

Proveedores de contenidos

Los proveedores de contenidos (***content providers***) son uno de los bloques constructivos más importantes de Android. Nos van a permitir acceder a información proporcionada por otras aplicaciones, o a la inversa, compartir nuestra información con otras aplicaciones.

Tras describir los principios en los que se basan los *ContentProviders*, pasaremos a demostrar cómo acceder a *ContentProviders* creados por otras aplicaciones. Esta operación resulta muy útil dado que permitirá tener acceso a información interesante, como la **lista de contactos, el registro de llamadas o los ficheros multimedia almacenados en el dispositivo**. Terminaremos mostrando cómo crear tu propio *ContentProvider*, de forma que otras aplicaciones puedan **acceder a tu información**.

El modelo de datos

La forma en la que un *ContentProvider* (Proveedor de contenidos) almacena la información es un aspecto de diseño interno, de forma que podríamos utilizar cualquiera de los métodos descritos hasta ahora para el almacenamiento de datos. No obstante, cuando hagamos una consulta al *ContentProvider* se **devolverá un objeto de tipo `Cursor`**. Esto hace que la forma más práctica para almacenar la información sea en una base de datos.

Utilizando términos del **modelo de bases de datos**, un *ContentProvider* proporciona sus datos a través de **una tabla simple, donde cada fila es un registro y cada columna es un tipo de datos** con un significado particular. Por ejemplo, el *ContentProvider* que utilizaremos en el siguiente ejemplo se llama `CallLog`, y permite acceder al registro de llamadas del teléfono. La información que nos proporciona tiene la estructura que se muestra a continuación:

<code>_id</code>	<code>date</code>	<code>number</code>	<code>Duration</code>	<code>Type</code>
1	12/10/10 16:10	555123123	65	INCOMING_TYPE
3	12/11/10 20:42	555453455	356	OUTGOING_TYPE
4	13/11/10 12:15	555123123	90	MISSED_TYPE
5	14/11/10 22:10	555783678	542	OUTGOING_TYPE

Tabla: Ejemplo de estructura de datos de un Content Provider

Cada registro incluye el campo numérico `_id` que lo identifica de forma única. Como veremos a continuación podremos utilizar este campo para identificar una llamada en concreto.

La forma de acceder a la información de un *ContentProvider* es muy similar al proceso descrito para las bases de datos. Es decir, vamos a poder realizar una consulta (incluso utilizando el lenguaje SQL), tras la cual se nos proporcionará un objeto de tipo *Cursor*. Este objeto, contiene una información con una estructura similar a la mostrada en la tabla anterior.

Las URI

Para acceder a un *ContentProvider* en particular será necesario identificarlo con una URI. Una URI (ver estándar RFC 2396) es una cadena de texto que permite identificar un recurso de información. Suele utilizarse frecuentemente en Internet, por ejemplo para acceder a una página web. Una URI está formada por cuatro partes, tal y como se muestra a continuación:

```
<standard_prefix>://<authority>/<data_path>/<id>
```

La parte **<standard_prefix>** de todos los *ContentProvider* ha de ser siempre **content**. En un primer ejemplo, utilizaremos un *ContentProvider* donde Android almacena el registro de llamadas. Para acceder a este *ContentProvider* utilizaremos la siguiente URI:

```
content://call_log/calls
```

En una tabla (**Tabla:** *ContentProviders* disponibles en Android) más adelante, se pueden ver otros ejemplos de URI que permitirán acceder a otras informaciones almacenadas en Android.

Para acceder a un elemento concreto hay que indicar un **<id>** en la URI. Por ejemplo, si lo que interesa es acceder solo a la llamada con **identificador** 4 (normalmente corresponderá a la cuarta llamada de la lista), habrá que indicar:

```
content://call_log/calls/4
```

Un mismo *ContentProvider* puede contener **múltiples conjuntos de datos** identificados por diferentes URI. Por ejemplo para acceder a los ficheros multimedia almacenados en el móvil utilizaremos el *ContentProvider* *MediaStore*, utilizando algunos de los siguientes ejemplos de URI:

```
content://media/internal/images
content://media/external/video/5
content://media/*/audio
```

Cada *ContentProvider* suele definir **constantes** de *String* con sus correspondientes URI. Por ejemplo, `android.provider.CallLog.Calls.CONTENT_URI` se corresponde con `content://call_log/calls`. Y la constante: `android.provider.MediaStore.Audio.Media.INTERNAL_CONTENT_URI` se corresponde con `content://media/internal/audio`.

Acceder a la información de un Content Provider

Android utiliza los *ContentProvider* para almacenar diferentes tipos de información. En la tabla siguiente podemos encontrar los *ContentProvider* más importantes disponibles en Android:

Clase	Información almacenada	Ejemplos de URIs
Browser	Enlaces favoritos, historial de navegación, historial de búsquedas	<code>content://browser/bookmarks</code>
CallLog	Llamadas entrantes, salientes y pérdidas.	<code>content://call_log/calls</code>
Contacts	Lista de contactos del usuario.	<code>content://contacts/people</code>
MediaStore	Ficheros de audio, vídeo e imágenes, almacenados en dispositivos de almacenamiento internos y externos.	<code>content://media/internal/images</code> <code>content://media/external/video</code> <code>content://media/*/audio</code>
Setting	Preferencias del sistema.	<code>content://settings/system/ringtone</code> <code>content://settings/system/notification_sound</code>
UserDictionary (a partir de 1.5)	Palabras definidas por el usuario, utilizadas en los métodos de entrada predictivos.	<code>content://user_dictionary/words</code>
Telephony (a partir de 1.5)	Mensajes SMS y MMS mandados o recibidos desde el teléfono.	<code>content://sms</code> <code>content://sms/inbox</code> <code>content://sms/sent</code> <code>content://mms</code>

Calendar (a partir de 4.0)	Permite consultar y editar los eventos del calendario	content://com.android.calendar/time content://com.android.calendar/events
Document (a partir de 4.4)	Permite acceder a ficheros locales o en la nube.	Content://com.dropbox.android.Dropbox/metadata/ Content://com.dominio.mio/dir/fich.txt

Tabla: ContentProviders disponibles en Android

Leer información de un Content Provider

Veamos un ejemplo que permite leer el registro de llamadas del teléfono. Creamos una nueva aplicación con los siguientes datos:

Activity main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Registro de llamadas"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"/>

    <TextView
        android:id="@+id/salida"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

De esta forma podremos identificar el `TextView` (`android:id="@+id/salida"`) desde el programa y utilizarlo para mostrar la salida.

Añade al final del fichero AndroidManifest.xml las líneas:

```
<uses-permission
    android:name="android.permission.READ_CALL_LOG">
</uses-permission>
```

NOTA IMPORTANTE!: A partir de la API 23 (Android 6.0) hay que habilitar el permiso dentro de la aplicación.

Al solicitar el permiso `READ_CALL_LOG` podremos acceder al registro de llamadas. Añade al final del método `onCreate()` de la actividad principal el siguiente código:

```
. . .
String[] TIPO_LLAMADA = {"", "entrante", "saliente", "perdida"};
TextView salida = (TextView) findViewById(R.id.salida);
Uri llamadas = Uri.parse("content://call_log/calls");

Cursor c = getContentResolver().query(llamadas, null, null, null, null);
```

```

while (c.moveToNext()) {
    salida.append("\n"
        + DateFormat.format("dd/MM/yy k:mm ",
            c.getLong(c.getColumnIndex(CallLog.Calls.DATE)))
        + c.getString(c.getColumnIndex(CallLog.Calls.DURATION)) + ") "
        + c.getString(c.getColumnIndex(CallLog.Calls.NUMBER)) + ", "
        + TIPO_LLAMADA[Integer.parseInt(
            c.getString(c.getColumnIndex(CallLog.Calls.TYPE)))]);
}

```

En primer lugar se define un *array de strings*, de forma que `TIPO_LLAMADA[1]` corresponda a tipo de llamada “*entrante*”, `TIPO_LLAMADA[2]` corresponda a tipo de llamada “*saliente*”, etc. A continuación se crea el objeto *salida* que es asignado al `TextView` correspondiente del *layout*.

