de datos no método onStart() das activities.

  Isto é debido a un problema que teremos no seguinte exemplo:

  - Imaxinemos que temos dúas activities que fan uso da base de datos e as dúas teñen o código que abre a base de datos no método onStart().

    - A primeira Activity se carga e pasa por dito método obtendo unha referencia á clase que xestiona a Base de Datos.
    - Agora esta primeira Activity **chama á segunda Activity**. Ao facelo, primeiro pasa polo método onStart da segunda Activity e depois polo método onStop da primeira Activity.
    - Polo tanto a segunda Activity vai ter unha referencia ao obxecto **coa base de datos pechada** xa que a pechou a primeira Activity ao pasar polo método onStop.

- Para resolvelo poderíamos:

  - Facer sempre a comprobación de que a base de datos estea aberta antes de facer unha operación e abrila se non o está.
  - Abrir a base de datos ao iniciar a activity principal (no método onCreate) e pechala ao saír da aplicación (no método onDestroy).
  - Non empregar dito patrón e crear unha instancia da clase en cada Activity que necesite acceder á base de datos, abrindo a base de datos no método onStart() e pechando no método onStop()

Transaccións

- No caso de querer facer varias operacións de modificación de forma conxunta sobre a a base de datos será necesario crear unha transacción (https://es.wikipedia.org/wiki/ACID).

  Para iso temos que empregar o seguinte código (tendo como referencia o obxecto SQLiteDatabase => db no exemplo):

```
 1 db.beginTransaction();
 2 try {
 3     // Operaciones varias que impliquen modificación de datos.
 4
 5     db.setTransactionSuccessful();
 6 } catch {
 7     //Erro na transacción. Non debemos facer un return, xa que debe chegar a executar finally
 8 } finally {
 9     db.endTransaction();  // Se todo vai ben a transacción se confirma. En caso de erro fará o RollBack
10 }
```

- Máis información no seguinte enlace (https://www.sqlite.org/atomiccommit.html).

Enlace a la página principal de la UD5 (https://wiki.cifprodolfoucha.es/index.php?title=Programaci%C3%B3n_de_dispositivos_m%C3%B3biles#UNIDADE_5:_Datos_Persistentes)

Enlace a la página principal del curso (https://wiki.cifprodolfoucha.es/index.php?title=Programación_de_dispositivos_móbiles)

-- Ángel D. Fernández González e Carlos Carrión Álvarez -- (2014).

Esta página se editó por última vez el 17 ene 2021 a las 17:09.