CONTENT PROVIDERS

COMP 41690

DAVID COYLE > D.COYLE@UCD.IE

CONTENT PROVIDERS

Content providers manage access to a structured set of data. They encapsulate data and provide a standard interface through which other applications can access the data.

Android has many inbuilt content providers. E.g. contacts data is used by multiple applications and is stored/accessed through a content provider.

Example of when you might need to build you own content provider:

- You need to share data between multiple applications.
- Or you want to offer complex data or files to other applications
- Or allow users to copy and paste complex data into other apps
- Or you want to provide custom search suggestions in your own application.

DATA PRESENTATION

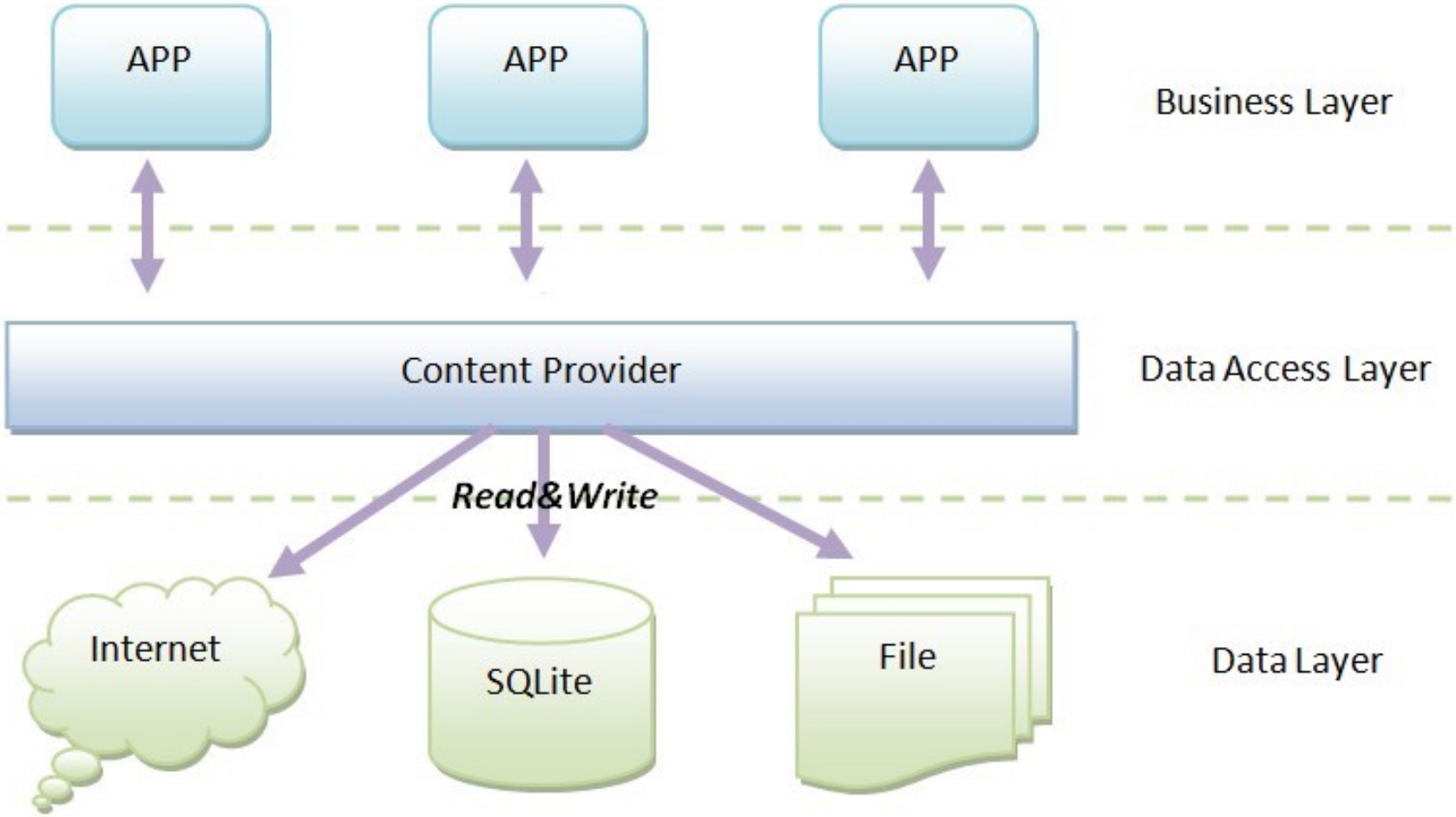
A content provider presents data to external applications as one or more tables that are similar to the tables found in a relational database:

- Each row represents an instance of some type of data the provider collects
- Each column in the row represents an individual piece of data collected for an instance.

