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.

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

Ao facelo, o Eclipse, 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:

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

public class BaseDatos extends SQLiteOpenHelper{

public BaseDatos(Context context, String name, CursorFactory factory,
int version) {
super(context, name, factory, version);
```

```
// TODO Auto-generated constructor stub
}

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

}

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

}
}
```

Analicemos o construtor.

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

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 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:

```
public final static String NOME_BD="basedatos";
public final static int VERSION_BD=1;

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

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 non 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()**: Abre a base de datos para operacións de só lectura.
- **getWritableDatabase()**: 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** que posúe o método **execSQL** para lanzar as ordes SQL necesarias.

Máis información sobre a orde **Create Table**: https://www.sqlite.org/lang_createtable.html

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

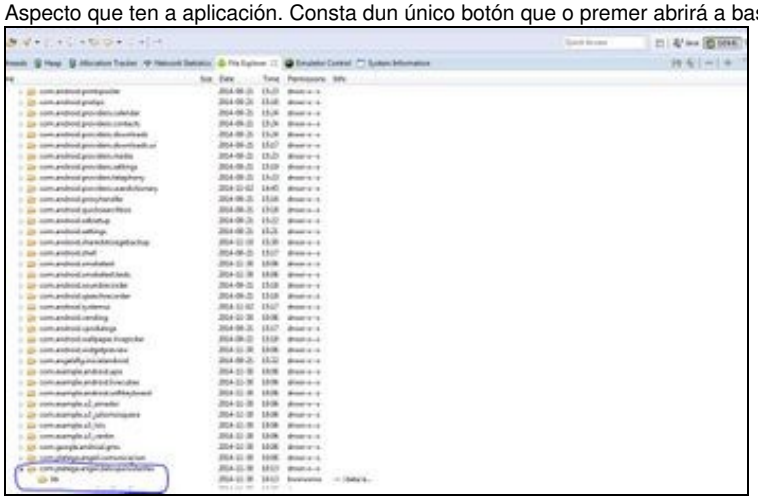
Manual: Caso práctico

Nesta práctica imos crear unha base de datos cunha táboa.

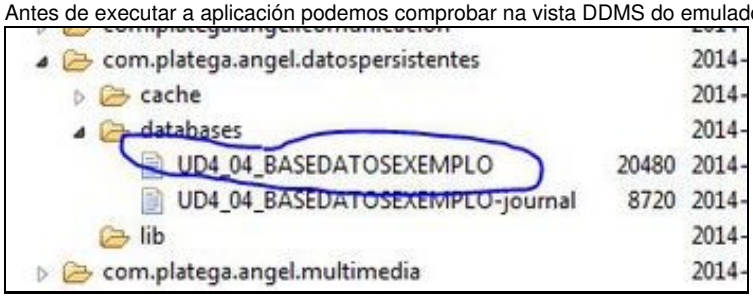
Para facelo teremos que abrir a base de datos en modo lectura ou lectura/escritura como xa comentamos. Neste exemplo vaise abrir en modo lectura/escritura.

- 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 vista DDMS 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 Activity Principal

- Nome do proxecto: **UD1_04_DatosPersistentes_BD**
- Nome da activity: **UD1_04_DatosPersistentes_BD.java**

Código xml do layout

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${relativePackage}.${activityClass}" >

    <Button
        android:id="@+id/UD1_04_btnCrearBD"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="CREAR BASE DE DATOS" />

</RelativeLayout>
```

Código da clase UD1_04_DatosPersistentes_BD

Obxectivo: Crear unha base de datos.

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
```

```
import android.widget.Toast;

#OLLO! hai que cambiar a seguinte liña para importar dende o paquete correcto.
import com.platega.......basedatos.UD1_04_BASEDATOS;

public class UD1_04_DatosPersistentes_BD extends Activity {
private UD1_04_BASEDATOS ud1_04_baseDatos;

private void xestionarEventos(){

Button btnCrearBD = (Button)findViewById(R.id.UD1_04_btnCrearBD);
btnCrearBD.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
// TODO Auto-generated method stub

ud1_04_baseDatos = new UD1_04_BASEDATOS(getApplicationContext());
ud1_04_baseDatos.getWritableDatabase();

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

}

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_ud1_04__datos_persistentes_bd);

xestionarEventos();
}
}
```

- Liña 12: Creamos unha propiedade que vai servir para poder realizar operacións contra a base de datos.
- Liña 23: Instanciamos a propiedade.
- Liña 24: Abrimos a base de datos para operacións de escritura / lectura. É agora cando, se a base de datos non existe, chamará o método onCreate.