Ahora comienza lo interesante, creamos la URI, llamadas asociada a `content://call_log/calls`. Para realizar la consulta creamos el `Cursor`, `c`. Se trata de la misma clase que hemos utilizado para hacer una consulta en una base de datos. A través del método `getContentResolver()`, se obtiene un `ContentResolver` asociado a la actividad, con el que podemos llamar al método `query()`. Este método permite varios parámetros para indicar exactamente los elementos que nos interesan, de forma similar a como se hace en una base de datos. Estos parámetros los veremos más adelante. Al no indicar ninguno se devolverán todas las llamadas registradas.

No tenemos más que desplazarnos por todos los elementos del cursor (`c.moveToNext()`) e ir añadiendo a la salida (`salida.append()`) la información correspondiente a cada registro. En concreto fecha, duración, número de teléfono y tipo de llamada. Una vez que el cursor se encuentra situado en una fila determinada, podemos obtener la información de una columna utilizando los métodos `getString()`, `getInt()`, `getLong()` y `getFloat()` dependiendo del tipo de dato almacenado. Estos métodos necesitan como parámetros el índice de columna. Para averiguar el índice de cada columna utilizaremos el método `getColumnIndex()`, indicando el nombre de la columna. En nuestro caso estos nombres son “*date*”, “*duration*”, “*number*” y “*type*”. En el ejemplo, en lugar de utilizar estos nombres se han utilizado cuatro constantes definidas con estos valores en la clase `Calls`.

Mención especial a la columna “*date*” que nos devuelve un entero largo que representa un instante concreto de una fecha. Para mostrarla en el formato deseado utilizamos el método estático `format()` de la clase `DateFormat`.

El resultado de ejecutar este programa se muestra a continuación:



The screenshot shows a list of call records with the following data:

Fecha y hora	Duración	Número de teléfono	Tipo de llamada
19/10/16 20:13	(7)	616407507	saliente
19/10/16 20:45	(11)	616407507	saliente
10/01/17 22:00	(14)	945233714	saliente
10/11/16 3:20	(9)	962114580	saliente
12/12/16 8:08	(184)	912245863	saliente
10/01/17 22:50	(48)	987458251	entrante
10/01/17 23:32	(0)	987458251	perdida
10/01/17 23:32	(5)	667452598	entrante
10/01/17 23:32	(48)	695878547	perdida
18/11/16 15:53	(36)	945003678	entrante

Veamos con más detalle el método `query()`:

```
Cursor query(Uri uri, String[] proyeccion,  
             String seleccion, String[] argsSelecc, String orden)
```

Donde los parámetros corresponden a:

<code>uri</code>	URI correspondiente al <i>ContentProvider</i> a consultar.
<code>proyeccion</code>	Lista de columnas que queremos que nos devuelva.
<code>seleccion</code>	Clausula SQL correspondiente a WHERE.
<code>argsSelecc</code>	Lista de argumentos utilizados en el parámetro <code>seleccion</code> .
<code>orden</code>	Clausula SQL correspondiente a ORDER BY.

Para comprobar el uso de estos parámetros reemplaza en el ejemplo anterior:

```
Cursor c = getContentResolver().query(llamadas, null, null, null, null);
```

por,

```
String[] proyeccion = new String[] {CallLog.Calls.DATE,  
    CallLog.Calls.DURATION, CallLog.Calls.NUMBER, CallLog.Calls.TYPE };
```

```
String[] argsSelecc = new String[] {"1"};
```

```
Cursor c = query(  
    llamadas,          // Uri del ContentProvider  
    proyeccion,        // Columnas que nos interesan  
    "type = ?",        // consulta WHERE  
    argsSelecc,        // parámetros de la consulta anterior  
    "date DESC");     // Ordenado por fecha, orden descendiente
```

De esta forma nos devolverán solo las columnas indicadas en `proyección`. Esto supone un ahorro de memoria en caso de que existan muchas columnas. En el siguiente parámetro se indica las filas que nos interesan por medio de una consulta de tipo `WHERE`. En caso de encontrar algún carácter `?` este es sustituido por el primer `string` del parámetro `argSelecc`. En caso de haber más caracteres `?` se irían sustituyendo siguiendo el mismo orden. Cuando se sustituyen los interrogantes cada elemento de `argSelecc` es introducido entre comillas. Por lo tanto, en el ejemplo la consulta `WHERE` resultante es `"WHERE type = '1'"`. Esta consulta implica que solo se mostrarán las llamadas entrantes. El último parámetro sería equivalente a indicar en SQL `"SORTED BY date DESC"` es decir el resultado estará ordenado por fecha en orden descendiente.

Registro de llamadas

```
16/12/19 8:59 (11) 945233714, saliente  
06/10/19 22:35 (14) 659107507, saliente  
06/10/19 21:32 (82) 659107507, saliente  
01/10/19 23:38 (13) 659107507, saliente  
20/09/19 0:34 (3) 616407507, saliente
```

Escribir información en un ContentProvider

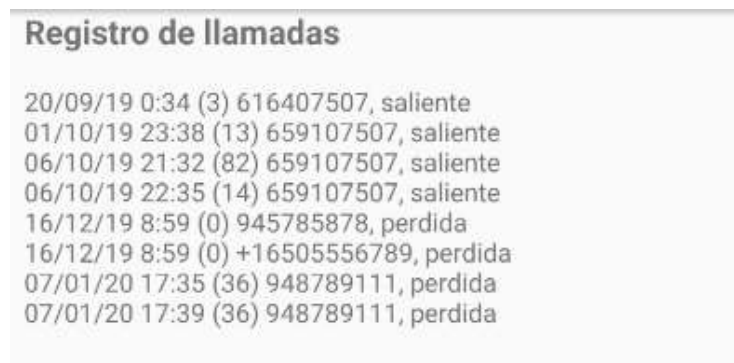
Añadir un nuevo elemento en un *ContentProvider* resulta muy sencillo. Para ver como se hace escribe el siguiente código:

```
ContentValues valores = new ContentValues();
valores.put(CallLog.Calls.DATE, new Date().getTime() );
valores.put(CallLog.Calls.NUMBER, "944011588");
valores.put(CallLog.Calls.DURATION, "67");
valores.put(CallLog.Calls.TYPE, CallLog.Calls.INCOMING_TYPE);
Uri nuevoElemento = getContentResolver().insert(CallLog.Calls.CONTENT_URI,
                                                valores);
```

Como se puede ver empezamos creando un objeto `ContentValues` donde vamos almacenando una serie de pares de valores, nombre de columna y valor asociado a la columna. A continuación, se llama a `getContentResolver().insert()` pasándole la URI del *ContentProvider* y los valores a insertar. Este método nos devuelve una URI que apunta de forma específica al elemento que acabamos de insertar. Se podría utilizar esta URI para hacer una consulta y obtener un cursor al nuevo elemento y así poder modificarlo, borrarlo u obtener el `_ID`. Recuerda que hay que pedir el permiso `WRITE_CALL_LOG`.