For example, one of the built-in providers in the Android platform is the user dictionary, which stores the spellings of non-standard words that the user wants to keep.

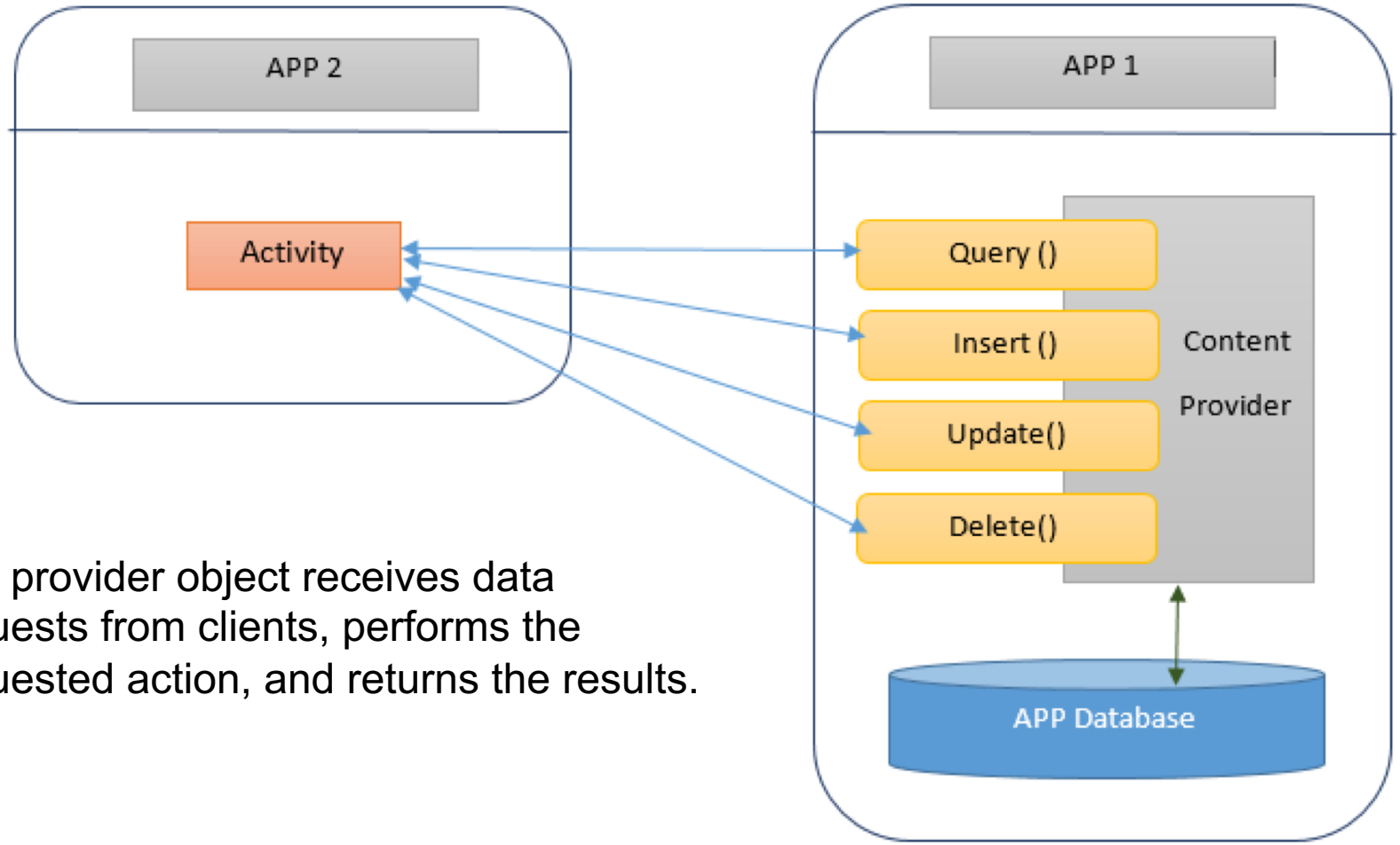
Table 1: Sample user dictionary table.