Creamos a clase que vai xestionar a BD

Neste exemplo imos crear unha clase de nome **UD1_04_BASEDATOS** que derive da clase SQLiteOpenHelper como explicamos antes.

- O nome da base de datos será **UD1_04_BDEXEMPLO**
- 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 UD1_04_BASEDATOS:

Código da clase UD1_04_BASEDATOS
Obxectivo: Clase que vai executar as ordes para a creación das táboas e operacións contra a base de datos (SELECT, UPDATE, DELETE).

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

public class UD1_04_BASEDATOS extends SQLiteOpenHelper{

public final static String NOME_BD="UD1_04_BDEXEMPLO";
public final static int VERSION_BD=1;

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

```
public UD1_04_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);
}

}
```

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**).

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/

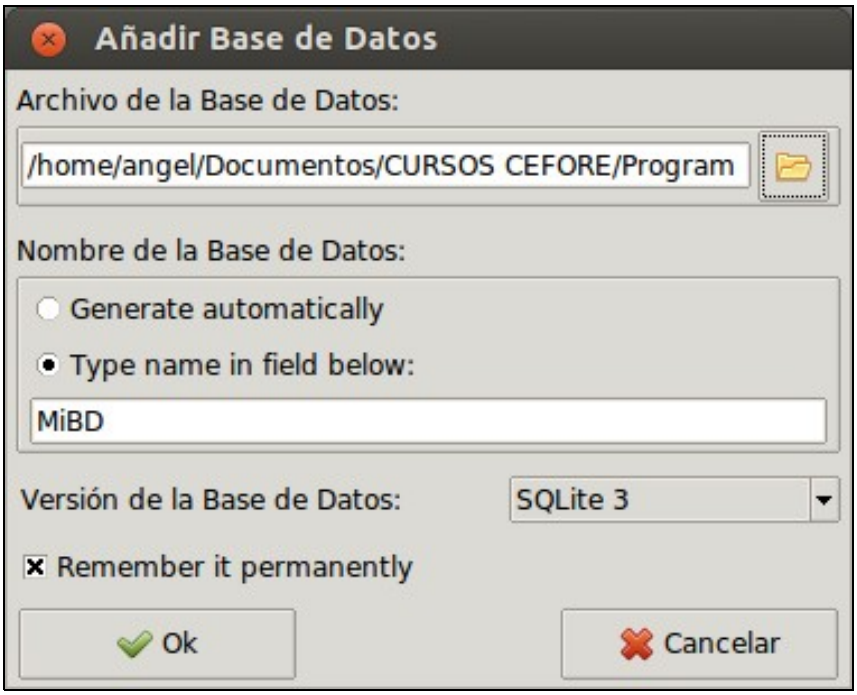
Nota: Se estamos a utilizar Linux deberemos darlle permiso de execución ao arquivo baixado coa orde:

```
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**.

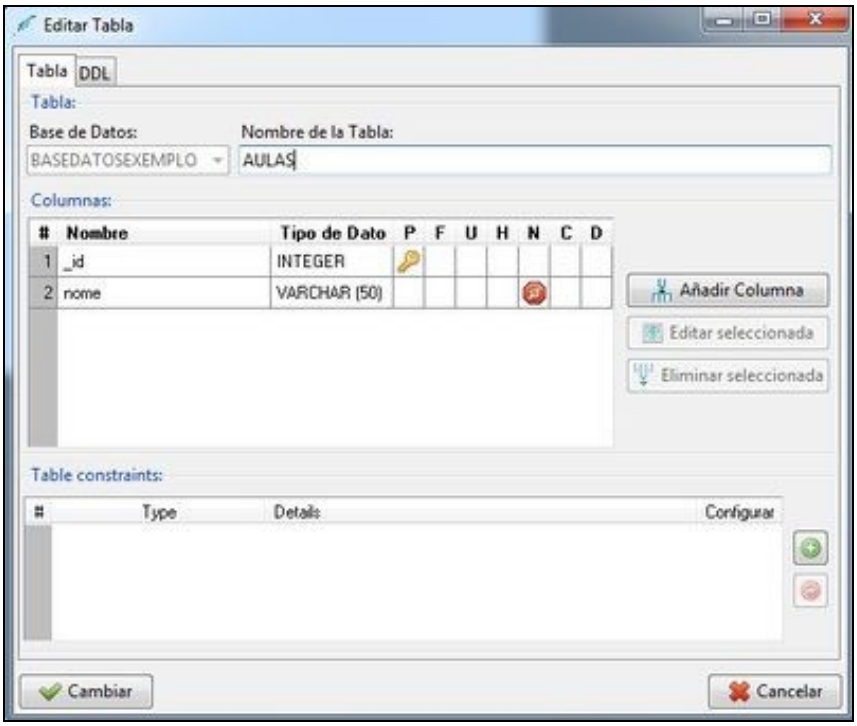


Aquí temos que indicar o sitio físico onde se vai gardar / abrir a base de datos e o nome da mesma.
A versión poñeremos sqlite3.
Debemos premer o botón de 'Ok'.

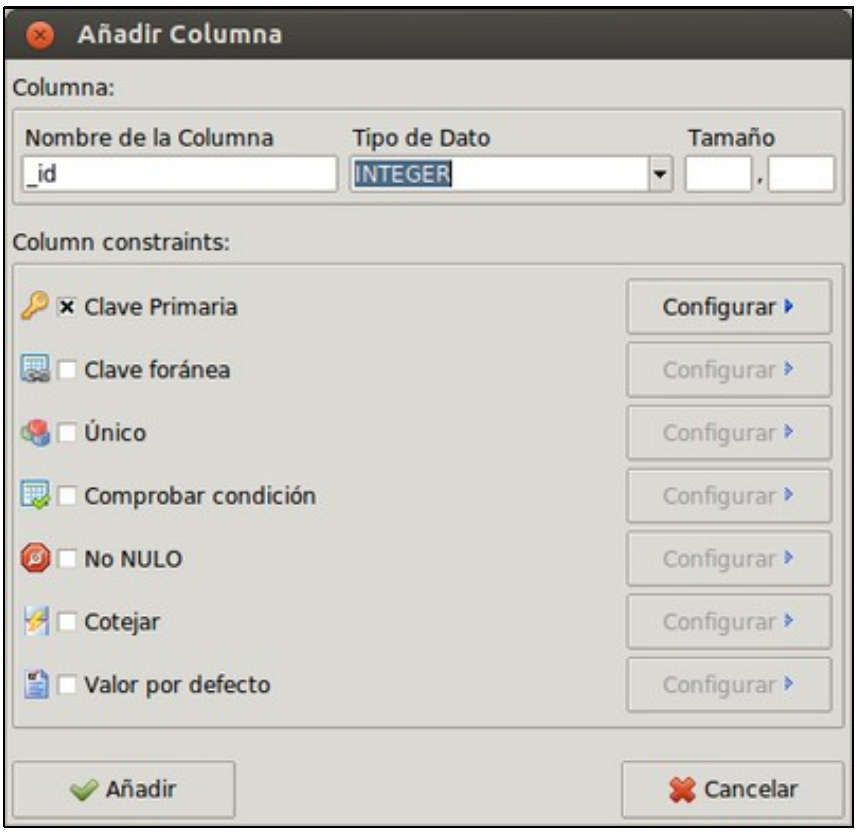
- Agora debemos de escoller a opción de **Tablas** e unha vez escollida premer a opción **Nueva Tabla** da parte superior.