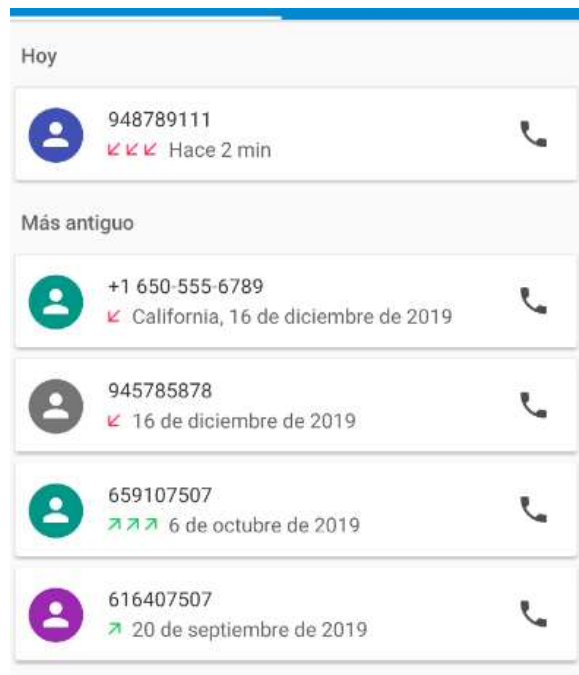
```
<uses-permission
    android:name="android.permission.WRITE_CALL_LOG" >
</uses-permission>
```

Si ejecutas ahora el programa, la nueva llamada insertada ha de aparecer en primer lugar.



Registro de llamadas	
20/09/19 0:34 (3)	616407507, saliente
01/10/19 23:38 (13)	659107507, saliente
06/10/19 21:32 (82)	659107507, saliente
06/10/19 22:35 (14)	659107507, saliente
16/12/19 8:59 (0)	945785878, perdida
16/12/19 8:59 (0)	+16505556789, perdida
07/01/20 17:35 (36)	948789111, perdida
07/01/20 17:39 (36)	948789111, perdida

Estamos modificando el registro de llamadas del sistema, por lo tanto, también puedes verificar esta información desde las aplicaciones del sistema.



Borrar y modificar elementos de un ContentProvider

Para eliminar elementos de un *ContentProvider*, se puede utilizar el método `delete()`:

```
int ContentProvider.delete(Uri uri, String seleccion, String[] argsSelecc)
```

Este método devuelve el número de elementos eliminados. Los tres parámetros del método son:

<code>uri</code>	URI correspondiente al <i>ContentProvider</i> a consultar.
<code>seleccion</code>	Clausula SQL correspondiente a <code>WHERE</code> .
<code>argsSelecc</code>	Lista de argumentos utilizados en el parámetro <code>seleccion</code> .

Si quisiéramos eliminar un solo elemento podríamos obtener su URI e indicarlo en el primer parámetro, dejando los otros dos a `null`. Si por el contrario se quieren eliminar varios elementos se puede utilizar el parámetro `seleccion`. Por ejemplo, para eliminar todos los registros de llamada del número "55555555", escribiríamos:

```
getContentResolver().delete(CallLog.Calls.CONTENT_URI, "number='55555555'", null);
```

También puedes utilizar el método `update()` para modificar elementos de un *ContentProvider*.

```
int ContentProvider.update(Uri uri, ContentValues valores, String seleccion, String[] argsSelecc)
```

Por ejemplo, si quisiéramos modificar los registros con número 55555555, por el número 44444444, escribiríamos:

```
ContentValues valores2 = new ContentValues();
valores2.put(CallLog.Calls.NUMBER, "44444444");
getContentResolver().update(CallLog.Calls.CONTENT_URI, valores2,
    "number='55555555'", null);
```

Ejemplo:

Vamos a ver un ejemplo de ContentProvider, en el que disponemos de 3 botones, uno para consultar el registro de llamadas, otro para insertar una nueva entrada en el registro de llamadas y un último botón para eliminar una entrada.

La interfaz gráfica sería:



Activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true">
        <Button
            android:id="@+id/btConsultar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Consultar"
            android:layout_weight="1"/>

        <Button
            android:id="@+id/btInsertar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Insertar"
            android:layout_weight="1"/>

        <Button
            android:id="@+id/btBorrar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Borrar"
            android:layout_weight="1"/>
    </LinearLayout>

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Registro de llamadas"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
```



```

        android:textStyle="bold"/>

<TextView
    android:id="@+id/salida"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

</LinearLayout>

```

MainActivity.java

```

package com.example.ej_contentprovider_01;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import android.Manifest;
import android.app.Activity;
import android.content.ContentValues;
import android.content.DialogInterface;
import android.content.pm.PackageManager;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.CallLog;
import android.text.format.DateFormat;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import java.util.Date;

public class MainActivity extends AppCompatActivity
    implements View.OnClickListener {

    private static final int SOLICITUD_PERMISO_READ_CALL_LOG = 0;
    private static final int SOLICITUD_PERMISO_WRITE_CALL_LOG = 1;

    private Button btConsultar;
    private Button btInsertar;
    private Button btBorrar;
    private TextView salida;

    Cursor c;
    Uri llamadas;
    String[] TIPO_LLAMADA = {"", "entrante", "saliente", "perdida"};

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

        salida = (TextView) findViewById(R.id.salida);
        llamadas = Uri.parse("content://call_log/calls");

        btConsultar = (Button) findViewById(R.id.btConsultar);

```

```

        btInsertar = (Button) findViewById(R.id.btInsertar);
        btBorrar = (Button) findViewById(R.id.btBorrar);

        btConsultar.setOnClickListener(this);
        btInsertar.setOnClickListener(this);
        btBorrar.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {

        switch (v.getId()) {
            case R.id.btConsultar:
                //Comprobamos cual es la versión de android
                if (android.os.Build.VERSION.SDK_INT >=
                    android.os.Build.VERSION_CODES.M) {
                    // versiones con android 6.0 o superior. Comprobar permisos
                    if (ContextCompat.checkSelfPermission(this,
                        Manifest.permission.READ_CALL_LOG)
                        == PackageManager.PERMISSION_GRANTED) {

                        //Realizamos la consulta
                        recuperarVisualizarLlamadas();
                    }
                    // Sino tenemos permisos los solicitamos
                    else {
                        //Solicitamos permisos
                        solicitarPermiso(Manifest.permission.READ_CALL_LOG,
                            "Sin el permiso para administrar llamadas " +
                            "no se puede acceder al registro de llamadas.",
                            SOLICITUD_PERMISO_READ_CALL_LOG, this);
                    }
                } else {
                    // para versiones anteriores a android 6.0
                    recuperarVisualizarLlamadas();
                }
                break;

            case R.id.btInsertar:

                if (android.os.Build.VERSION.SDK_INT >=
                    android.os.Build.VERSION_CODES.M) {
                    // versiones con android 6.0 o superior. Comprobar permisos
                    if (ContextCompat.checkSelfPermission(this,
                        Manifest.permission.WRITE_CALL_LOG) ==
                        PackageManager.PERMISSION_GRANTED) {

                        //Insertamos nueva llamada
                        insertaLlamada();
                    }
                    // Sino tenemos permisos los solicitamos
                    else {
                        //Solicitamos permisos
                        solicitarPermiso(Manifest.permission.WRITE_CALL_LOG,
                            "Sin el permiso para administrar llamadas " +
                            "no se puede acceder al registro de llamadas.",
                            SOLICITUD_PERMISO_WRITE_CALL_LOG, this);
                    }
                } else {
                    // para versiones anteriores a android 6.0
                    insertaLlamada();
                }
            }
        }
    }
}

```

```

        break;

    case R.id.btBorrar:

        if (android.os.Build.VERSION.SDK_INT >=
            android.os.Build.VERSION_CODES.M) {
            // versiones con android 6.0 o superior. Comprobar permisos
            if (ContextCompat.checkSelfPermission(this,
                Manifest.permission.WRITE_CALL_LOG) ==
                PackageManager.PERMISSION_GRANTED) {

                //Eliminamos llamada(s)
                getResolver().delete(CallLog.Calls.CONTENT_URI,
                    "number='945233714'", null);
            }
            // Sino tenemos permisos los solicitamos
            else {
                //Solicitamos permisos
                solicitarPermiso(Manifest.permission.WRITE_CALL_LOG,
                    "Sin el permiso para administrar llamadas " +
                    "no se puede acceder al registro de llamadas",
                    SOLICITUD_PERMISO_WRITE_CALL_LOG, this);
            }
        } else {
            // para versiones anteriores a android 6.0
            getResolver().delete(CallLog.Calls.CONTENT_URI,
                "number='945233714'", null);
        }
        break;

    default:
}

}

public static void solicitarPermiso(final String permiso,
    String justificacion,
    final int requestCode,
    final Activity actividad) {

    if (ActivityCompat.shouldShowRequestPermissionRationale(actividad,
        permiso)) {