word	app id	frequency	locale	_ID
mapreduce	user1	100	en_US	1
precompiler	user14	200	fr_FR	2
applet	user2	225	fr_CA	3
const	user1	255	pt_BR	4
int	user5	100	en_UK	5


APP
APP
APP
Business Layer
Content Provider
Data Access Layer
Read&Write
Internet
SQLite
File
Data Layer

Client application

Provider:
implements ContentProvider

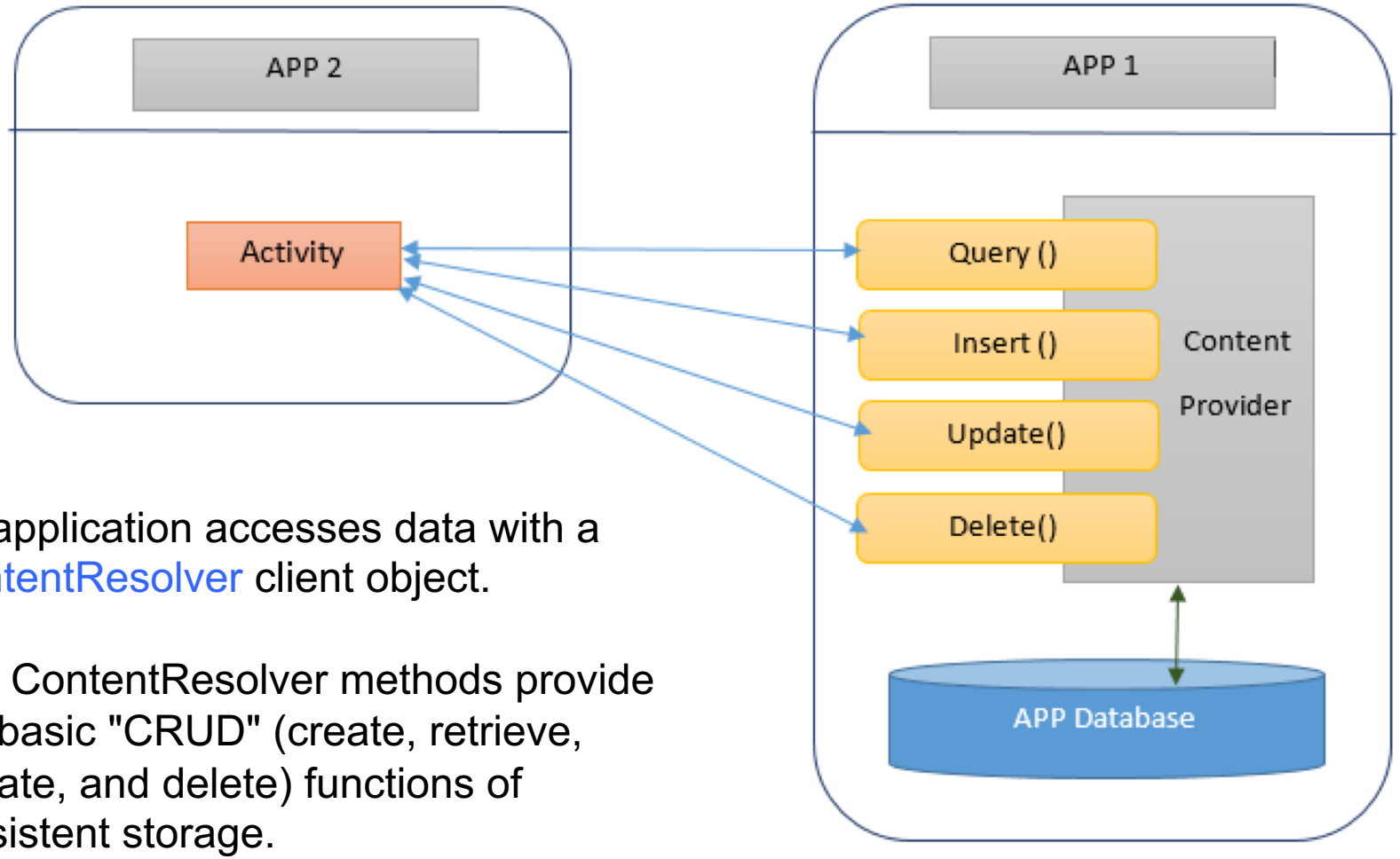


The provider object receives data requests from clients, performs the requested action, and returns the results.