- 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:

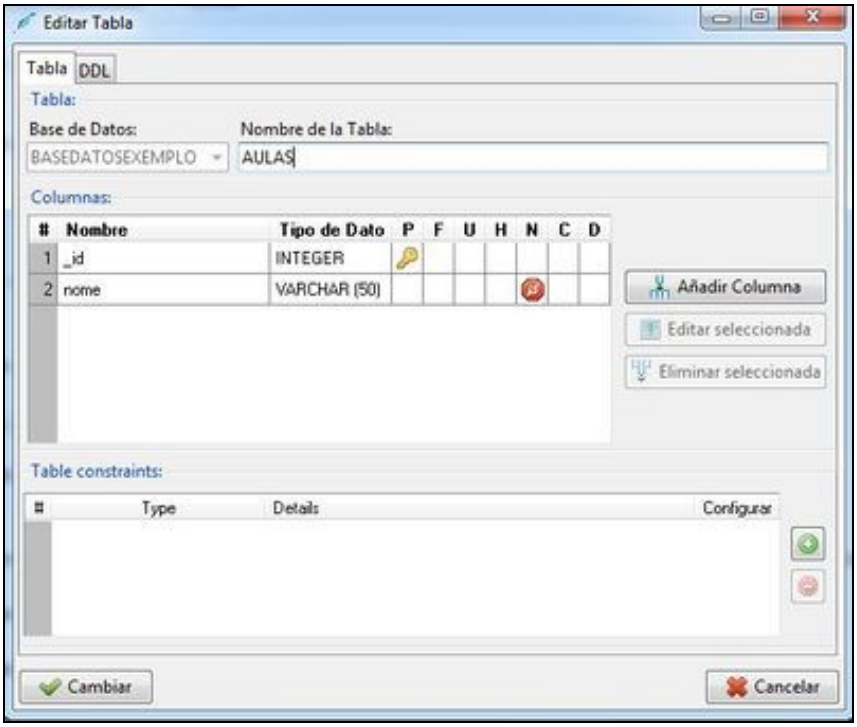


Se prememos na opción configurar podemos facer varias cousas dependendo do que teñamos escollido.

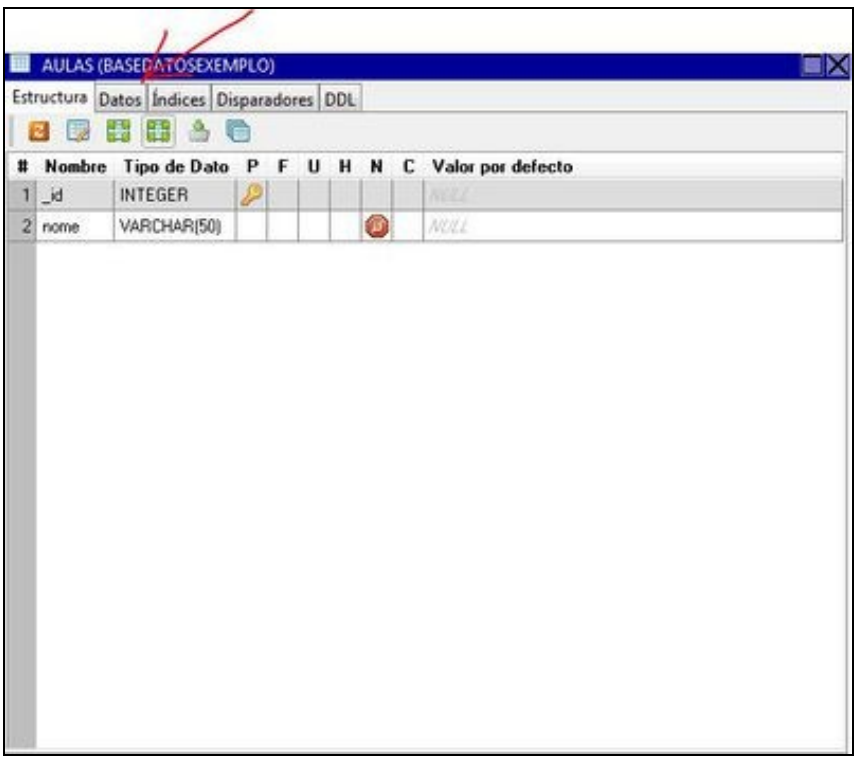
Por exemplo:

◊ 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.

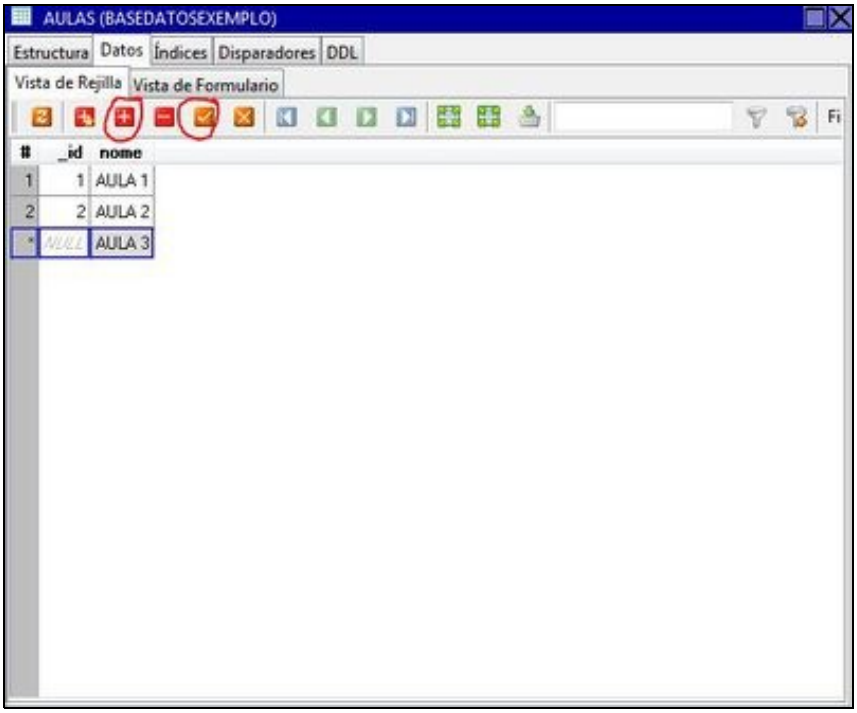
Unha vez temos todos os campo debemos premer o botón de **Crear** (na imaxe aparece cambiar porque xa estaba creada):



- 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.

Normalmente a copia debería facerse nun fío separado do fío principal. O uso dos fíos xa verémolo na unidade 3.

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/:

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