        //Informamos al usuario para qué y por qué se necesitan los permisos
        new AlertDialog.Builder(actividad)
            .setTitle("Solicitud de permiso")
            .setMessage(justificacion)
            .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog,
                    int whichButton) {
                    ActivityCompat.requestPermissions(actividad,
                        new String[]{permiso}, requestCode);
                }
            }).show();
    } else {
        //Muestra el cuadro de dialogo para la solicitud de los permisos
        // y registra el permiso según respuesta del usuario
        ActivityCompat.requestPermissions(actividad,
            new String[]{permiso}, requestCode);
    }
}

@Override
public void onRequestPermissionsResult(int requestCode,

```

```

        @NonNull String[] permissions,
        @NonNull int[] grantResults) {
    if (requestCode == SOLICITUD_PERMISO_READ_CALL_LOG) {
        if (grantResults.length == 1 &&
            grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            //borrarLlamada();
            recuperarVisualizarLlamadas();
        } else {
            salida.append("\n NO ES POSIBLE REALIZAR LA CONSULTA");
        }
    }
    if (requestCode == SOLICITUD_PERMISO_WRITE_CALL_LOG) {
        if (grantResults.length == 1 &&
            grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            insertaLlamada();
        } else {
            salida.append("\n NO ES POSIBLE INSERTAR UNA NUEVA LLAMADA");
        }
    }
}

private void insertaLlamada() {
    ContentValues valores = new ContentValues();
    valores.put(CallLog.Calls.DATE, new Date().getTime());
    valores.put(CallLog.Calls.NUMBER, "948789111");
    valores.put(CallLog.Calls.DURATION, "36");
    valores.put(CallLog.Calls.TYPE, CallLog.Calls.MISSED_TYPE);
    Uri nuevoElemento = getContentResolver().insert(CallLog.Calls.CONTENT_URI,
                                                    valores);
}

//Recuperamos las llamadas y las visualizamos en un TextView llamado salida
private void recuperarVisualizarLlamadas() {

    salida.setText("");
    Cursor c = getContentResolver().query(llamadas, null, null, null, null);
    int cantRegistros = c.getCount();
    if (cantRegistros == 0) {
        new AlertDialog.Builder(this)
            .setTitle("Info")
            .setMessage("No hay llamadas para visualizar")
            .setPositiveButton("Ok",
                new DialogInterface.OnClickListener() {

                    public void onClick(DialogInterface dialog,
                                        int whichButton) {

                    }

                })
            .show();
    } else {
        while (c.moveToNext()) {
            salida.append("\n"
                + DateFormat.format("dd/MM/yy k:mm ",
                    c.getLong(c.getColumnIndex(CallLog.Calls.DATE)))
                + c.getString(c.getColumnIndex(CallLog.Calls.DURATION)) + " "
                + c.getString(c.getColumnIndex(CallLog.Calls.NUMBER)) + ", "
                + TIPO_LLAMADA[Integer.parseInt(
                    c.getString(c.getColumnIndex(CallLog.Calls.TYPE)))]);
        }
    }
}
}

```

Por último damos los permisos necesarios en el **AndroidManifest.xml**, quedando el fichero con el siguiente código.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ej_contentprovider_01">

    <uses-permission
        android:name="android.permission.READ_CALL_LOG">
    </uses-permission>

    <uses-permission
        android:name="android.permission.WRITE_CALL_LOG">
    </uses-permission>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Creación de Content Provider

Para crear tu propio *ContentProvider*, vamos a seguir tres pasos:

1. Definir una estructura de almacenamiento de datos. En este ejemplo usaremos una base de datos SQLite.
2. Crear nuestra clase extendiendo *ContentProvider*.
3. Declarar el *ContentProvider* en el *AndroidManifest.xml* de nuestra aplicación.

Si no deseas compartir tu información con otras aplicaciones, no es necesario crear un *ContentProvider*. En ese caso resulta mucho más sencillo usar una base de datos directamente, tal y como se ha estudiado previamente.

Definir la estructura de almacenamiento del ContentProvider

El objetivo último de un *ContentProvider* es almacenar información de forma permanente. Por lo tanto, resulta imprescindible utilizar alguno de los mecanismos descritos anteriormente para almacenar datos. Podemos realizar consultas a un *ContentProvider* de forma similar a como se hace a una base de datos (se pueden hacer consultas SQL y nos devuelve un objeto de tipo *Cursor*). Por lo tanto, la forma más sencilla de almacenar los datos de un *ContentProvider* es en una base de datos. De esta forma, si nos solicitan una consulta SQL, no tendremos más que trasladarla a nuestra base de datos, y el objeto *Cursor* que nos devuelva será el resultado que nosotros devolveremos.

Para crear la base de datos de nuestro *ContentProvider*, crea una clase que se llame *CientesSqliteHelper* y añádele el siguiente código;

```
public class CienteSqliteHelper extends SQLiteOpenHelper {
    //Sentencia SQL para crear la tabla de Clientes
    String sqlCreate = "CREATE TABLE clientes" +
        "(_id INTEGER PRIMARY KEY AUTOINCREMENT," +
        " nombre TEXT," +
        " telefono TEXT," +
        " email TEXT)";

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

    @Override
    public void onCreate(SQLiteDatabase db) {
        String nombre;
        String telefono;
        String email;

        //Se ejecuta la sentencia SQL de creación de la tabla
        db.execSQL(sqlCreate);
    }
}
```

```

        //Insertamos 15 clientes de ejemplo
        for (int i=1; i<=15; i++){
            if (i<10) {
                //Generamos los dtos de muestra
                nombre = "Cliente0"+i;
                telefono = "900-123-00"+i;
                email = "email0"+i+"@mail.com";
            }
            else {
                nombre = "Cliente"+i;
                telefono = "900-123-0"+i;
                email = "email"+i+"@mail.com";
            }
            //Insertamos los datos en la tabla Clientes
            db.execSQL("INSERT INTO clientes (nombre, telefono, email) " +
                "VALUES ('"+nombre+"','"+telefono+"','"+email+"')");
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int i, int i1) {

        //Se elimina la versión anterior de la tabla
        db.execSQL("DROP TABLE IF EXISTS clientes");

        //Se crea la nueva versión de la tabla
        db.execSQL(sqlCreate);
    }
}

```

Esta clase que acabamos de añadir al proyecto se encarga de crear la base de DBClientes extendiendo la clase SQLiteOpenHelper. Este proceso es similar al ya descrito al estudiar las bases de datos. En el método onCreate, se crea y se introducen algunos datos en la tabla Clientes de la base de datos DBClientes.

Extendiendo la clase ContentProvider

Ahora vayamos con la tarea más laboriosa, la creación de una clase descendiente de ContentProvider.

Los métodos principales que tenemos que implementar son:

- getType () → Devuelve el tipo MIME de los elementos del ContentProvider.
- query () → Permite realizar consultas al ContentProvider.
- insert() → Inserta nuevos datos
- delete () → borra elementos del ContentProvider
- update () → permite modificar los datos existentes.

La clase CotentProvider es thread-safe, es decir, toma las precauciones necesarias para evitar problemas con las llamadas simultaneas de varios procesos. Por lo tanto, en la creación de una subclase no nos tenemos que preocupar de este aspecto.

Añadimos una nueva clase que se llame ClientesProvider e introduciremos el siguiente código:

```
public class ClientesProvider extends ContentProvider {

    public static final String AUTORIDAD = "com.example.ej_contentprovider_02";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTORIDAD +
"/clientes");

    public static final int TODOS_LOS_ELEMENTOS = 1;
    public static final int UN_ELEMENTO = 2;
    private static UriMatcher URI_MATCHER = null;
    static {
        URI_MATCHER = new UriMatcher(UriMatcher.NO_MATCH);
        URI_MATCHER.addURI(AUTORIDAD, "clientes", TODOS_LOS_ELEMENTOS);
        URI_MATCHER.addURI(AUTORIDAD, "clientes/#", UN_ELEMENTO);
    }

    public static final String TABLA_CLIENTES = "clientes";
    private SQLiteDatabase baseDeDatos;

    @Override
    public boolean onCreate() {
        ClienteSqliteHelper dbHelper = new ClienteSqliteHelper(getContext(),
"DBClientes", null, 1);
        baseDeDatos = dbHelper.getWritableDatabase();
        return baseDeDatos != null && baseDeDatos.isOpen();
    }
}
```

Como no podía ser de otra forma, la clase extiende de ContentProvider. A continuación creamos constantes AUTORIDAD y CONTENT_URI, que identificarán nuestro ContentProvider mediante la URI:

content://com.example.ej_contentprovider_02/clientes

Las siguientes líneas permiten crear el objeto estatico URI-MATCHER de la clase UriMatcher. Es habitual en Java utilizar variables estáticas finales para albergar objetos que no van a cambiar en toda la vida de la aplicación, es decir, que son constantes. Conviene recordar que solo se creará un objeto URI_MATCHER aunque se instancie varias veces la clase ClientesProvider. La clase UriMatcher permite diferenciar entre diferentes tipos de URI que vamos a manipular. En nuestro caso permitimos dos tipos de URI: acabada en /clientes que identifica a todos los clientes almacenados, y acabada en /clientes/#, donde hay que sustituir la # por un código numérico que coincida con el campo `_id` de nuestra tabla. Cada tipo de URI ha de tener un código numérico asociado, en el ejemplo NO_MATCH (-1), TODO_LOS_ELEMENTOS (1) y UN_ELEMENTO (2).

La declaración de variables termina con la constante TABLA_CLIENTES, que identifica la tabla que usaremos para almacenar la información y el objeto baseDeDatos donde se almacenará la información.

A continuación está el método `onCreate ()`, que se llama cuando se crea una instancia de esta clase. Basicamente se crea un `SQLiteHelper` a partir de la clase descrita más arriba `ClienteSQLiteHelper` y se asigna la base de datos resultante a la variable `baseDeDatos`. Devolvemos `true` solo en el caso de que no haya habido problemas en su creación.

Veamos ahora la implementación del método `getType()`, que a partir de una URI nos devuelve el tipo MIME que le corresponde:

```
@Override
public String getType(final Uri uri) {
    switch (URI_MATCHER.match(uri)) {
        case TODOS_LOS_ELEMENTOS:
            return "vnd.android.cursor.dir/vnd.com.example.clientes";
        case UN_ELEMENTO:
            return "vnd.android.cursor.item/vnd.com.example.clientes";
        default:
            throw new IllegalArgumentException("URI incorrecta: " + uri);
    }
}
```

NOTA: Los tipos MIME (Multipurpose Internet Mail Extensions) fueron creados para identificar un tipo de datos concreto que añadíamos como anexo a un correo electrónico, aunque en la actualidad son utilizados por muchos sistemas y protocolos. Un tipo MIME tiene dos partes: `tipo_generico/tipo_especifico`; por ejemplo, `image/gif`, `image/jpeg`, `text/html`, etc.

Existe un convenio para identificar los tipos MIME que proporciona un `ContentProvider`. Si se trata de un recurso único, utilizamos:

```
vnd.android.cursor.item/vnd.ELEMENTO
```

Y si se trata de una colección de recursos, utilizaremos:

```
vnd.android.cursor.dir/vnd.ELEMENTO
```

Donde `ELEMENTO` ha de ser reemplazado por un identificador que describa el tipo de datos. En nuestro caso hemos elegido `com.example.clientes`. Utilizar prefijos para definir el elemento minimiza el riesgo de confusión con otro ya existente.

A continuación debemos de sobrescribir los métodos `query`, `insert`, `delete` y `update`, que permitan consultar, insertar, borrar y actualizar elementos de nuestro `ContentProvider`. Vayamos con el primer método:

```
@Override
public Cursor query(@NonNull Uri uri,
                   @Nullable String[] proyeccion,
                   @Nullable String seleccion,
                   @Nullable String[] argSeleccion,
                   @Nullable String orden) {

    //SQLiteQueryBuilder es una clase asistente en android.database.sqlite que
    //permite construir consultas SQL que se ejecutarán por SQLiteDatabase
    // en una instancia de una base de datos SQLite.
```

```

SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
queryBuilder.setTables(TABLA_CLIENTES);
switch (URI_MATCHER.match(uri)) {
    case TODOS_LOS_ELEMENTOS:
        break;
    case UN_ELEMENTO:
        String id = uri.getPathSegments().get(1);
        queryBuilder.appendWhere("_id = " + id);
        break;
    default:
        throw new IllegalArgumentException("URI incorrecta " + uri);
}
return queryBuilder.query(baseDeDatos, proyeccion, seleccion, argSeleccion,
    null, null, orden);
}

```

El método `query()` que hemos de implementar tiene parámetros similares a `SQLiteQueryBuilder.query()`. Por lo tanto, no tenemos más que trasladar la consulta a nuestra base de datos. No obstante, existe un pequeño inconveniente si nos pasan una URI que identifique un solo elemento, como la que se muestra a continuación:

content://com.example.ej_contentprovider_02/clientes/23

Tenemos que asegurarnos que solo se devuelve el elemento con `_id = 23`. Para conseguirlo se introduce una clausula `switch`, que en caso de tratarse de una URI de tipo `UN_ELEMENTO`, añade a la clausula `WHERE` de la consulta la condición correspondiente mediante el método `appendWhere()`. La clausula `WHERE` puede tener más condiciones si se ha utilizado el parámetro `seleccion`.

```

@Override
public Uri insert(@NonNull Uri uri, @Nullable ContentValues valores) {
    long IdFila = baseDeDatos.insert(TABLA_CLIENTES, null, valores);
    if (IdFila > 0) {
        return ContentUris.withAppendedId(CONTENT_URI, IdFila);
    } else {
        throw new SQLException("Error a insertar registro en " + uri);
    }
}

@Override
public int delete(@NonNull Uri uri, @Nullable String seleccion,
    @Nullable String[] argSeleccion) {
    switch (URI_MATCHER.match(uri)) {
        case TODOS_LOS_ELEMENTOS:
            break;
        case UN_ELEMENTO:
            String id = uri.getPathSegments().get(1);
            if (TextUtils.isEmpty(seleccion)) {
                seleccion = "_id = " + id;
            } else {
                seleccion = "_id = " + id + " AND (" + seleccion + ")";
            }
            break;
        default:
    }
}

```

```

        throw new IllegalArgumentException("URI incorrecta: " + uri);
    }
    return baseDeDatos.delete(TABLA_CLIENTES, seleccion, argSeleccion);
}

```

El método `insert()`, no requiere ninguna explicación adicional.

El método `delete()`, igual a como ocurrió en el método `query()`, presenta el inconveniente de que pueden habernos indicado una URI que identifique un solo elemento (`.../clientes/23`). En el método `query()` solucionamos este problema llamando a `SQLiteQueryBuilder.appendWhere()`. Sin embargo, ahora no disponemos de un objeto de esta clase, por lo que nos vemos obligados a realizar este trabajo a mano. En caso de no haberse indicado nada en `seleccion`, este parámetro valdrá `"_id=23"`; y en caso de haberse introducido una condición, por ejemplo `"nombre='XXXX'"`, `seleccion`, valdrá `"_id=23 AND (nombre='XXXX')"`.