ContentResolver.query() ⟶ ⟵ ContentProvider.query()

Client application

Provider:
implements ContentProvider



An application accesses data with a ContentResolver client object.

The ContentResolver methods provide the basic "CRUD" (create, retrieve, update, and delete) functions of persistent storage.

ContentResolver.query() ⇄ ContentProvider.query()

CREATING A CONTENT PROVIDER

1. Create a Content Provider class that extends the **ContentProvider** base class.

2. Define a content **URI address** which identifies the data in the provider and is used to access the data.

3. Create your own data storage system (e.g. a database) to keep the content. Usually, Android uses SQLite database.

4. Implement Content Provider methods to perform different CRUD operations.

5. Finally register your Content Provider in your activity file using <provider> tag.

CREATING A CONTENT PROVIDER

1. Create a Content Provider class that extends the **ContentProvider** base class.

2. Define a content **URI address** which identifies the data in the provider and is used to access the data.

3. Create your own data storage system (e
content. Usually, Android uses SQLite d

4. Implement Content Provider methods to
operations.

5. Finally register your Content Provider in
<provider> tag.

A content provider that controls multiple data sets (multiple tables) exposes a separate URI for each one.

All URIs for providers begin with the string "content://". This identifies the data as being controlled by a content provider.

CREATING A CONTENT PROVIDER

1. Create a Content Provider class that extends the **ContentProvider** base class.

2. Define a content **URI address** which identifies the data in the provider and is used to access the data.

3. **Create your own data storage system (e.g. a database) to keep the content. Usually, Android uses SQLite database.**

4. Implement Content Provider methods to perform different CRUD operations.

5. Finally register your Content Provider in your activity file using <provider> tag.

DESIGNING DATA STORAGE

A content provider is the interface to data saved in a structured format. Before you create the interface, you must decide how to store the data.

These are some of the data storage technologies that are available in Android:

- The Android system includes an SQLite database API that Android's own providers use to store table-oriented data. The SQLiteOpenHelper class helps you create databases, and the SQLiteDatabase class is the base class for accessing
- For storing file data, Android has a variety of file-oriented APIs. If you're designing a provider that offers media-related data such as music or videos, you can have a provider that combines table data and files.
- For working with network-based data, use classes in java.net and android.net. You can also synchronize network-based data to a local data store such as a database, and then offer the data as tables or files.

For more details on designing data storage see:
https://developer.android.com/guide/topics/providers/content-provider-creating.html

CREATING A CONTENT PROVIDER

1. Create a Content Provider class that extends the **ContentProvider** base class.

2. Define a content **URI address** which identifies the data in the provider and is used to access the data.

3. Create your own data storage system (e.g. a database) to keep the content. Usually, Android uses SQLite database.

4. Implement Content Provider methods to perform different CRUD operations.

5. Finally register your Content Provider in your activity file using <provider> tag.

CREATING A CONTENT PROVIDER

Create a Content Provider class that extends the **ContentProvider** baseclass.

You then need to override specific methods in Content Provider class:

onCreate() This method is called when the provider is started.

query() This method receives a request from a client. The result is returned as a **Cursor object**.

insert() This method inserts a new record into the content provider.

delete() This method deletes an existing record from the content provider

update() This method updates an existing record from the content provider.

getType() This method returns the MIME type of the data at the given URI.

```java
package org.davidcoyle.myapplication;

import ...

public class MyContentProvider extends ContentProvider {
    public MyContentProvider() {
    }

    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {
        // Implement this to handle requests to delete one or more rows.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public String getType(Uri uri) {
        // TODO: Implement this to handle requests for the MIME type of the data
        // at the given URI.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        // TODO: Implement this to handle requests to insert a new row.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public boolean onCreate() {
        // TODO: Implement this to initialize your content provider on startup.
        return false;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                        String[] selectionArgs, String sortOrder) {
        // TODO: Implement this to handle query requests from clients.
        throw new UnsupportedOperationException("Not yet implemented");
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
                      String[] selectionArgs) {
        // TODO: Implement this to handle requests to update one or more rows.
        throw new UnsupportedOperationException("Not yet implemented");
    }
}
```

DECLARING CONTENT PROVIDER

```
<provider
    android:name=".MyProvider"
    android:authorities="com.example.contentproviderexample.MyProvider">
</provider>
```

Note: here authorities is the URI authority to access the content provider.