```
String bddestino = "/data/data/" + getPackageName() + "/databases/"
+ UD1_05_BASEDATOS.NOME_BD;
File file = new File(bddestino);
if (file.exists())
return; // XA EXISTE A BASE DE DATOS
```

Copiamos a base de datos.

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

InputStream inputstream;
try {
inputstream = getAssets().open(UD1_05_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();
} catch (IOException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
```

Caso práctico

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

Preparación

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

Media:PDM_UD1_05_BDEXEMPLO.zip

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

Creamos a Activity

- Nome do proxecto: **UD1_05_DatosPersistentes_BD**
- Nome da activity: **UD1_05_DatosPersistentes_BD.java**

Código do layout xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${relativePackage}.${activityClass}" >
```

```
  <Button
        android:id="@+id/UD1_05_btnAbrirBD"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="ABRIR BASE DE DATOS" />


</RelativeLayout>
```

Código da clase UD1_05_DatosPersistentes_BD

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

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

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

import com.platega.angel.datospersistentes.basedatos.UD1_05_BASEDATOS;

public class UD1_05_DatosPersistentes_BD extends Activity {

private UD1_05_BASEDATOS ud1_05_baseDatos;


private void copiarBD() {
String bddestino = "/data/data/" + getPackageName() + "/databases/"
+ UD1_05_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(UD1_05_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 btnAbrirBD = (Button)findViewById(R.id.UD1_05_btnAbrirBD);
btnAbrirBD.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
// TODO Auto-generated method stub


ud1_05_baseDatos = new UD1_05_BASEDATOS(getApplicationContext());
ud1_05_baseDatos.getWritableDatabase();

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

}

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ud1_05__datos_persistentes__bd);

xestionarEventos();
copiarBD();
}
}
```

- Liñas 21-56: Procedemento para copiar a base de datos.
    - ◊ Liñas 25-28: 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 63-73: Xestionamos o evento de click sobre o botón de abrir. Como a base de datos xa está copiada podemos utilizala.

Creamos a clase que xestionará a Base de Datos

Importante: Fixarse como non facemos ningún 'create table' no método onCreate.

Isto é así xa que a base de datos xa vai estar creada no cartafol /assets/

Código da clase UD1_05_BASEDATOS

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

public class UD1_05_BASEDATOS extends SQLiteOpenHelper{

public final static String NOME_BD="UD1_05_BDEXEMPLO";
public final static int VERSION_BD=1;


public UD1_05_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

}

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

}
```

```
}
```

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:

  ◊ adb shell (WINDOWS)
  ◊ ./adb shell (LINUX)
- 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.

Operacións sobre unha base de datos

Introdución

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

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

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.

Podemos obtelo de dúas formas:

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

```
@Override
public void onCreate(SQLiteDatabase db) {
// TODO Auto-generated method stub

}

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

}
```

  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:

```
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:

```
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, 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:

```
ContentValues datosexemplo = new ContentValues();
datosexemplo.put("NOME_COL1", "Valor col1_a");
datosexemplo.put("NOME_COL2", "Valor_col2_a");
long idFila1 = sqlLiteDB.insert(NOME_TABOA, null, datosexemplo);

               datosexemplo.clear();
datosexemplo.put("NOME_COL1", "Valor col1_b");
datosexemplo.put("NOME_COL2", "Valor_col2_b");
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.

