Android

Content Providers

Content-Providers

- **Persistencia:** Manejar datos y exponerlos a otras aplicaciones.
- Interfaz con un conjunto de métodos estándar.
- Único modo de compartir datos entre aplicaciones.
- Android provee numerosos *ContentProviders* básicos (audio, video, imágenes, contactos, etc.)
- Paquete *android.provider*

Content-Providers

- Hacer tus datos públicos:
  - Crear tu propio Content-Provider (extender subclase ContentProvider).
  - Añadir los datos a uno existente:
    - Mismo tipo de datos
    - Permisos para escribir

Content Resolver

- Interfaz común para
  - Hacer consultas
  - Proveer resultados
  - Insertar, modificar y borrar datos.
- ContentResolver: Permite trabajar con los ContentProviders:
  - Se obtiene llamando al método *getContentResolver()* dentro de la implementación de una Activity u otro componente de la aplicación.
- Android identifica el ContentProvider objetivo de la query y se asegura de que está levantado y ejecutándose.
- El sistema se encarga de instanciar todos los ContentProvider.
- Normalmente, solo hay una instancia para cada tipo de ContentProvider
- El contentProvider puede comunicarse con ContentResolver de otras aplicaciones.
- ContentResolver y ContentProvider se encargan de la comunicación entre procesos

Modelo de datos

_ID	NUMBER	NUMBER_KEY	LABEL	NAME	TYPE
13	(425) 555 6677	425 555 6677	Kirkland office	Bully Pulpit	TYPE_WORK
44	(212) 555-1234	212 555 1234	NY apartment	Alan Vain	TYPE_HOME
45	(212) 555-6657	212 555 6657	Downtown office	Alan Vain	TYPE_MOBILE
53	201.555.4433	201 555 4433	Love Nest	Rex Cars	TYPE_HOME

URI

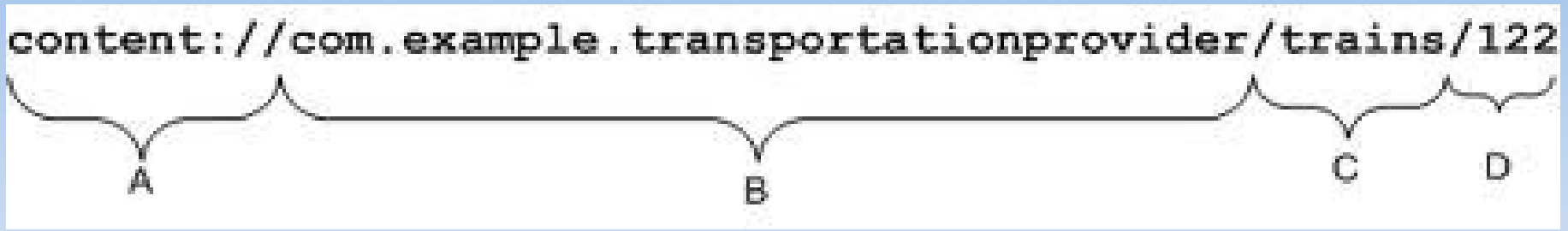
- URI identifica el conjunto de datos (tabla).
- Una URI distinta para cada tabla
- Un ContentProvider expone varias URIs
- Es buena idea definir una constante para la URI. Ejemplos:

  android.provider.Contacts.Phones.CONTENT_URI

  android.provider.Contacts.Photos.CONTENT_URI

- Cada método de ContentResolver lleva como primer argumento la URI. Indica con qué ContentProvider y con qué tabla debe hablar.

URI



- **A:** Prefijo estándar. Nunca modificar
- **B:** Authority. Identifca el CP, fully-qualified class name, así se asegura unicidad.
- **C:** Determina el tipo de datos a manejar. Puede tener cero (si solo maneja un tipo) o más segmentos.
- **D:** Identificador de la fila (_ID). Va vacío si hace una query a un conjunto de filas.

Content-Provider Query

- Se necesitan tres cosas:
  - URI
  - Nombre de los campos (columnas)
  - Tipo de los campos (columnas)
- ContentResolver.query()
- Activity.managedQuery()
  - La activity controla el ciclo de vida del Cursor
  - Se descarga cuando la activity se pausa
  - Hace “requery” cuando se reinicia.
- Activity.startManagingCursor(Cursor c):

  Provoca que la activity maneje el Cursor

Content-Provider Query

- *public final Cursor **managedQuery** (Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)*
- **uri:** La URI del ContentProvider a consultar.
  - Se puede indicar el id de la fila: *content://. . . ./23*
- **projection:** Columnas a retornar.
- **selection:** Cláusula SQL WHERE.
- **selectionArgs:** Argumentos de la selection para completar (“?” PreparedStatements).
- **sortOrder:** SQL ORDER BY clause.

