

	UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN	
CICLO: I/2022	GUIA DE LABORATORIO #5 - Parte I	
	Nombre de la Práctica:	Guía #5 - Almacenamiento de datos SQLite
	Materia:	Desarrollo de Software para Móviles

I. OBJETIVOS

1. Implementar un ejemplo en SQLite
2. Conocer los beneficios de usar SQLite en Android

II. INTRODUCCION TEORICA

¿QUÉ ES SQLite?

<https://www.sqlite.org/index.html>

Es un motor de base de datos SQL transaccional de código abierto, ligero, autónomo, de configuración simple y sin servidor, que se caracteriza por almacenar información persistente de forma sencilla, SQLite gracias a sus características se diferencia de otros gestores de bases de datos, proporcionando grandes ventajas sobre ellos.

Así mismo, por ser de dominio público es gratuito tanto para fines privados como para comerciales, se puede descargar de forma libre. Es importante mencionar que SQLite cuenta con varios enlaces a lenguajes de programación entre los que podemos destacar: Java, C, C ++, JavaScript, C #, Python, VB Script, entre otros.

En cuanto a los tipos de datos que maneja **SQLite**, destacaremos los más utilizados:

- Int.
- Varchar.
- Blob.
- Real.
- Double.
- Float.
- Text.
- Boolean.
- Date.

SQLite cumple con las características **ACID** (atomicidad, consistencia, aislamiento y durabilidad), forma parte integral de las aplicaciones basadas en el cliente, SQLite utiliza una sintaxis SQL dinámica y realiza múltiples tareas para hacer lecturas y escrituras al mismo tiempo, ambas (lectura y escritura) se efectúan directamente en los archivos de disco ordinarios.

Para reducir la latencia se cuenta con una biblioteca SQLite, la cual es llamada dinámicamente a través de funciones simples y los programas de la aplicación utilizan esta funcionalidad, de igual forma implementa el estándar SQL-92 y usa un sistema inusual para sistemas de administración de bases de datos compatibles con SQL.

Algunas de las ventajas que brinda son:

Los paquetes `android.database` y `android.database.sqlite` ofrecen una alternativa de mayor rendimiento donde la compatibilidad de la fuente no representa mayor problema, aprovechando los recursos.

Es ideal para consultar y almacenar datos de forma estructurada.

La aplicación solo tiene que cargar tantos datos como necesite, en lugar de leer todo el archivo de la aplicación y mantener un análisis completo en la memoria, por ende, el tiempo de inicio y el consumo de memoria se reducen.

El contenido se actualiza de forma continua y atómica, para que no se pierda el trabajo en caso de una falla de energía o algún bloqueo.

Se puede acceder al contenido y actualizarlo mediante potentes consultas SQL, lo que reduce en gran medida la complejidad del código de la aplicación.

Las aplicaciones pueden aprovechar la búsqueda de texto completo y las capacidades de RTREE integradas en SQLite.

Una gran cantidad de programas, escritos en diferentes lenguajes de programación, puede acceder al mismo archivo de aplicación sin problemas de compatibilidad.

Los problemas de rendimiento a menudo se pueden resolver utilizando `CREATE INDEX` en lugar de rediseñar, reescribir y volver a probar el código de la aplicación.

Las bases de datos creadas en Android solo son visibles para la aplicación que las creó.

Por su usabilidad SQLite permite que el desarrollo sea más simple, convirtiéndose en una práctica tecnología para los dispositivos móviles. Las preferencias que nos

ofrece, permiten almacenar datos de forma puntual como, por ejemplo: el usuario, la clave, la fecha y la hora de su última conexión, el idioma, entre otros.

IV. PROCEDIMIENTO

Almacenamiento en una base de datos SQLite

Repositorio: <https://github.com/AlexanderSiguenza/SQLiteApp.git>

Ejemplo: Confeccionar un programa que permita almacenar los datos de artículos. Crear la tabla artículos y definir los campos código, descripción del artículo y precio.

El programa debe permitir:

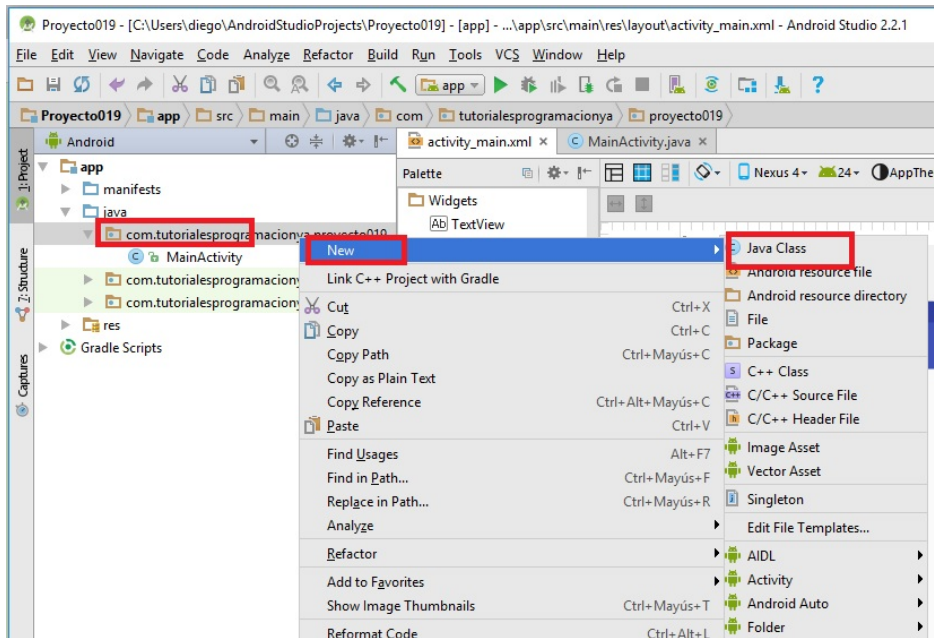
- 1 - Carga de artículo.
- 2 - Consulta por el código.
- 3 - Consulta por la descripción.
- 4 - Borrado de un artículo ingresando su código.
- 4 - Modificación de la descripción y el precio.

Crear un proyecto en Android Studio y definir como nombre: **SQLiteApp**

Lo primero que haremos es crear una clase que herede de **SQLiteOpenHelper**. Esta clase nos permite crear la base de datos y actualizar la estructura de tablas y datos iniciales.

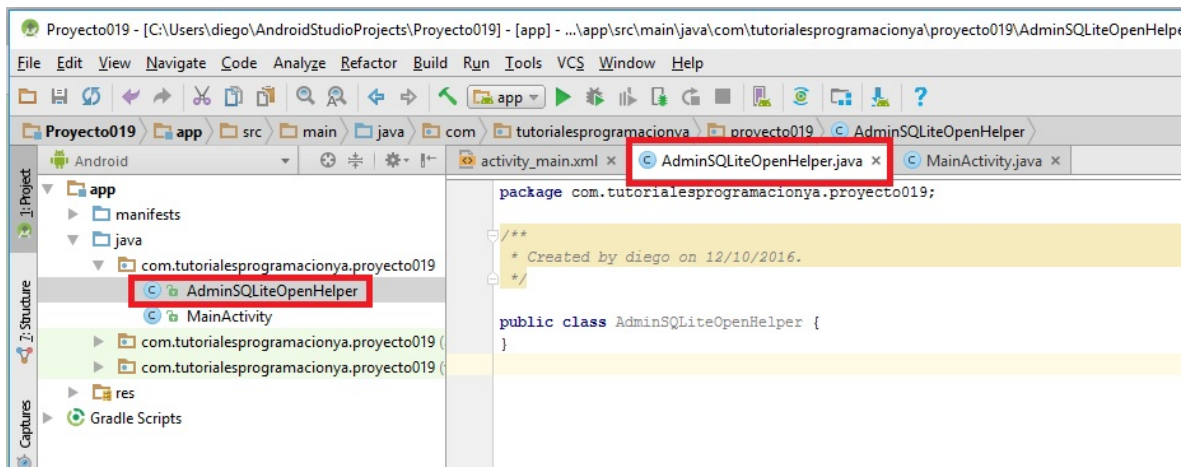
Para crear una nueva clase desde Android Studio procedemos a presionar el botón derecho del mouse sobre la carpeta que contienen todos los archivos javos del proyecto y seleccionamos New - > Java Class:

Guía #5 - Almacenamiento de datos SQLite y Content Providers



En este diálogo ingresamos el nombre de nuestra clase, en nuestro ejemplo la llamaremos **AdminSQLiteOpenHelper**:

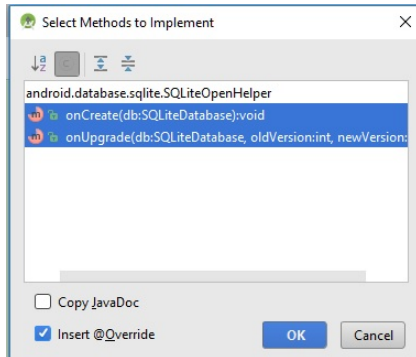
Ya tenemos un nuevo archivo en nuestro proyecto:



Ahora tenemos que codificar esta clase que tiene por objetivo administrar la base de datos que crearemos. Primero hacemos que nuestra clase herede de la clase SQLiteOpenHelper:

```
import android.database.sqlite.SQLiteOpenHelper;
public class AdminSQLiteOpenHelper extends SQLiteOpenHelper{
}
```

La clase `SQLiteOpenHelper` requiere que se implementen dos métodos obligatoriamente **onCreate** y **onUpgrade**, podemos hacer que el Android Studio nos ayude en su codificación así no tenemos que tipearlos nosotros: presionamos las teclas "ALT" y "ENTER" teniendo el cursor sobre el nombre de la clase "`AdminSQLiteOpenHelper`" y nos aparece un menú donde seleccionamos "Implement Methods" y luego un diálogo donde seleccionamos los métodos a implementar:



Ya tenemos la clase con los dos métodos:

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

public class AdminSQLiteOpenHelper extends SQLiteOpenHelper{
    @Override
    public void onCreate(SQLiteDatabase db) {
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

Pero todavía nos marca un error ya que la clase padre `SQLiteOpenHelper` implementa constructores que tienen parámetros, entonces tenemos que definir el constructor en esta clase, para ello estando el cursor sobre el nombre de la clase "`AdminSQLiteOpenHelper`" presionamos nuevamente las teclas "ALT" y "Enter" y en el menú contextual que aparece seleccionamos "Create constructor matching super". Aparece un diálogo con todos los constructores que implementa la clase padre (seleccionamos el que tiene tres parámetros):

Crear Clase en Android Studio

Ahora la clase no muestra ningún error y pasaremos a implementar la creación de la tabla artículos dentro del método onCreate:

```
package edu.udb.sqliteapp;

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

public class AdminSQLiteOpenHelper extends SQLiteOpenHelper {

    public AdminSQLiteOpenHelper(Context context, String name,
        SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

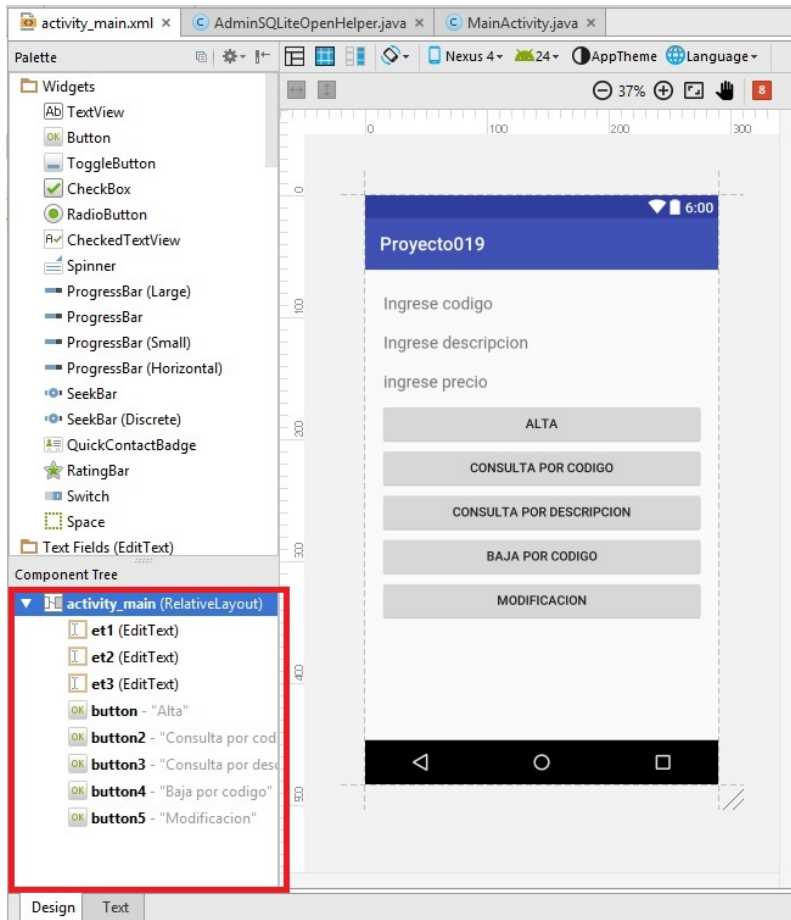
    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("create table articulos(codigo int primary key,descripcion
        text,precio real)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

Hemos codificado en el método **onCreate** la creación de la tabla artículos con los campos código (que es entero y clave primaria), descripción que es de tipo texto y precio es un valor real.

El método **onCreate** se ejecutará una única vez (Eventualmente si uno quiere modificar la estructura de la tabla debemos hacerlo en el método **onUpgrade**).

Ahora implementemos la interfaz visual para resolver nuestro problema. Debemos crear en nuestro archivo **activity_main.xml** la siguiente interfaz:



Como vemos disponemos tres EditText y cinco Button:

EditText de tipo "Number" (ID="et1", hint="Ingrese codigo")

EditText de tipo "Plain Text" (ID="et2", hint="Ingrese descripcion")

EditText de tipo "Number Decimal" (ID="et3", hint="Ingrese precio")

Button (ID="button", text="Alta", onClick="alta")

Button (ID="button2", text="Consulta por codigo",
onClick="consultaporcodigo")

Button (ID="button3", text="Consulta por descripcion",
onClick="consultapordescrpcion")

Button (ID="button4", text="Baja por codigo", onClick="bajaporcodigo")

Button (ID="button5", text="Modificacion", onClick="modificacion")

El código fuente de nuestro **activity_main.xml** es el siguiente:

```
package edu.udb.sqliteapp;

import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private EditText et1,et2,et3;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        et1=(EditText)findViewById(R.id.et1);
        et2=(EditText)findViewById(R.id.et2);
        et3=(EditText)findViewById(R.id.et3);
    }

    public void alta(View v) {
        AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
        SQLiteDatabase bd = admin.getWritableDatabase();
        String cod = et1.getText().toString();
        String descri = et2.getText().toString();
        String pre = et3.getText().toString();
        ContentValues registro = new ContentValues();
        registro.put("codigo", cod);
        registro.put("descripcion", descri);
        registro.put("precio", pre);
        bd.insert("articulos", null, registro);
        bd.close();
        et1.setText("");
        et2.setText("");
        et3.setText("");
        Toast.makeText(this, "Se cargaron los datos del
artículo",Toast.LENGTH_SHORT).show();
    }
}
```



```

public void consultaporcodigo(View v) {
    AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String cod = et1.getText().toString();
    Cursor fila = bd.rawQuery("select descripcion,precio from articulos where
codigo=" + cod, null);
    if (fila.moveToFirst()) {
        et2.setText(fila.getString(0));
        et3.setText(fila.getString(1));
    } else
        Toast.makeText(this, "No existe un artículo con dicho código",
            Toast.LENGTH_SHORT).show();
    bd.close();
}

public void consultapordescripcion(View v) {
    AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String descri = et2.getText().toString();
    Cursor fila = bd.rawQuery("select codigo,precio from articulos where
descripcion=" + descri + "'", null);
    if (fila.moveToFirst()) {
        et1.setText(fila.getString(0));
        et3.setText(fila.getString(1));
    } else
        Toast.makeText(this, "No existe un artículo con dicha descripción",
            Toast.LENGTH_SHORT).show();
    bd.close();
}

public void bajaporcodigo(View v) {
    AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String cod= et1.getText().toString();
    int cant = bd.delete("articulos", "codigo=" + cod, null);
    bd.close();
    et1.setText("");
    et2.setText("");
    et3.setText("");
    if (cant == 1)
        Toast.makeText(this, "Se borró el artículo con dicho código",
            Toast.LENGTH_SHORT).show();
}

```

```

        else
            Toast.makeText(this, "No existe un artículo con dicho código",
                Toast.LENGTH_SHORT).show();
    }

    public void modificacion(View v) {
        AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
        SQLiteDatabase bd = admin.getWritableDatabase();
        String cod = et1.getText().toString();
        String descri = et2.getText().toString();
        String pre = et3.getText().toString();
        ContentValues registro = new ContentValues();
        registro.put("codigo", cod);
        registro.put("descripcion", descri);
        registro.put("precio", pre);
        int cant = bd.update("articulos", registro, "codigo=" + cod, null);
        bd.close();
        if (cant == 1)
            Toast.makeText(this, "se modificaron los datos", Toast.LENGTH_SHORT)
                .show();
        else
            Toast.makeText(this, "no existe un artículo con el código ingresado",
                Toast.LENGTH_SHORT).show();
    }
}

```

Recordar de inicializar la propiedad onClick de cada botón para enlazarlo con el método respectivo:

"alta","consultaporcodigo","consultapordescripcion","bajaporcodigo" y "modificación".

Alta de datos.

Cuando se presiona el botón "ALTA" se ejecuta el método "alta" recordemos inicializar la propiedad "onClick" del botón desde la ventana de visualización del archivo XML.

Lo primero que hacemos en este método es crear un objeto de la clase que planteamos anteriormente y le pasamos al constructor this (referencia del Activity actual), "administración" (es el nombre de la base de datos que crearemos en el caso que no exista) luego pasamos null y un uno indicando que

es la primer versión de la base de datos (en caso que cambiemos la estructura o agreguemos tablas por ejemplo podemos pasar un dos en lugar de un uno para que se ejecute el método `onUpgrade` donde indicamos la nuestra estructura de la base de datos)

Luego de crear un objeto de la clase `AdminSQLiteOpenHelper` procedemos a crear un objeto de la clase `SQLiteDatabase` llamando al método `getWritableDatabase` (la base de datos se abre en modo lectura y escritura). Creamos un objeto de la clase `ContentValues` y mediante el método `put` inicializamos todos los campos a cargar.

Seguidamente llamamos al método `insert` de la clase `SQLiteDatabase` pasando en el primer parámetro el nombre de la tabla, como segundo parámetro un `null` y por último el objeto de la clase `ContentValues` ya inicializado (este método es el que provoca que se inserte una nueva fila en la tabla artículos en la base de datos llamada administración)

Borramos seguidamente los `EditText` y mostramos un mensaje para que conozca el operador que el alta de datos se efectuó en forma correcta:

```
public void alta(View v) {
    AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String cod = et1.getText().toString();
    String descri = et2.getText().toString();
    String pre = et3.getText().toString();
    ContentValues registro = new ContentValues();
    registro.put("codigo", cod);
    registro.put("descripcion", descri);
    registro.put("precio", pre);
    bd.insert("articulos", null, registro);
    bd.close();
    et1.setText("");
    et2.setText("");
    et3.setText("");
    Toast.makeText(this, "Se cargaron los datos del
artículo",Toast.LENGTH_SHORT).show();
}
```

Consulta de artículo por código.

Cuando se presiona el botón "CONSULTA POR CODIGO" se ejecuta el método `consultaporcodigo`:

```
public void consultaporcodigo(View v) {
    AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String cod = et1.getText().toString();
    Cursor fila = bd.rawQuery("select descripcion,precio from articulos where
codigo=" + cod, null);
    if (fila.moveToFirst()) {
        et2.setText(fila.getString(0));
        et3.setText(fila.getString(1));
    } else
        Toast.makeText(this, "No existe un artículo con dicho código",
            Toast.LENGTH_SHORT).show();
    bd.close();
}
```

En el método consulta por código lo primero que hacemos es crear un objeto de la clase `AdminSQLiteOpenHelper` y obtener una referencia de la base de datos llamando al método **`getWritableDatabase`**. Seguidamente definimos una variable de la clase `Cursor` y la inicializamos con el valor devuelto por el método llamado `rawQuery`.

La clase `Cursor` almacena en este caso una fila o cero filas (una en caso que hayamos ingresado un código existente en la tabla `articulos`), llamamos al método `moveToFirst()` de la clase `Cursor` y retorna `true` en caso de existir un artículo con el código ingresado, en caso contrario retorna `cero`.

Para recuperar los datos propiamente dichos que queremos consultar llamamos al método `getString` y le pasamos la posición del campo a recuperar (comienza a numerarse en cero, en este ejemplo la columna `cero` representa el campo descripción y la columna `1` representa el campo precio)

Consulta de artículo por descripción.

Cuando se presiona el botón "CONSULTA POR DESCRIPCION" se ejecuta el método consulta por descripción:

```
public void consultapordescripcion(View v) {
    AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String descri = et2.getText().toString();
    Cursor fila = bd.rawQuery("select codigo,precio from articulos where
descripcion=" + descri + "'", null);
    if (fila.moveToFirst()) {
        et1.setText(fila.getString(0));
        et3.setText(fila.getString(1));
    } else
        Toast.makeText(this, "No existe un artículo con dicha descripción",
            Toast.LENGTH_SHORT).show();
    bd.close();
}
```

En el método consulta por descripción lo primero que hacemos es crear un objeto de la clase AdminSQLiteOpenHelper y obtener una referencia de la base de datos llamando al método getWritableDatabase.

Seguidamente definimos una variable de la clase Cursor y la inicializamos con el valor devuelto por el método llamado rawQuery.

Es importante notar en el where de la cláusula SQL hemos dispuesto comillas simples entre el contenido de la variable descri:

```
"select codigo,precio from articulos where descripcion=" + descri + "'", null);
```

Esto es obligatorio para los campos de tipo text (en este caso descripcion es de tipo text)

Baja o borrado de datos.

Para borrar uno o más registros la clase SQLiteDatabase tiene un método que le pasamos en el primer parámetro el nombre de la tabla y en el segundo la condición que debe cumplirse para que se borre la fila de la tabla. El método delete retorna un entero que indica la cantidad de registros borrados:

```
public void bajaporcodigo(View v) {
    AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String cod= et1.getText().toString();
    int cant = bd.delete("articulos", "codigo=" + cod, null);
    bd.close();
    et1.setText("");
    et2.setText("");
    et3.setText("");
    if (cant == 1)
        Toast.makeText(this, "Se borró el artículo con dicho código",
            Toast.LENGTH_SHORT).show();
    else
        Toast.makeText(this, "No existe un artículo con dicho código",
            Toast.LENGTH_SHORT).show();
}
```

Modificación de datos.


En la modificación de datos debemos crear un objeto de la clase ContentValues y mediante el método put almacenar los valores para cada campo que será modificado. Luego se llama al método update de la clase SQLiteDatabase pasando el nombre de la tabla, el objeto de la clase ContentValues y la condición del where (el cuanto parámetro en este ejemplo no se lo emplea)

```
public void modificacion(View v) {
    AdminSQLiteOpenHelper admin = new
AdminSQLiteOpenHelper(this,"administracion", null, 1);
    SQLiteDatabase bd = admin.getWritableDatabase();
    String cod = et1.getText().toString();
    String descri = et2.getText().toString();
    String pre = et3.getText().toString();
    ContentValues registro = new ContentValues();
    registro.put("codigo", cod);
    registro.put("descripcion", descri);
    registro.put("precio", pre);
}
```