```

@Override
public int update(@NonNull Uri uri,
                 @Nullable ContentValues valores,
                 @Nullable String seleccion,
                 @Nullable String[] argSeleccion) {
    switch (URI_MATCHER.match(uri)) {
        case TODOS_LOS_ELEMENTOS:
            break;
        case UN_ELEMENTO:
            String id = uri.getPathSegments().get(1);
            if (TextUtils.isEmpty(seleccion)) {
                seleccion = "_id = " + id;
            } else {
                seleccion = "_id = " + id + "AND (" + seleccion + ")";
            }
            break;
        default:
            throw new IllegalArgumentException("URI incorrecta: " + uri);
    }
    return baseDeDatos.update(TABLA_CLIENTES, valores, seleccion, argSeleccion);
}

```

Finalizamos con el método `update()`, que es muy similar a `delete()`.

Declarar el ContentProvider en AndroidManifest.xml

Si queremos que nuestro ContentProvider sea visible para otras aplicaciones, resulta imprescindible hacérselo saber al sistema declarándolo en el `AndroidManifest.xml`. Para conseguirlo solo hay que añadir el siguiente código dentro de la etiqueta `<application>`:

```

<!-- Para que el contentProvider sea visible para otras aplicaciones-->
<provider
    android:authorities="com.example.ej_contentprovider_02"
    android:name = "com.example.ej_contentprovider_02.ClientesProvider"
    android:exported="true"/>

```

La declaración de un `ContentProvider` requiere que se especifiquen los atributos:

- `name`: nombre cualificado de la clase donde hemos implementado nuestro `ContentProvider`.
- `authorities`: parte correspondiente a la autoridad de las URI que vamos a publicar. Puede indicarse más de una autoridad.

También se pueden indicar otros atributos en la etiqueta `<provider>`. Veamos los más importantes:

- `label`: etiqueta que describe el `ContentProvider` que se mostrará al usuario. Es una referencia a un recurso de tipo `string`.
- `icon`: Una referencia a un recurso de tipo `drawable` con un icono que represente nuestro `ContentProvider`.
- `enabled`: indica si está habilitado. El valor por defecto es `true`.
- `exported`: indica si se puede acceder a él desde otras aplicaciones. El valor por defecto es `false`. Si está en `false` solo podrá ser utilizado por aplicaciones con el mismo id de usuario que la aplicación donde se crea.
- `readPermission`: permiso requerido para consultar el `ContentProvider`.
- `writePermission`: permiso requerido para modificar el `ContentProvider`.
- `permission`: permiso requerido para consultar o modificar el `ContentProvider`. Si efectivamente se indica `readPermission` o `writePermission`. Para más información, consultar tema de seguridad.
- `multiprocess`: indica si cualquier proceso puede crear una instancia del `ContentProvider` (`true`) o solo el proceso de la aplicación donde se ha creado (`false`, valor por defecto).
- `process`: nombre del proceso en el que el `ContentProvider` ha de ejecutarse. Habitualmente, todos los componentes de una aplicación se ejecutan en el mismo proceso creado para la aplicación. Si no se indica lo contrario, el nombre del proceso coincide con el nombre del paquete de la aplicación (si lo deseas puedes cambiar este nombre con el atributo `process` de la etiqueta `<application>`). Si prefieres que un componente de la aplicación se ejecute en su propio proceso, has de utilizar este atributo.
- `initOrder`: orden en que el `ContentProvider` ha de ser instalado en relación con otros `ContentProvider` del mismo proceso.
- `syncable`: indica si la información del `ContentProvider` está sincronizada con un servidor.

Utilización de un Content Provider creado por nosotros

Una vez creado y declarado nuestro `ContentProvider`, vamos a probarlo desde otra aplicación, en este caso denominada `ej_contentProvider_03`.

Vamos a crear una nueva clase llamada, por ejemplo, `LeerClientesProvider` e introduce el siguiente código:

```

public class LeerClientesProvider {
    private Activity activity;

    public LeerClientesProvider (Activity activity){
        this.activity = activity;
    }
    public void guardarCliente (String nombre, String telefono, String email){
        Uri uri =
Uri.parse("content://com.example.ej_contentprovider_02/clientes");

        ContentValues valores = new ContentValues();
        valores.put ("nombre", nombre);
        valores.put ("telefono", telefono);
        valores.put ("email", email);
        try {
            activity.getContentResolver().insert(uri, valores);
        } catch (Exception e) {
            Toast.makeText(activity, "Verificar que está instalado "+
                "com.example.ej_contentprovider_02",
Toast.LENGTH_SHORT).show();
            Log.e("LeerClientesProvider", "Error: "+e.toString(), e);
        }
    }

    public List<String> listaClientes (){
        List<String> result = new ArrayList<String>();
        Uri uri=
Uri.parse("content://com.example.ej_contentprovider_02/clientes");
        Cursor cursor = activity.getContentResolver().query(uri,null,
null,null,"nombre DESC");

        if (cursor != null){
            while (cursor.moveToNext()){
                String nombre = cursor.getString(cursor.getColumnIndex("nombre"));
                String telefono = cur-
sor.getString(cursor.getColumnIndex("telefono"));
                String email = cursor.getString(cursor.getColumnIndex("email"));

                result.add (nombre + " " + telefono + " " + email );
            }
        }
        return result;
    }
}

```

Ejemplo Completo. Aplicación de creación de un ContentProvider y aplicación para la utilización del ContentProvider creado anteriormente.

Aplicación **EJ_ContentProvider_02**: Aplicación que crea el ContentProvider. Su interfaz gráfica tendrá varios botones para insertar, actualizar/modificar, eliminar y consultar.



Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/LinearLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text="@string/id" />

<EditText
    android:id="@+id/txtCodigo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="5"
    android:ems="10">

    <requestFocus />
</EditText>
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/nombre" />

    <EditText
        android:id="@+id/txtNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="5"
        android:ems="10">

        <requestFocus />
    </EditText>
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/telefono" />

    <EditText
        android:id="@+id/txtTelefono"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:ems="10"
        android:inputType="text"/>
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/mail" />

```

```

        <EditText
            android:id="@+id/txtEmail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10"
            android:inputType="text"/>
    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal">

        <Button
            android:id="@+id/btnInsertar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/insertar"
            android:layout_weight="1" />

        <Button
            android:id="@+id/btnActualizar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/actualizar"
            android:layout_weight="1" />

        <Button
            android:id="@+id/btnEliminar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textAllCaps="false"
            android:text="@string/eliminar"
            android:layout_weight="1" />
    </LinearLayout>

    <Button
        android:id="@+id/btnConsultar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/consultar"
        android:layout_gravity="center_horizontal" />

    <TextView
        android:id="@+id/txtResultado"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text=""
        android:textSize="15dp"
        android:background="#C1C1C1"
        android:layout_margin="5dp"/>

</LinearLayout>

```

ClienteSqliteHelper.java

```

package com.example.ej_contentprovider_02;

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

```



```

import android.database.sqlite.SQLiteOpenHelper;

import androidx.annotation.Nullable;

public class ClienteSqliteHelper extends SQLiteOpenHelper {
    //Sentencia SQL para crear la tabla de Clientes
    String sqlCreate ="CREATE TABLE clientes" +
        "(_id INTEGER PRIMARY KEY AUTOINCREMENT," +
        " nombre TEXT," +
        " telefono TEXT," +
        " email TEXT)";

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

    @Override
    public void onCreate(SQLiteDatabase db) {
        String nombre;
        String telefono;
        String email;

        //Se ejecuta la sentencia SQL de creación de la tabla
        db.execSQL(sqlCreate);

        //Insertamos 15 clientes de ejemplo
        for (int i=1; i<=15; i++){
            if (i<10) {
                //Generamos los dtos de muestra
                nombre = "Cliente0"+i;
                telefono = "900-123-00"+i;
                email = "email0"+i+"@mail.com";
            }
            else {
                nombre = "Cliente"+i;
                telefono = "900-123-0"+i;
                email = "email"+i+"@mail.com";
            }
            //Insertamos los datos en la tabla Clientes
            db.execSQL("INSERT INTO clientes (nombre, telefono, email) " +
                "VALUES ('"+nombre+"','"+telefono+"','"+email+"')");
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int i, int i1) {

        //Se elimina la versión anterior de la tabla
        db.execSQL("DROP TABLE IF EXISTS clientes");

        //Se crea la nueva versión de la tabla
        db.execSQL(sqlCreate);
    }
}