Typically this will be the fully qualified name of the content provider class.

ACCESS A CONTENT PROVIDER

To access data from a provider, follow these basic steps:

1. Request the read access permission for the provider.
2. Define the code that sends a query to the provider.

An client application accesses the data from a content provider with a **ContentResolver** client object.

The **ContentResolver** methods match and call identically-named methods in the provider object. It provides the basic "CRUD" (create, retrieve, update, and delete) functions of persistent storage.

QUERY EXAMPLES

Return all rows:

Cursor allRows =

getContentResolver().query(MyProvider.
CONTENT_URI, null, null, null, null);

```
// Queries the user dictionary and returns results
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,   // The content URI of the words table
    mProjection,                        // The columns to return for each row
    mSelectionClause                    // Selection criteria
    mSelectionArgs,                     // Selection criteria
    mSortOrder);                        // The sort order for the returned rows
```

Table 2: Query() compared to SQL query.

query() argument	SELECT keyword/parameter	Notes
`Uri`	`FROM table_name`	`Uri` maps to the table in the provider named *table_name*.
`projection`	`col,col,col,...`	`projection` is an array of columns that should be included for each row retrieved.
`selection`	`WHERE col = value`	`selection` specifies the criteria for selecting rows.
`selectionArgs`	(No exact equivalent. Selection arguments replace `?` placeholders in the selection clause.)	
`sortOrder`	`ORDER BY col,col,...`	`sortOrder` specifies the order in which rows appear in the returned `Cursor`.

DEMO

1. Go to the Moodle and download the file "Content_providers_example.zip"
2. Unzip and it contains two Android Studio projects/apps.
    1. A Content Provider app.
    2. A client app.
3. Open both projects in Android Studio and run both on a phone or emulator.

Note: When you examine the code you will notice that the MyUser app does not call ContentResolver directly.

In order to improve thread safety, most clients will not call **ContentResolver.query()** directly. It is better to do queries asynchronously on a separate thread. One way to do this is to use the **CursorLoader class**, which is described in more detail in the **Loaders** guide.

1. MyProvider extends the ContentProvider base class.

MyProvider.java

```
package com.example.contentproviderexample;

import ...

public class MyProvider extends ContentProvider {

    static final String PROVIDER_NAME = "com.example.contentproviderexample.MyProvider";
    static final String URL = "content://" + PROVIDER_NAME + "/cte";
    static final Uri CONTENT_URI = Uri.parse(URL);

    static final String id = "id";
    static final String name = "name";
    static final int uriCode = 1;
    static final UriMatcher uriMatcher;

    private static HashMap<String, String> values;
    static {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(PROVIDER_NAME, "cte", uriCode);
        uriMatcher.addURI(PROVIDER_NAME, "cte/*", uriCode);
    }
```

2. It defines a content URI.

CONTENT PROVIDER URI

Providers specify the query string in the form of a URI which has following format: <prefix>://<authority>/<path>/<id>

<prefix>

content:// All the content provider URIs should start with this value

<authority>

'authority' is Java namespace of the content provider implementation. (fully qualified Java package name)

<path>

'path' (sometimes referred to as <data-type>) is the virtual directory within the provider that identifies the data being requested.

<id>

'id' is optional part that specifies the primary key of a record being requested. We can omit this part to request all records.

```
Uri.parse("content://com.example.contentproviderexample.MyProvider/cte")
```

3. MyProvider creates its own data storage system.

MyProvider.java

```
    private SQLiteDatabase db;
    static final String DATABASE_NAME = "mydb";
    static final String TABLE_NAME = "names";
    static final int DATABASE_VERSION = 1;
    static final String CREATE_DB_TABLE = " CREATE TABLE " + TABLE_NAME
            + " (id INTEGER PRIMARY KEY AUTOINCREMENT, "
            + " name TEXT NOT NULL);";

    private static class DatabaseHelper extends SQLiteOpenHelper {
        DatabaseHelper(Context context) { super(context, DATABASE_NAME, null, DATABASE_VERSION); }

        @Override
        public void onCreate(SQLiteDatabase db) { db.execSQL(CREATE_DB_TABLE); }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
            onCreate(db);
        }
    }
```