Content-Provider Query

- Ejemplo helpers para uris
- Valor *null* retorna todas las columnas/filas

```
import android.provider.Contacts.People;
import android.content.ContentUris;
import android.net.Uri;
import android.database.Cursor;

// Use the ContentUris method to produce the base URI for the contact with _ID == 23.
Uri myPerson = ContentUris.withAppendedId(People.CONTENT_URI, 23);

// Alternatively, use the Uri method to produce the base URI.
// It takes a string rather than an integer.
Uri myPerson = Uri.withAppendedPath(People.CONTENT_URI, "23");

// Then query for this specific record:
Cursor cur = managedQuery(myPerson, null, null, null, null);
```

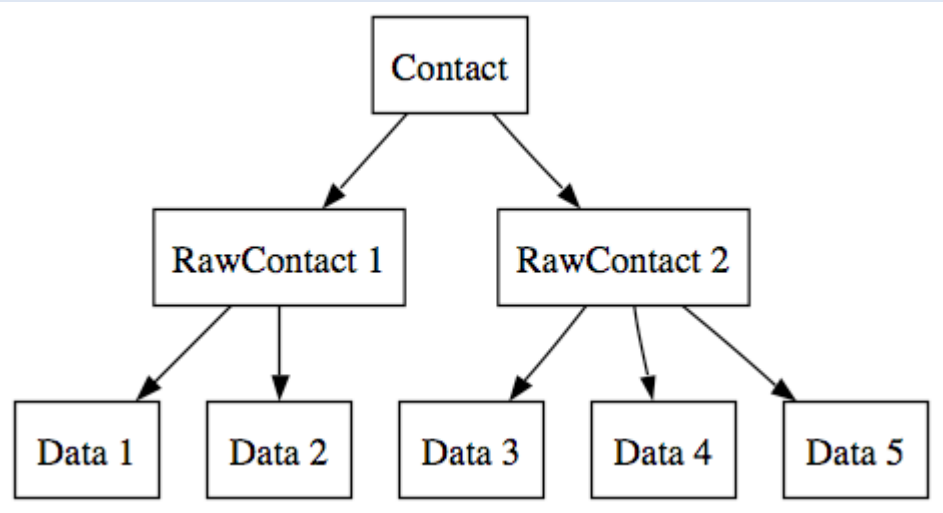
[Content-Provider] Query

- Interfaz BaseColumns: _ID, _COUNT

```
import android.provider.Contacts.People;
import android.database.Cursor;

// Form an array specifying which columns to return.
String[] projection = new String[] {
                              People._ID,
                              People._COUNT,
                              People.NAME,
                              People.NUMBER
                          };

// Get the base URI for the People table in the Contacts content provider.
Uri contacts =  People.CONTENT_URI;

// Make the query.
Cursor managedCursor = managedQuery(contacts,
                        projection, // Which columns to return
                        null,       // Which rows to return (all rows)
                        null,       // Selection arguments (none)
                        // Put the results in ascending order by name
                        People.NAME + " ASC");
```

Nueva API 2.0

- Tres tipos de tablas: Contact, Raw Contact, Data
- Data:
  - Tabla genérica. Guarda todo tipo de información (Nº Tlf, Email, Foto...)
  - Cada fila indica el tipo a través de MIME type
  - ContactsContract.CommonDataKinds → Subclases para tipos comunes
  - Se pueden definir propios MIME types.
- Una fila de RawContact representa un conjunto de *Data's* asociada a una fuente de contactos. Ej. Contacto de google, facebook, etc.
- Una fila de Contact representa un conjunto de *RawContact's* de un mismo contacto