```

ClientesProvider.java

```

package com.example.ej_contentprovider_02;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteQueryBuilder;
import android.net.Uri;
import android.text.TextUtils;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;

public class ClientesProvider extends ContentProvider {

    public static final String AUTORIDAD = "com.example.ej_contentprovider_02";
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTORIDAD +
"/clientes");

    public static final int TODOS_LOS_ELEMENTOS = 1;
    public static final int UN_ELEMENTO = 2;
    private static UriMatcher URI_MATCHER = null;
    static {
        URI_MATCHER = new UriMatcher(UriMatcher.NO_MATCH);
        URI_MATCHER.addURI(AUTORIDAD, "clientes", TODOS_LOS_ELEMENTOS);
        URI_MATCHER.addURI(AUTORIDAD, "clientes/#", UN_ELEMENTO);
    }

    public static final String TABLA_CLIENTES = "clientes";
    private SQLiteDatabase baseDeDatos;

    @Override
    public boolean onCreate() {
        ClienteSqliteHelper dbHelper = new ClienteSqliteHelper(getContext(),
"DBClientes", null, 1);
        baseDeDatos = dbHelper.getWritableDatabase();
        return baseDeDatos != null && baseDeDatos.isOpen();
    }

    @Nullable
    @Override
    public String getType(final Uri uri) {
        switch (URI_MATCHER.match(uri)) {
            case TODOS_LOS_ELEMENTOS:
                return "vnd.android.cursor.dir/vnd.com.example.clientes";
            case UN_ELEMENTO:
                return "vnd.android.cursor.item/vnd.com.example.clientes";
            default:
                throw new IllegalArgumentException("URI incorrecta: " + uri);
        }
    }

    //Sobreescribimos los métodos query, insert, delete y update

    @Nullable
    @Override
    public Cursor query(@NonNull Uri uri,
        @Nullable String[] proyeccion,

```

```

        @Nullable String seleccion,
        @Nullable String[] argSeleccion,
        @Nullable String orden) {

    //SQLiteQueryBuilder es una clase asistente en android.database.sqlite que
    permite construir consultas SQL
    // que se ejecutarán por SQLiteDatabase en una instancia de una base de
    datos SQLite.

    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
    queryBuilder.setTables(TABLA_CLIENTES);
    switch (URI_MATCHER.match(uri)) {
        case TODOS_LOS_ELEMENTOS:
            break;
        case UN_ELEMENTO:
            String id = uri.getPathSegments().get(1);
            queryBuilder.appendWhere("_id = " + id);
            break;
        default:
            throw new IllegalArgumentException("URI incorrecta " + uri);
    }
    return queryBuilder.query(baseDeDatos, proyeccion, seleccion, argSelec-
cion,
        null, null, orden);
}

@Nullable
@Override
public Uri insert(@NonNull Uri uri, @Nullable ContentValues valores) {
    long IdFila = baseDeDatos.insert(TABLA_CLIENTES, null, valores);
    if (IdFila > 0) {
        return ContentUris.withAppendedId(CONTENT_URI, IdFila);
    } else {
        throw new SQLException("Error a insertar registro en " + uri);
    }
}

@Override
public int delete(@NonNull Uri uri, @Nullable String seleccion,
        @Nullable String[] argSeleccion) {
    switch (URI_MATCHER.match(uri)) {
        case TODOS_LOS_ELEMENTOS:
            break;
        case UN_ELEMENTO:
            String id = uri.getPathSegments().get(1);
            if (TextUtils.isEmpty(seleccion)) {
                seleccion = "_id = " + id;
            } else {
                seleccion = "_id = " + id + " AND (" + seleccion + ")";
            }
            break;
        default:
            throw new IllegalArgumentException("URI incorrecta: " + uri);
    }
    return baseDeDatos.delete(TABLA_CLIENTES, seleccion, argSeleccion);
}

@Override
public int update(@NonNull Uri uri,
        @Nullable ContentValues valores,
        @Nullable String seleccion,
        @Nullable String[] argSeleccion) {

```

```

        switch (URI_MATCHER.match(uri)) {
            case TODOS_LOS_ELEMENTOS:
                break;
            case UN_ELEMENTO:
                String id = uri.getPathSegments().get(1);
                if (TextUtils.isEmpty(seleccion)) {
                    seleccion = "_id = " + id;
                } else {
                    seleccion = "_id = " + id + "AND (" + seleccion + ")";
                }
                break;
            default:
                throw new IllegalArgumentException("URI incorrecta: " + uri);
        }
        return baseDeDatos.update(TABLA_CLIENTES, valores, seleccion, argSelec-
cion);
    }
}

```

MainActivity.java

```

package com.example.ej_contentprovider_02;

import androidx.appcompat.app.AppCompatActivity;

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

public class MainActivity extends AppCompatActivity {

    private EditText txtId;
    private EditText txtNombre;
    private EditText txtTelefono;
    private EditText txtEmail;

    private TextView txtResultado;

    private Button btnInsertar;
    private Button btnActualizar;
    private Button btnEliminar;

    private Button btnConsultar;

    private SQLiteDatabase db;
    private ClienteSqliteHelper usdbh;

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

        //Obtenemos las referencias a los controles
        txtId = (EditText) findViewById(R.id.txtCodigo);
        txtNombre = (EditText) findViewById(R.id.txtNombre);
    }
}

```

```

txtTelefono = (EditText) findViewById(R.id.txtTelefono);
txtEmail = (EditText) findViewById(R.id.txtEmail);

txtResultado = findViewById(R.id.txtResultado);

btnInsertar = (Button) findViewById(R.id.btnInsertar);
btnActualizar = (Button) findViewById(R.id.btnActualizar);
btnEliminar = (Button) findViewById(R.id.btnEliminar);

btnConsultar = (Button) findViewById(R.id.btnConsultar);

//Abrimos la base de datos 'DBClientes' en modo escritura
usdbh = new ClienteSqliteHelper(this, "DBClientes", null, 1);

btnInsertar.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {

        //Recuperamos los valores de los campos de texto
        String nomb = txtNombre.getText().toString();
        String telf = txtTelefono.getText().toString();
        String mail = txtEmail.getText().toString();

        db = usdbh.getWritableDatabase();

        if (nomb.length()==0 || telf.length()==0 || mail.length()==0){
            Toast.makeText(getApplicationContext(), "Hay que introducir to-
dos los campos para insertar un nuevo registro", Toast.LENGTH_SHORT).show();
        }
        else {

            //Creamos el registro a insertar como Objeto ContentValues
            ContentValues nuevoRegistro = new ContentValues();
            nuevoRegistro.put("nombre", nomb);
            nuevoRegistro.put("telefono", telf);
            nuevoRegistro.put("email", mail);

            //Insertamos el registro en la base de datos
            db.insert("clientes", null, nuevoRegistro);

            Toast.makeText(getApplicationContext(), "Registro inserta-
do", Toast.LENGTH_SHORT).show();

            //Vaciamos los campos
            txtId.setText("");
            txtNombre.setText("");
            txtTelefono.setText("");
            txtEmail.setText("");
        }
        db.close();
    }
});

btnConsultar.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        db = usdbh.getReadableDatabase();
        //Alternativa 1: método.rawQuery
        Cursor c =db.rawQuery("SELECT _id, nombre, telefono, email FROM
clientes",null);

        //Recorremos los resultados para mostrarlos en pantalla