```
int cant = bd.update("articulos", registro, "codigo=" + cod, null);
bd.close();
if (cant == 1)
    Toast.makeText(this, "se modificaron los datos", Toast.LENGTH_SHORT)
        .show();
else
    Toast.makeText(this, "no existe un artículo con el código ingresado",
        Toast.LENGTH_SHORT).show();
}
```

Cuando ejecutamos el programa tenemos la siguiente interfaz:



	<p align="center">UNIVERSIDAD DON BOSCO FACULTAD DE INGENIERÍA ESCUELA DE COMPUTACIÓN</p>	
<p>CICLO: I/2022</p>	<p align="center">GUIA DE LABORATORIO #5 – Parte II</p>	
	<p>Nombre de la Práctica:</p>	<p>Content Providers</p>
	<p>MATERIA:</p>	<p>Desarrollo de Software para Móviles</p>

I. OBJETIVOS

- a. Conocer qué es un content provider.
- b. Crear un content provider en una aplicación Android.
- c. Consumir un content provider en una aplicación Android.

II. INTRODUCCION TEORICA

Un proveedor de contenido (content provider) administra el acceso a un repositorio central de datos. Un proveedor forma parte de una aplicación para Android y a menudo proporciona su propia IU para trabajar con los datos. No obstante, los proveedores de contenido están principalmente orientados a que los usen otras aplicaciones que acceden al proveedor usando un objeto de cliente del proveedor. Juntos, los proveedores y clientes de proveedores ofrecen una interfaz estándar y uniforme para los datos que también manipula la comunicación dentro del proceso y el acceso seguro a los datos.

Por lo general, son dos las situaciones en las que se trabaja con proveedores de contenido: cuando quieres implementar código para acceder a un proveedor de contenido existente en otra aplicación o cuando decides crear un proveedor de contenido nuevo en tu aplicación a fin de compartir datos con otras aplicaciones.

Un proveedor de contenido presenta datos a aplicaciones externas en forma de una o más tablas que son similares a las tablas de una base de datos relacional. Una fila representa una instancia de algún tipo de datos que recopila el proveedor, y cada columna de la fila representa un ítem individual de los datos recopilados para una instancia.

El proveedor de contenido organiza el acceso a la capa de almacenamiento de los datos en tu aplicación para una serie de API y componentes diferentes (como se detalla en la figura 1) e incluye lo siguiente:

1. Compartir con otras aplicaciones el acceso a los datos de tu aplicación.
2. Enviar datos a un widget.
3. Mostrar sugerencias personalizadas de búsqueda para tu aplicación mediante el marco de trabajo de búsqueda usando SearchRecentSuggestionsProvider.
4. Sincronizar los datos de la aplicación con tu servidor mediante una implementación

de `AbstractThreadedSyncAdapter`.

5. Cargar datos en tu IU usando `CursorLoader`.

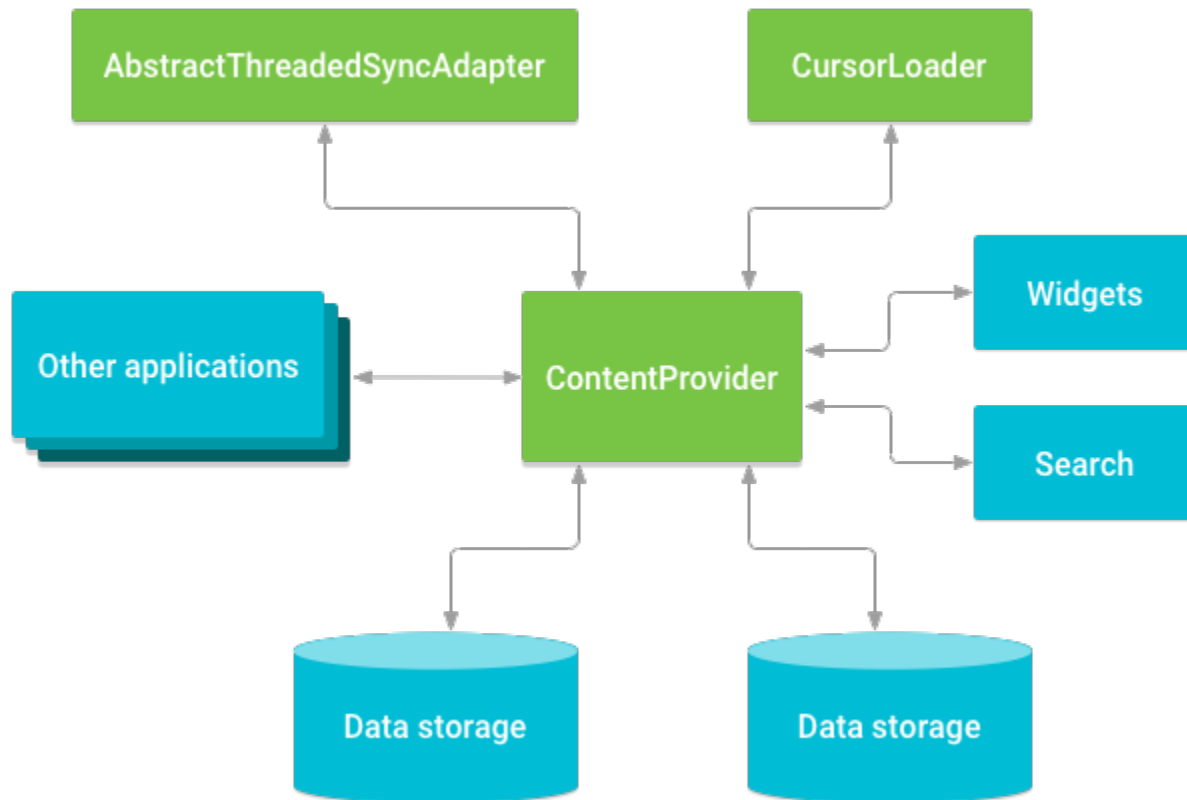


Figura 1: Relaciones de un content provider.

III. PROCEDIMIENTO

En esta ocasión se desarrollará una aplicación, donde se creará un content provider personalizado (con datos de prueba para efectos de la práctica) y se consultarán usando las apis de Android: `Cursor` y `LoaderManager`.

- I. Ejecuta Android Studio y crea un nuevo proyecto. En **Project Template** selecciona **Empty Activity** y presiona **Next**.