[ContentProvider] Nueva API

```
import android.provider.ContactsContract.Data;
import android.provider.ContactsContract.CommonDataKinds.Phone;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;

public class ChoosePhoneActivity extends ListActivity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.list);
        setTitle("Choose a phone");

        // Query: contacts with phone shorted by name
        Cursor mCursor = getContentResolver().query(
                Data.CONTENT_URI,
                new String[] { Data._ID, Data.DISPLAY_NAME, Phone.NUMBER, Phone.TYPE },
                Data.MIMETYPE + "='" + Phone.CONTENT_ITEM_TYPE
                    + "' AND "+ Phone.NUMBER + " IS NOT NULL", null,
                Data.DISPLAY_NAME + " ASC");

        startManagingCursor(mCursor);

        // Setup the list
        ListAdapter adapter = new SimpleCursorAdapter(this, // context
                android.R.layout.simple_list_item_2, // Layout for the rows
                mCursor, // cursor
                new String[] { Data.DISPLAY_NAME, Phone.NUMBER }, // cursor fields
                new int[] { android.R.id.text1, android.R.id.text2 } // view
                                                                     // fields
        );
        setListAdapter(adapter);
```

ContactsContract.Data

- Data es una tabla genérica, según el MIMETYPE, determina el significado de las columnas DATA1 a DATA15.
  - Ej. Phone.CONTENT_ITEM_TYPE → DATA1= Número de teléfono
  - Ej. Email.CONTENT_ITEM_TYPE → DATA1 = email
- Se definen tipos comunes de datos, como ContactsContract.CommonDataKinds.Phone o ContactsContract.CommonDataKinds.Email, y sus alias. Ej:
  - Phone.NUMBER = Data.DATA1
- Data.DATA1 está indexada.
- Data.DATA15 se usa para BLOBs

Leer datos del cursor

- Debes saber el tipo de dato de la columna
- Cursor.getBlob(). Devuelve byte[]
- ContentResolver.openInputStream() - leer fichero

```
import android.provider.Contacts.People;

private void getColumnData(Cursor cur){
    if (cur.moveToFirst()) {

        String name;
        String phoneNumber;
        int nameColumn = cur.getColumnIndex(People.NAME);
        int phoneColumn = cur.getColumnIndex(People.NUMBER);
        String imagePath;

        do {
            // Get the field values
            name = cur.getString(nameColumn);
            phoneNumber = cur.getString(phoneColumn);

            // Do something with the values.
            ...

        } while (cur.moveToNext());

    }
}
```

Modificar datos

- Añadir registros
- Añadir nuevos valores a registros existentes
- Batch updating
- Borrar registros

- Se necesita tener permisos! Ej.

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

Insertar datos
Ej. Contactos API 1.x

- Se utiliza un Map (*ContentValues*) que relacione columna y *valor*
- Llamar a ContentResolver.insert(URI, *ContentValues*)
- Devuelve la URI completa, referente al registro insertado
- Con esta URI, puedes pedir un cursor.

```
import android.provider.Contacts.People;
import android.content.ContentResolver;
import android.content.ContentValues;

ContentValues values = new ContentValues();

// Add Abraham Lincoln to contacts and make him a favorite.
values.put(People.NAME, "Abraham Lincoln");

// 1 = the new contact is added to favorites
// 0 = the new contact is not added to favorites
values.put(People.STARRED, 1);

Uri uri = getContentResolver().insert(People.CONTENT_URI, values);
```

Añadir datos
Ej. Contactos API 2.x

```
ArrayList<ContentProviderOperation> ops = new ArrayList<ContentProviderOperation>();
        ops.add(ContentProviderOperation.newInsert(ContactsContract.RawContacts.CONTENT_URI)
                .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, "accountname@gmail.com")
                .withValue(ContactsContract.RawContacts.ACCOUNT_NAME, "com.google")
                .build());
        ops.add(ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
                .withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)
                .withValue(ContactsContract.Data.MIMETYPE,
                        ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE)
                .withValue(ContactsContract.CommonDataKinds.StructuredName.DISPLAY_NAME, name)
                .build());
       ops.add(ContentProviderOperation.newInsert(ContactsContract.Data.CONTENT_URI)
                .withValueBackReference(ContactsContract.Data.RAW_CONTACT_ID, 0)
                .withValue(ContactsContract.Data.MIMETYPE,
                        ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE)
                .withValue(ContactsContract.CommonDataKinds.Phone.NUMBER, phone)
                .withValue(ContactsContract.CommonDataKinds.Phone.TYPE,
                                        ContactsContract.CommonDataKinds.Phone.TYPE_HOME)
                .build());

        try {
                cr.applyBatch(ContactsContract.AUTHORITY, ops);
            } catch (RemoteException e) {
                e.printStackTrace();
            } catch (OperationApplicationException e) {
                e.printStackTrace();
            }
```

Update y Delete

- *public final int **update** (Uri uri, ContentValues values, String where, String[] selectionArgs)*

  Devuelve el número de filas actualizadas

- *public final int **delete** (Uri url, String where, String[] selectionArgs)*

  Devuelve el número de filas borradas.