4. MyProvider implements CRUD methods.

MyProvider.java

```
    @Override
    public int delete(Uri uri, String selection, String[] selectionArgs) {...}


    @Override
    public Uri insert(Uri uri, ContentValues values) {...}


    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
            String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
        qb.setTables(TABLE_NAME);

        switch (uriMatcher.match(uri)) {
        case uriCode:
            qb.setProjectionMap(values);
            break;
        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
        }
        if (sortOrder == null || sortOrder == "") {
            sortOrder = name;
        }
        Cursor c = qb.query(db, projection, selection, selectionArgs, null,
                null, sortOrder);
        c.setNotificationUri(getContext().getContentResolver(), uri);
        return c;
    }

    @Override
    public int update(Uri uri, ContentValues values, String selection,
            String[] selectionArgs) {...}
```

CURSORS

Cursors are a standard method to represent a 2 dimensional table of any data. This could be a relational database, but …

The use of Content Providers and Cursors in Android basically allows you to access different types of data as if you were working with a relational database.

A Cursor represents the result of a query and basically points to one row of the query result.

You can then step through the rows, one by one.

AndroidManifest.xml

manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.contentproviderexample"
    android:versionCode="1"
    android:versionName="1.0" >

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="ContentProviderExample"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.contentproviderexample.MainActivity"
            android:label="ContentProviderExample" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider
            android:name=".MyProvider"
            android:authorities="com.example.contentproviderexample.MyProvider"
            android:exported="true"
            android:multiprocess="true" >
        </provider>
    </application>

</manifest>
```

5. MyProvider is registered in the manifest file.

MainActivity.java

```
package com.example.contentprovideruser;

import ...

public class MainActivity extends FragmentActivity implements LoaderManager.LoaderCallbacks<Cursor> {
    TextView resultView=null;
    CursorLoader cursorLoader;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {...}

    public void onClickDisplayNames(View view) { getSupportLoaderManager().initLoader(1, null, this); }

    @Override
    public Loader<Cursor> onCreateLoader(int arg0, Bundle arg1) {
        cursorLoader= new CursorLoader(this, Uri.parse("content://com.example.contentproviderexample.MyProvider/cte"), null, null, null, null);
        return cursorLoader;
    }

    @Override
    public void onLoadFinished(Loader<Cursor> arg0, Cursor cursor) {
        cursor.moveToFirst();
        StringBuilder res=new StringBuilder();
        while (!cursor.isAfterLast()) {
            res.append("\n"+cursor.getString(cursor.getColumnIndex("id"))+ "-"+ cursor.getString(cursor.getColumnIndex("name")));
            cursor.moveToNext();
        }
        resultView.setText(res);
    }

    @Override
    public void onLoaderReset(Loader<Cursor> arg0) {
        // TODO Auto-generated method stub