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

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

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

Nun exemplo concreto:

TÁBOA AULAS (_id, nome):

Esta consulta sql:

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

Pasará a ser:

```
ContentValues datos = new ContentValues();
datos.put("nome","Novo nome");
String condicionwhere = "_id=?";
String[] parametros = new String[]{String.valueOf(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:

```
String condicionwhere = "COL1=?";
String[] parametros = new String[]{'Valor a buscar'};
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:

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

Pasará a ser:

```
String condicionwhere = "_id=?";
String[] parametros = new String[]{String.valueOf(5)};
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:

  ```
  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**

  ```
  Cursor cursor = sqlLiteDB.query(DATABASE_TABLE,
    new String[] { col1,col2,... },
    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:

```
Cursor cursor = sqlLiteDB.query("AULAS",
  new String[] { nome },
  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).

```
Cursor cursor = sqlLiteDB.rawQuery("select _id,nome from AULAS", null);
  if (cursor.moveToFirst()) {                // Se non ten datos xa non entra
while (!cursor.isAfterLast()) {     // Quédase no bucle ata que remata de percorrer o cursor. Fixarse que leva un ! (not) di
long id =cursor.getLong(0);
                String nome = cursor.getString(1);
cursor.moveToNext();
}
  }
```

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.

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] e asociar dito Adaptador a un elemento gráfico como unha lista.

Por exemplo:

```
SimpleCursorAdapter mAdapter = new SimpleCursorAdapter(this,R.layout.list_layout_creado, cursor, new String[] { "nome" },new
```

Sendo:

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

Lembrar que xa vimos o uso de adaptadores no curso de Iniciación.

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 UD1_06_BASEDATOS extends SQLiteOpenHelper{

public SQLiteDatabase sqlLiteDB;
        ..............
```

- Cando dende a Activity abrimos a base de datos, ao mesmo tempo gardamos a referencia que nos devolve o método **getWritableDatabase()**:

```
public class UD1_06_DatosPersistentes_BD extends Activity {
        ..............
ud1_06_baseDatos = new UD1_06_BASEDATOS(getApplicationContext());
ud1_06_baseDatos.sqlLiteDB = ud1_06_baseDatos.getWritableDatabase();
```

- 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 e o 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**

```
public class Aulas {

private long _id;
private String nome;

public Aulas (long id, String nome){
this._id=id;
this.nome=nome;
}

public long get_id() {
return _id;
}
```

```
public void set_id(long _id) {
this._id = _id;
}
public String getNome() {
return nome;
}
public void setNome(String nome) {
this.nome = nome;
}

public String toString(){
return nome;
}
}
```

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.

```
public long engadirAula(Aulas aula_engadir){
ContentValues valores = new ContentValues();
valores.put("nome", aula_engadir.getNome());
long id = db.insert("AULAS",null,valores);

return id;
}
```

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.

```
public ArrayList<Aulas> obterAulas() {
ArrayList<Aulas> aulas_devolver = new ArrayList<Aulas>();

Cursor datosConsulta = sqlLiteDB.rawQuery(CONSULTAR_AULAS, null);
if (datosConsulta.moveToFirst()) {
Aulas aula;
while (!datosConsulta.isAfterLast()) {
aula = new Aulas(datosConsulta.getLong(0),
datosConsulta.getString(1));
aulas_devolver.add(aula);
datosConsulta.moveToNext();
}
}
return aulas_devolver;
```

```
}
```

Caso práctico

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

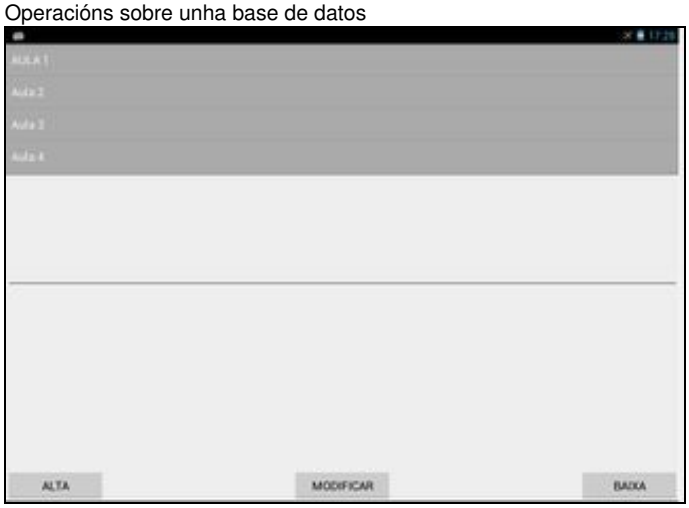
Dita BD consta dunha única táboa de nome **AULAS** cos seguintes campos:

- **_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:PDM_UD1_06_BDEXEMPLO.zip

Creamos a activity

- Nome do proxecto: **UD1_06_DatosPersistentes_BD**
- Nome da activity: **UD1_06_DatosPersistentes_BD.java**

Código do layout xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${relativePackage}.${activityClass}" >

    <ListView
        android:id="@+id/UD1_06_lstAulas"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:background="@android:color/darker_gray"
        android:choiceMode="singleChoice"
        android:listSelector="#666666"
        android:scrollbars="vertical"
        android:scrollbarStyle="outsideOverlay"
        />


    <EditText
        android:id="@+id/UD1_06_editNomeAula"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:inputType="text"
        android:textSize="20sp"
        />

    <Button
        android:id="@+id/UD1_06_btnAltaAula"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:width="150dp"
        android:gravity="center_horizontal"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:text="ALTA"
        />

    <Button
        android:id="@+id/UD1_06_btnModificarAula"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:width="150dp"
        android:gravity="center_horizontal"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:text="MODIFICAR" />

    <Button
        android:id="@+id/UD1_06_btnBaixaAula"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:width="150dp"
        android:gravity="center_horizontal"
        android:layout_alignParentBottom="true"
        android:layout_alignParentRight="true"
        android:text="BAIXA" />

</RelativeLayout>
```