  Si el ContentProvider permite transacciones, la operación es atómica.

Crear un ContentProvider!!! Pasos

- Instalar un sistema para guardar datos:
  - SQLite: Viene la librería en Android
- Extender la clase ContentProvider
  - query()
  - insert()
  - update()
  - delete()
  - getType()
  - onCreate()
- Declarar el ContentProvider en el Manifest

Subclase ContentProvider

- query()

  Retorna un Cursor. Android provee Cursores prefabricados, como SQLiteCursor, Matrix Cursor.
- Deben programarse de modo Thread-safe, ya que un CP puede llamarse desde varios CR
- ContentResolver.notifyChange() avisa a los listener de que han cambiado los datos

Subclase ContentProvider Consejos

- Definir la URI como constante:

  public static final Uri CONTENT_URI = Uri.parse("content://com.example.codelab.transportationprovider");

- Definir URIs para subtablas si tiene, con el mismo authority

  content://com.example.codelab.transportationprovider/train

  content://com.example.codelab.transportationprovider/air/domestic

- Definir constantes para las columnas
  - Consejo: Mismo nombre que las columnas de las tablas
  - Definir una columna “_id” con la constante “_ID”
  - Si usas SQLite: INTEGER PRIMARY KEY [AUTOINCREMENT]
  - Con AUTOINCREMENT – Al insertar fila, coge el valor como si fuera una seq
  - Sin AUTOINCREMENT – Al insertar file, coge el maximo de los _ID
- Documentar el tipo de dato de cada columna (para los clientes)

Subclase ContentProvider Consejos

- Si defines un nuevo tipo de datos, define un MIME implementando ContentProvider.getType().
- El MIME depende de la URI, no es lo mismo pedir un registro que varios registros.
- Para saber el MIME que tienes que devolver, ayúdate de la URI para distinguir si lo que piden.

For a single record: vnd.android.cursor**.item**/vnd.yourcompanyname.contenttype
For example, a request for train record 122, like this URI,
content://com.example.transportationprovider/trains/122
might return this MIME type:
vnd.android.cursor.item/vnd.example.rail

For multiple records: vnd.android.cursor**.dir**/vnd.yourcompanyname.contenttype
For example, a request for all train records, like the following URI,
content://com.example.transportationprovider/trains
might return this MIME type:
vnd.android.cursor.dir/vnd.example.rail

ContentResolver Subclase Consejos

Definir un dato binario muy grande como para poner en tabla:

- El campo de ese dato debería devolver una content URI
- El CP debería también definir un campo _data con la ruta del fichero
- Con la URI obtenida, el cliente llama a ContentResolver.openInputStream()
- Entonces, internamente el CP hace una petición al campo _data.
- ¿Porque? El CP tiene más permisos y solamente ofrece un wrapper al cliente.

Declarar el ContentProvider

- Etiqueta <provider> en el AndroidManifest.xml
  - name: nombre de la subclase CP
  - authorities: Parte authority de la URI, omitiendo la parte “content://”
  - Especificar permisos
  - multiprocess: true → Permite una instancia del CP en cada cliente. Se ahorra IPC.

ContentProvider

SQLite!

SQLite

- Cualquier BD que crees será accesible solo dentro de tu aplicación.
- Para crear una BD en SQLite se recomienda extender subclase de *SQLiteOpenHelper* y sobreescribir *onCreate(),* incluyendo el código de creación de BD.
- Instance objeto SQLiteOpenHelper y ...
  - Para escribir en BD: helper.getWritableDatabase()
  - Para leer de BD: getReadableDatabase()
- Devuelven objeto SQLiteDatabase, con métodos para operar con BD:
  - *public Cursor query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)*
  - *public long insert (String table, String nullColumnHack, ContentValues values)*
  - *public int delete (String table, String whereClause, String[] whereArgs)*
  - *public int update (String table, ContentValues values, String whereClause, String[] whereArgs)*

SQLite

- Implementación de ejemplo:

  http://thinkandroid.wordpress.com/2010/01/13/writing-your-own-conte ntpr

- ORM

  http://ormlite.com/sqlite_java_android_orm.shtml