    }

    @Override
    public void onDestroy() { super.onDestroy(); }

}
```

MyUser is a client. It needs to:

1. Request the read access permission for the provider.

2. Define the code that sends a query to the provider.

CONTENT RESOLVERS AND LOADERS

We have said previously:

An client application accesses the data from a content provider with a **ContentResolver** client object.

The **ContentResolver** methods match and call identically-named methods in the provider object. It provides the basic "CRUD" (create, retrieve, update, and delete) functions of persistent storage.

You can retrieve a ContentResolver by getting it from your applications Context: context.getContentResolver.query().

In practice using a **ContentResolver** directly is not always the best option.

ContentResolver is not thread safe. It should not be used on a main UI thread and need to be implemented in asynchronously on a separate thread.

One way to do this is to use the CursorLoader class, which is a subclass of Loader.

LOADERS AND CURSORLOADERS

Introduced in Android 3.0, Loaders allow you to asynchronously load data in an activity or fragment:

- They are available to every Activity and Fragment.
- They provide asynchronous loading of data.
- They monitor the source of their data and deliver new results when the content changes.
- They automatically reconnect to the last loader's cursor when being recreated after a configuration change. Thus, they don't need to re-query their data.

https://developer.android.com/guide/components/loaders.html#summary

A **CursorLoader** is a loader that queries a ContentResolver and returns a Cursor.

It implements the Loader protocol in a standard way for querying cursors, building on AsyncTaskLoader to perform the cursor query on a background thread so that it does not block the application's UI.

https://developer.android.com/reference/android/content/CursorLoader.html

INBUILT PROVIDERS

Android makes a lot of content providers available to your apps (providing you request / get the correct permissions)

- Calendar
- CallLog
- Contacts
- MediaStore
- Settings
- UserDictionary

Every content provider exposes a public URI (wrapped as a **Uri** object) that uniquely identifies its data set. A content provider that controls multiple data sets (multiple tables) exposes a separate URI for each one.

See a list in the reference documentation for the android.provider package:

https://developer.android.com/reference/android/provider/package-summary.html

EXAMPLE: CALENDARS

Calendars use a data model where data is stored in a number of tables:

Calenders
Events
Attendees
Reminders
Instances

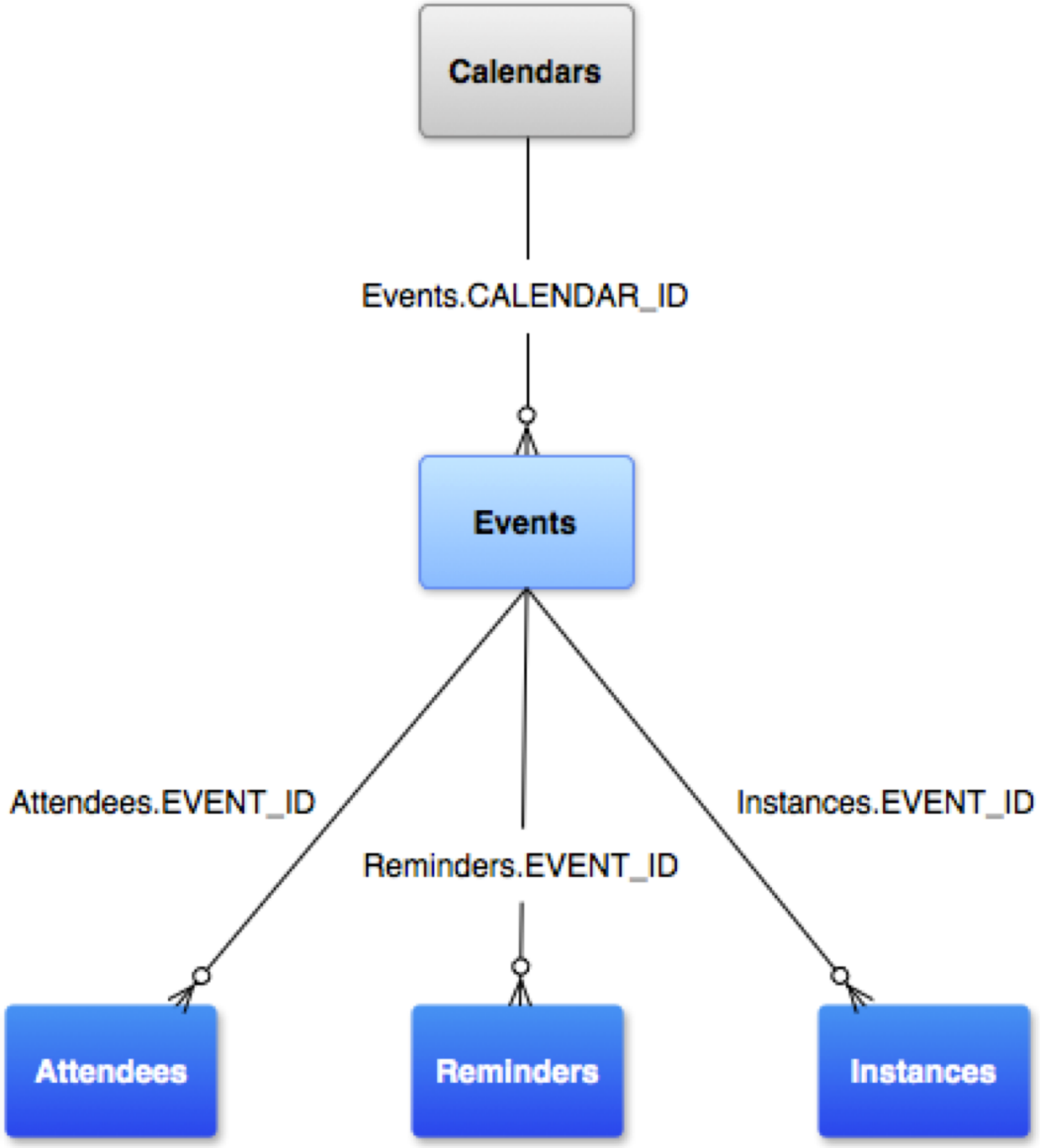


Table (Class)	Description
`CalendarContract.Calendars`	This table holds the calendar-specific information. Each row in this table contains the details for a single calendar, such as the name, color, sync information, and so on.
`CalendarContract.Events`	This table holds the event-specific information. Each row in this table has the information for a single event—for example, event title, location, start time, end time, and so on. The event can occur one-time or can recur multiple times. Attendees, reminders, and extended properties are stored in separate tables. They each have an `EVENT_ID` that references the `_ID` in the Events table.
`CalendarContract.Instances`	This table holds the start and end time for each occurrence of an event. Each row in this table represents a single event occurrence. For one-time events there is a 1:1 mapping of instances to events. For recurring events, multiple rows are automatically generated that correspond to multiple occurrences of that event.
`CalendarContract.Attendees`	This table holds the event attendee (guest) information. Each row represents a single guest of an event. It specifies the type of guest and the guest's attendance response for the event.
`CalendarContract.Reminders`	This table holds the alert/notification data. Each row represents a single alert for an event. An event can have multiple reminders. The maximum number of reminders per event is specified in `MAX_REMINDERS`, which is set by the sync adapter that owns the given calendar. Reminders are specified in minutes before the event and have a method that determines how the user will be alerted.

Constant	Description
`NAME`	The name of the calendar.
`CALENDAR_DISPLAY_NAME`	The name of this calendar that is displayed to the user.
`VISIBLE`	A boolean indicating whether the calendar is selected to be displayed. A value of 0 indicates that events associated with this calendar should not be shown. A value of 1 indicates that events associated with this calendar should be shown. This value affects the generation of rows in the `CalendarContract.Instances` table.
`SYNC_EVENTS`	A boolean indicating whether the calendar should be synced and have its events stored on the device. A value of 0 says do not sync this calendar or store its events on the device. A value of 1 says sync events for this calendar and store its events on the device.

1. Formulate a query