Código da clase UD1_06_DatosPersistentes_BD

Obxectivo: Realizar operacións contra unha base de datos.

```
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 android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;
#OLLO! Ao importar as clases seguintes
import com......basedatos.Aulas;
import com......basedatos.UD1_06_BASEDATOS;

public class UD1_06_DatosPersistentes_BD extends Activity {
private UD1_06_BASEDATOS ud1_06_baseDatos;
private Aulas aula_seleccionada=null;// Garda a aula seleccionada da lista

private void copiarBD() {
String bddestino = "/data/data/" + getPackageName() + "/databases/"
+ UD1_06_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(UD1_06_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.UD1_06_btnAltaAula);
 btnAltaAula.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
// TODO Auto-generated method stub
EditText editAula = (EditText) findViewById(R.id.UD1_06_editNomeAula);
// Habería que comprobar se hai algún dato na caixa de texto, pero neste exemplo centrámonos no funcionamento da base de datos.

Aulas aula = new Aulas(0, editAula.getText().toString());
long id = ud1_06_baseDatos.engadirAula(aula); // Obtemos o id. Neste exemplo non faremos nada con el.
aula.set_id(id);

// Poderíamos engadir a nova aula ó adaptador pero neste exemplo
// recargamos a lista
cargarLista();
editAula.setText("");

Toast.makeText(getApplicationContext(), "AULA ENGADIDA",Toast.LENGTH_LONG).show();
}
});

 Button btnBaixaAula = (Button)findViewById(R.id.UD1_06_btnBaixaAula);
 btnBaixaAula.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
// TODO Auto-generated method stub

ud1_06_baseDatos.borrarAula(aula_seleccionada);
EditText editAula = (EditText)findViewById(R.id.UD1_06_editNomeAula);
editAula.setText("");

// Poderíamos borrar a aula do adaptador e non cargar a lista novamente
cargarLista();
Toast.makeText(getApplicationContext(), "AULA BORRADA",Toast.LENGTH_LONG).show();
}
});

 Button btnModificaraAula = (Button)findViewById(R.id.UD1_06_btnModificarAula);
 btnModificaraAula.setOnClickListener(new OnClickListener() {

@Override
public void onClick(View v) {
// TODO Auto-generated method stub
```

```
EditText editAula = (EditText)findViewById(R.id.UD1_06_editNomeAula);

Aulas aula_modificar = new Aulas(aula_seleccionada.get_id(),editAula.getText().toString());
ud1_06_baseDatos.modificarAula(aula_modificar);

// Poderíamos borrar a aula do adaptador e non cargar a lista novamente
cargarLista();
Toast.makeText(getApplicationContext(), "AULA MODIFICADA",Toast.LENGTH_LONG).show();
}
});



ListView lista = (ListView)findViewById(R.id.UD1_06_lstAulas);
lista.setOnItemClickListener(new OnItemClickListener() {

@Override
public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
long arg3) {
// TODO Auto-generated method stub
// Obtemos o obxecto aula seleccionado
ArrayAdapter<Aulas> adaptador = (ArrayAdapter<Aulas>) arg0.getAdapter();
aula_seleccionada = adaptador.getItem(arg2);
EditText editAula = (EditText)findViewById(R.id.UD1_06_editNomeAula);
editAula.setText(aula_seleccionada.getNome());

}
});




}

/**
         * Accede á base de datos para obter as aulas e as asina á lista.
         */
private void cargarLista(){

ListView lista = (ListView)findViewById(R.id.UD1_06_lstAulas);

ArrayList<Aulas> aulas = ud1_06_baseDatos.obterAulas();
ArrayAdapter<Aulas> adaptador = new ArrayAdapter<Aulas>(getApplicationContext(),
android.R.layout.simple_list_item_1,aulas);
lista.setAdapter(adaptador);

}


@Override
public void onStart(){
super.onStart();

if (ud1_06_baseDatos==null) {// Abrimos a base de datos para escritura
ud1_06_baseDatos = new UD1_06_BASEDATOS(this);
ud1_06_baseDatos.sqlLiteDB = ud1_06_baseDatos.getWritableDatabase();

cargarLista();
}
}

@Override
public void onStop(){
super.onStop();

if (ud1_06_baseDatos!=null){// Pechamos a base de datos.
ud1_06_baseDatos.close();
ud1_06_baseDatos=null;
}

}

@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_ud1_06__datos_persistentes__bd);

copiarBD();
xestionarEventos();
}
}
```