```

```

txtResultado.setText("");
if (c.moveToFirst()){
    //Recorremos el cursor hasta que no haya más registros.
    do {
        int codigo = c.getInt(0);
        String nombre =c.getString(1);
        String telefono =c.getString(2);
        String email = c.getString(3);

        txtResultado.append (codigo + " - " +nombre + " - "
        "+telefono + " - " +email+"\n" );

    }while (c.moveToNext());
    }

db.close();
});

btnActualizar.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {

        db = usdbh.getWritableDatabase();
        //Recuperamos los valores de los campos de texto
        String cod = txtId.getText().toString();
        String nom = txtNombre.getText().toString();
        String telef = txtTelefono.getText().toString();
        String mail = txtEmail.getText().toString();

        //Establecemos los campos-valores a actualizar
        ContentValues valores = new ContentValues();
        if (nom.length() != 0)
            valores.put("nombre", nom);
        if (telef.length() != 0)
            valores.put("telefono", telef);
        if (mail.length() != 0)
            valores.put("email", mail);
        if (cod.length() != 0) {
            //Actualizamos el registro en la base de datos
            db.update("clientes", valores, "_id=" + cod, null);

            Toast.makeText(getApplicationContext(), "Registro actualiza-
do", Toast.LENGTH_SHORT).show();

        } else
            Toast.makeText(getApplicationContext(), "Para actualizar el
registro es necesario el codigo de cliente", Toast.LENGTH_SHORT).show();

        //Vaciamos los campos
        txtId.setText("");
        txtNombre.setText("");
        txtTelefono.setText("");
        txtEmail.setText("");

        db.close();
    }
});

btnEliminar.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {

        db = usdbh.getWritableDatabase();

```

```

        //Recuperamos los valores de los campos de texto
        String cod = txtId.getText().toString();

        if (cod.length() == 0) {
            Toast.makeText(getApplicationContext(), "Debes introducir un
codigo para poder eliminar usuario", Toast.LENGTH_SHORT).show();

            //Vaciamos los campos
            txtId.setText("");
            txtNombre.setText("");
            txtTelefono.setText("");
            txtEmail.setText("");
        }
        else {
            //Alternativa 1: método execSQL()
            String sql = "DELETE FROM clientes WHERE _id=" + cod;
            db.execSQL(sql);
            Toast.makeText(getApplicationContext(), "Registro borra-
do", Toast.LENGTH_SHORT).show();

            //Vaciamos los campos
            txtId.setText("");
            txtNombre.setText("");
            txtTelefono.setText("");
            txtEmail.setText("");
        }

        db.close();
    }
});

new ClientesProvider();
}
}

```

AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ej_contentprovider_02">

    <application

        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <!-- Para que el contentProvider sea visible para otras aplicaiones-->
        <provider
            android:authorities="com.example.ej_contentprovider_02"
            android:name = "com.example.ej_contentprovider_02.ClientesProvider"

```

```

        android:exported="true"/>

    </application>

</manifest>

```

Aplicación **EJ_ContentProvider_03**: Aplicación que consulta y/o modifica el ContenProvider creado anteriormente. Su interfaz gráfica tendrá varios botones para consultar e insertar nuevos elementos.



Activity main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <!--<LinearLayout
        android:layout_width="match_parent"
        android:layout height="wrap content"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/id" />

        <EditText
            android:id="@+id/txtCodigo"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="5"
            android:ems="10">

            <requestFocus />
        </EditText>
    </LinearLayout>

```



```

</LinearLayout>
-->
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/nombre" />

    <EditText
        android:id="@+id/txtNombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="5"
        android:ems="10">

        <requestFocus />
    </EditText>
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/telefono" />

    <EditText
        android:id="@+id/txtTelefono"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:ems="10"
        android:inputType="text"/>
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/mail" />

    <EditText
        android:id="@+id/txtEmail"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:ems="10"
        android:inputType="text"/>
</LinearLayout>

```

```

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button
        android:id="@+id/btnLeer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Leer clientes"
        android:layout_weight="1" />

    <Button
        android:id="@+id/btnEscribir"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Nuevo Registro"
        android:layout_weight="1" />

</LinearLayout>

<TextView
    android:id="@+id/tvListado"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"/>

</LinearLayout>

```

LeerClientesProvider.java

```

package com.example.ej_contentprovider_03;

import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.net.Uri;
import android.util.Log;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

public class LeerClientesProvider {
    private Activity activity;

    public LeerClientesProvider (Activity activity){
        this.activity = activity;
    }

    public void guardarCliente (String nombre, String telefono, String email){
        Uri uri =
        Uri.parse("content://com.example.ej_contentprovider_02/clientes");

        ContentValues valores = new ContentValues();
        valores.put ("nombre", nombre);
        valores.put ("telefono", telefono);
        valores.put ("email", email);
        try {
            activity.getContentResolver().insert(uri, valores);
        } catch (Exception e) {
            Toast.makeText(activity, "Verificar que está instalado "+

```

```

        "com.example.ej_contentprovider_02",
        Toast.LENGTH_SHORT).show();
        Log.e("LeerClientesProvider", "Error: "+e.toString(), e);
    }
}

public List<String> listaClientes () {
    List<String> result = new ArrayList<String>();
    Uri uri=
Uri.parse("content://com.example.ej_contentprovider_02/clientes");
    Cursor cursor = activity.getContentResolver().query(uri, null,
null, null, "nombre DESC");

    if (cursor != null) {
        while (cursor.moveToNext()) {
            String nombre = cursor.getString(cursor.getColumnIndex("nombre"));
            String telefono = cur-
sor.getString(cursor.getColumnIndex("telefono"));
            String email = cursor.getString(cursor.getColumnIndex("email"));

            result.add (nombre + " " + telefono + " " + email );
        }
    }
    return result;
}
}

```

MainActivity.java

```

package com.example.ej_contentprovider_03;

import androidx.appcompat.app.AppCompatActivity;

import android.content.ContentValues;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import java.util.List;

public class MainActivity extends AppCompatActivity {

    public Button btnLeer;
    public Button btnNuevo;
    public TextView tvListado;

    private EditText txtId;
    private EditText txtNombre;
    private EditText txtTelefono;
    private EditText txtEmail;

    public LeerClientesProvider leerClientesP;

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

```

```

leerClientesP = new LeerClientesProvider(this);

btnLeer = findViewById(R.id.btnLeer);
btnNuevo = findViewById(R.id.btnEscribir);
tvListado = findViewById(R.id.tvListado);

//Obtenemos las referencias a los controles
txtNombre = (EditText) findViewById(R.id.txtNombre);
txtTelefono = (EditText) findViewById(R.id.txtTelefono);
txtEmail = (EditText) findViewById(R.id.txtEmail);

btnLeer.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        List<String> miListaClientes = leerClientesP.listaClientes();

        tvListado.setText("");
        int cantClientes = miListaClientes.size();
        for (int i = 0; i < cantClientes; i++) {
            tvListado.setText(tvListado.getText() + miListaClientes.get(i)
+ "\n");

            Log.i("lista", miListaClientes.get(i));
        }
    }
});

btnNuevo.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {

        //Recuperamos los valores de los campos de texto
        String nomb = txtNombre.getText().toString();
        String telf = txtTelefono.getText().toString();
        String mail = txtEmail.getText().toString();

        if (nomb.length() == 0 || telf.length() == 0 || mail.length() ==
0) {
            Toast.makeText(getApplicationContext(), "Hay que introducir
todos los campos para insertar un nuevo registro", Toast.LENGTH_SHORT).show();
        } else {
            //Vaciamos campos
            txtNombre.setText("");
            txtTelefono.setText("");
            txtEmail.setText("");
            leerClientesP.guardarCliente(nomb, telf, mail);
        }
    }
});
}

```

ACTIVIDADES Content Provider

1.- Realizar el acceso a un content provider de los de la propia plataforma Android. Puedes consultar los datos disponibles en [paquete android.provider](#) (por ejemplo: historial de llamadas, lista de contactos y teléfonos, las bibliotecas multimedia, ...)

2.- Crea y utiliza/consulta un content provider creado por ti.