```
// Projection array. Creating indices for this array instead of doing
// dynamic lookups improves performance.
public static final String[] EVENT_PROJECTION = new String[] {
    Calendars._ID,                           // 0
    Calendars.ACCOUNT_NAME,                  // 1
    Calendars.CALENDAR_DISPLAY_NAME,         // 2
    Calendars.OWNER_ACCOUNT                  // 3
};

// The indices for the projection array above.
private static final int PROJECTION_ID_INDEX = 0;
private static final int PROJECTION_ACCOUNT_NAME_INDEX = 1;
private static final int PROJECTION_DISPLAY_NAME_INDEX = 2;
private static final int PROJECTION_OWNER_ACCOUNT_INDEX = 3;
```

2. Use this query with a ContentResolver to retrieve a Cursor

```
// Run query
Cursor cur = null;
ContentResolver cr = getContentResolver();
Uri uri = Calendars.CONTENT_URI;
String selection = "((" + Calendars.ACCOUNT_NAME + " = ?) AND ("
                        + Calendars.ACCOUNT_TYPE + " = ?) AND ("
                        + Calendars.OWNER_ACCOUNT + " = ?))";
String[] selectionArgs = new String[] {"sampleuser@gmail.com", "com.google",
        "sampleuser@gmail.com"};
// Submit the query and get a Cursor object back.
cur = cr.query(uri, EVENT_PROJECTION, selection, selectionArgs, null);
```

FURTHER DETAILS

See:

Content Providers overview:

https://developer.android.com/guide/topics/providers/content-provider-basics.html

More detail on how to design good Content Providers, e.g. how to structure your data:

https://developer.android.com/guide/topics/providers/content-provider-creating.html

PRACTICAL

Task 1: Download and run the content provider demo

Task 2: Complete the tutorial at this link:

https://developer.android.com/training/contacts-provider/retrieve-names.html

It focuses on accessing the inbuilt contacts list.

Task 3: Complete the second part of the weather app tutorial

Any questions around coursework, speak to me.

QUESTIONS?

Contact:

d.coyle@ucd.ie

Please ask in the Discussion Forum.

Next class:

Data Storage