- Liñas 69-86: Xestionamos o evento Click sobre o botón de alta.
    - ◊ Liña 75: 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 76: Chamamos á base de datos có aula a dar de alta. Devolve o id (autonumérico).
    - ◊ Liña 81: Chamamos a o método que volve chamar á base de datos para cargar as aulas novamente.
- Liñas 88-103: Xestionamos o evento Click sobre o botón de baixa.
    - ◊ Liña 95: 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 100: Chamamos a o método que volve chamar á base de datos para cargar as aulas novamente.
- Liñas 105-121: Xestionamos o evento Click sobre o botón de modificar.
    - ◊ Liña 114: 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 115: Chamamos á base de datos para modificar o nome da aula seleccionada.
    - ◊ Liña 118: Chamamos a o método que volve chamar á base de datos para cargar as aulas novamente.
- Liñas 125-139: Xestionamos o evento Click sobre a lista.
    - ◊ Liña 134: Gardamos na propiedade 'aula_seleccionada' a aula seleccionada.
    - ◊ Liña 136: Cambiamos o texto do EditText polo seleccionado.
- Liñas 149-158: Cargamos as aulas dende a base de datos á lista.
- Liñas 165-170: Cando a aplicación empeza abrimos a base de datos e cargamos a lista coas aulas.
- Liñas 177-180: Liberamos os recursos.

Creamos a clase que xestiona a base de datos

Nome da clase: UD1_06_BASEDATOS

Código da clase que xestiona a base de datos.

```
import java.util.ArrayList;

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

public class UD1_06_BASEDATOS extends SQLiteOpenHelper{

public SQLiteDatabase sqlLiteDB;

public final static String NOME_BD="UD1_06_BDEXEMPLO";
public final static int VERSION_BD=1;

private final String CONSULTAR_AULAS ="SELECT _id,nome FROM AULAS order by nome";
private final String TABOA_AULAS="AULAS";


public UD1_06_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

}

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

}

public long engadirAula(Aulas aula_engadir){
ContentValues valores = new ContentValues();
valores.put("nome", aula_engadir.getNome());
long id = sqlLiteDB.insert("AULAS",null,valores);

return id;

}

public int borrarAula(Aulas aula){
    String condicionwhere = "_id=?";
    String[] parametros = new String[]{String.valueOf(aula.get_id())};
    int rexistrosafectados = sqlLiteDB.delete(TABOA_AULAS,condicionwhere,parametros);

    return rexistrosafectados;

}

public int modificarAula(Aulas aula_modificar){
ContentValues datos = new ContentValues();
datos.put("nome", aula_modificar.getNome());

String where = "_id=?";
String[] params = new String[]{String.valueOf(aula_modificar.get_id())};

int rexistrosModificados = sqlLiteDB.update(TABOA_AULAS, datos, where, params);

return rexistrosModificados;
}


public ArrayList<Aulas> obterAulas() {
ArrayList<Aulas> aulas_devolver = new ArrayList<Aulas>();

Cursor datosConsulta = sqlLiteDB.rawQuery(CONSULTAR_AULAS, null);
if (datosConsulta.moveToFirst()) {
Aulas aula;
while (!datosConsulta.isAfterLast()) {
aula = new Aulas(datosConsulta.getLong(0),
datosConsulta.getString(1));
aulas_devolver.add(aula);
datosConsulta.moveToNext();
}
}

return aulas_devolver;
}

}
```

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