- II. Como **Name** coloca **Guia12AppContentProvider**, en **Package name** escribe **sv.edu.udb.guia12app.content.provider** en **Save location** escoge la carpeta de tu preferencia, en **Language** elige **Java** y el **Minimum SDK** selecciona **API 16: Android 4.1 (Jelly Bean)**

Nota: En esta práctica se hará uso del lenguaje de programación Java pero, se invita al estudiante a realizarla también utilizando Kotlin.

III. Presiona **Finish**.

IV. Crea una nueva clase en Android Studio y nómbrala “**StudentsContract**”. Su objetivo es contener el nombre de las tablas, los nombres de las columnas, las URI de contenido y los tipos MIME de cada dato.

URL Código Fuente:

<https://gist.github.com/hugovalenciadev/cea27918fb9dbbf27408001110b445c6#file-studentscontract-java>

V. Crea una nueva clase y nómbrala “**DatabaseHelper**” esta clase servirá para crear la base de datos que usará el content provider, extenderá de “**SQLiteOpenHelper**” una clase utilitaria que permite a las aplicaciones Android gestionar bases de datos SQLite de forma fácil.

URL Código Fuente:

<https://gist.github.com/hugovalenciadev/cea27918fb9dbbf27408001110b445c6#file-studentscontract-java>.

VI. Crea una nueva clase y llámala “**StudentsContentProvider**”, esta es la clase principal donde estará el content provider personalizado.

URL Código Fuente:

<https://gist.github.com/hugovalenciadev/cea27918fb9dbbf27408001110b445c6#file-studentscontract-java>.

VII. Registra el content provider en tu **AndroidManifest.xml**.

URL Código Fuente:

<https://gist.github.com/hugovalenciadev/cea27918fb9dbbf27408001110b445c6#file-studentscontract-java>.

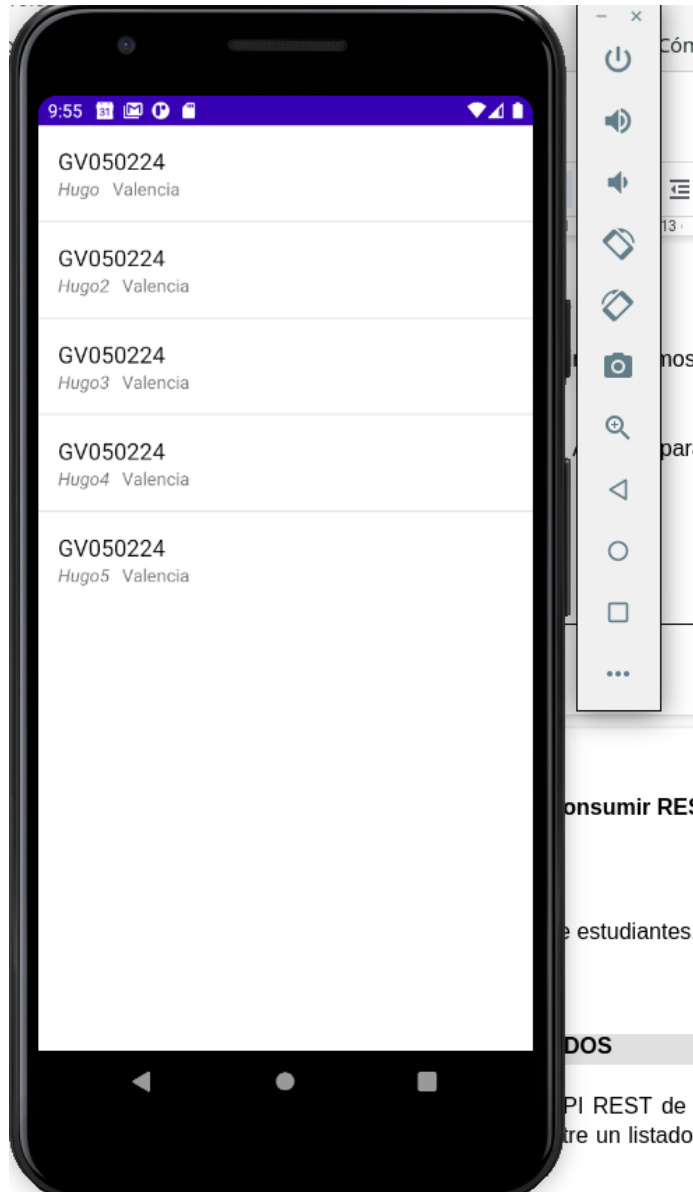
Ahora leeremos estos datos y los mostraremos en pantalla.

VIII. Crea un layout personalizado (**item.xml**) este servirá para mostrar cada fila de los datos registrados.

URL Código Fuente:

<https://gist.github.com/hugovalenciadev/cea27918fb9dbbf27408001110b445c6#file-studentscontract-java>..

- IX. Crea una clase “**StudentsAdapter**”, ésta será un Adapter para mostrar los datos guardados en una ListView.
URL Código Fuente:
<https://gist.github.com/hugovalenciadev/cea27918fb9dbbf27408001110b445c6#file-studentscontract-java..>
- X. Modifica tu clase **MainActivity**.
URL Código Fuente:
<https://gist.github.com/hugovalenciadev/cea27918fb9dbbf27408001110b445c6#file-studentscontract-java..>
- XI. Modifica el layout de **MainActivity**.
URL Código Fuente:
<https://gist.github.com/hugovalenciadev/cea27918fb9dbbf27408001110b445c6#file-studentscontract-java..>
- XII. Ejecuta la aplicación, en pantalla verás un listado de estudiantes.



V. DISCUSION DE RESULTADOS

1. Realizar CRUD en Sqlite, para la tabla **persona** con los campos (nombre, apellido, edad, teléfono) el diseño puede ser similar o queda a su creatividad. (Parte I - Almacenamiento de datos SQLite)
2. Complete la aplicación Android de la guía (Parte II - Content Providers), agregando una Activity que muestre un formulario (carnet, nombre y apellido) que permita agregar nuevos registros a la base de datos del content provider.

VII. BIBLIOGRAFIA

1. Cómo guardar datos con SQLite

<https://developer.android.com/training/data-storage/sqlite?hl=es-419>

2. Documentación Oficial Android

<https://developer.android.com/guide/topics/providers/content-provider-basics?hl=es-419>