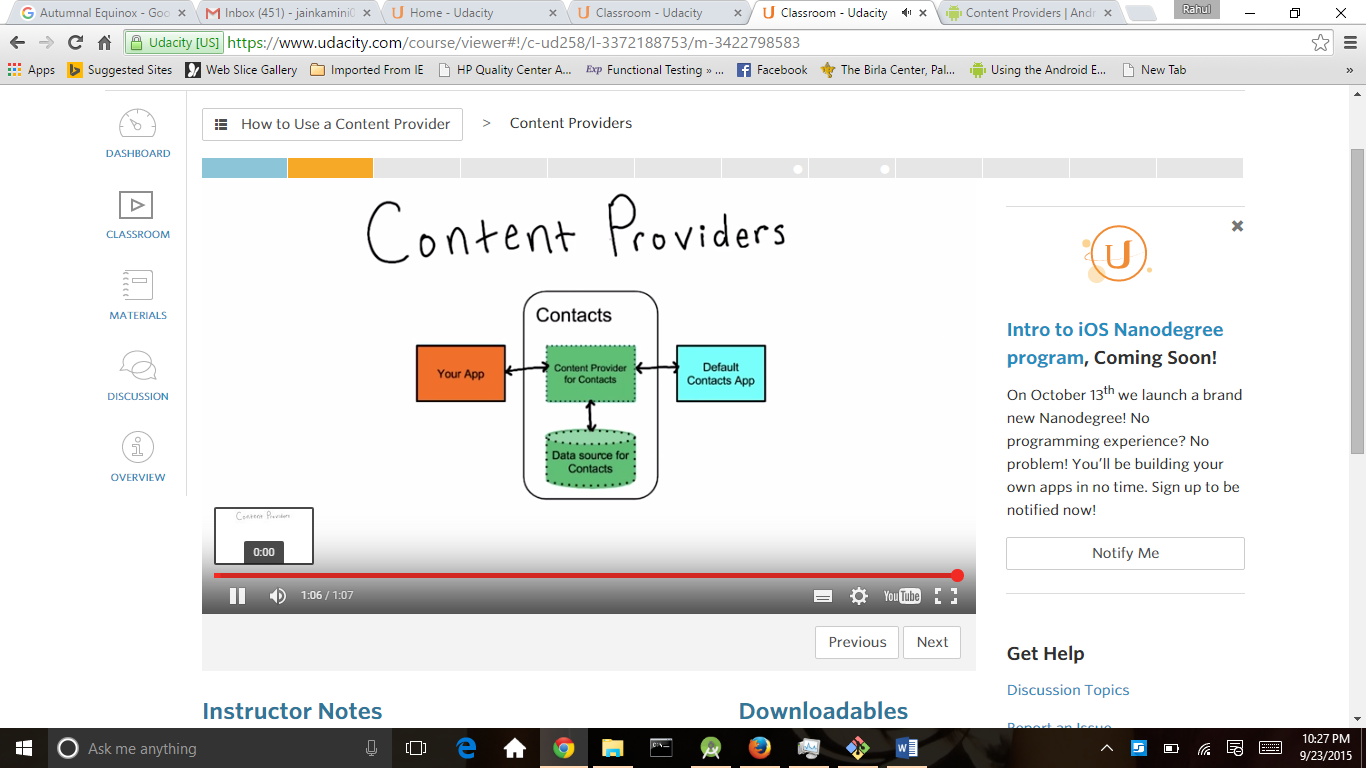
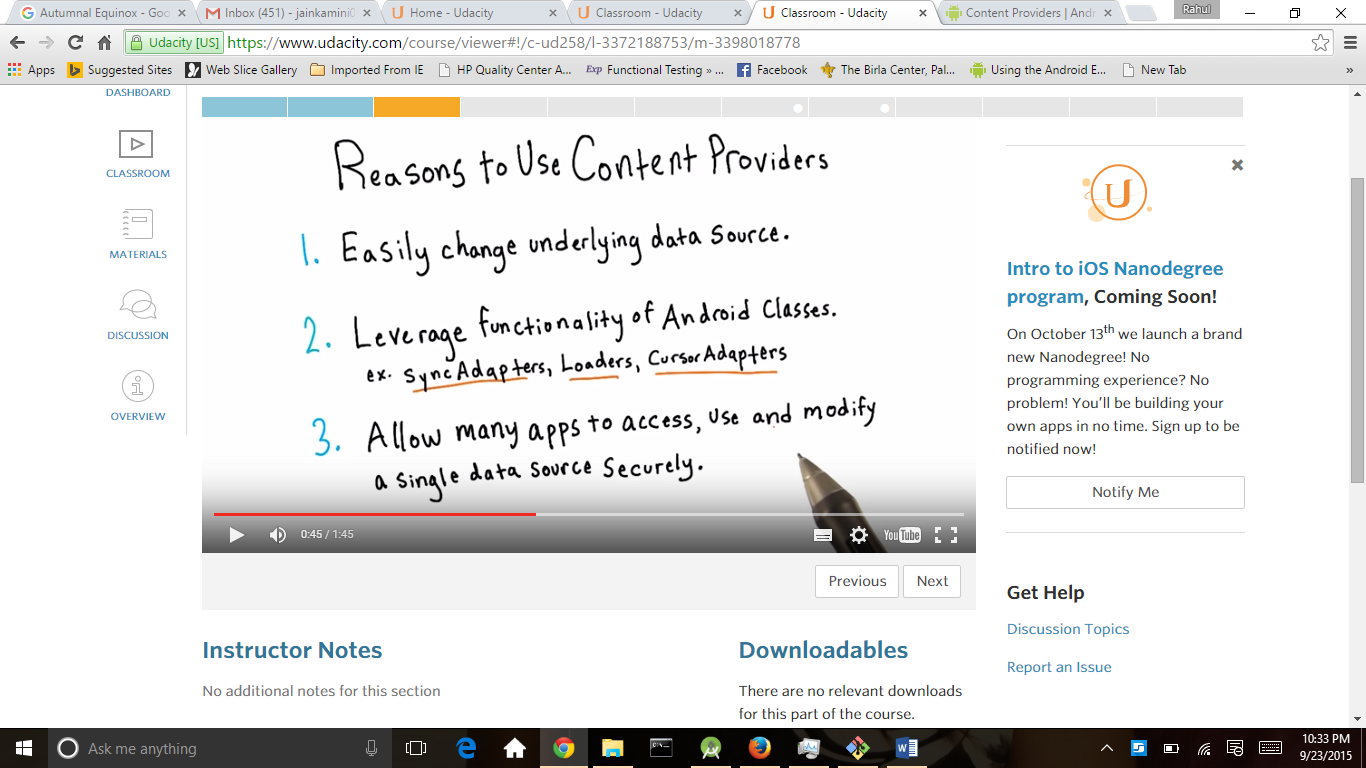
**Content Provider**

[**http://developer.android.com/guide/topics/providers/content-providers.html**](http://developer.android.com/guide/topics/providers/content-providers.html)

**https://github.com/udacity/DictionaryProviderExample/commits/Setting\_up\_DictionaryProviderExample**







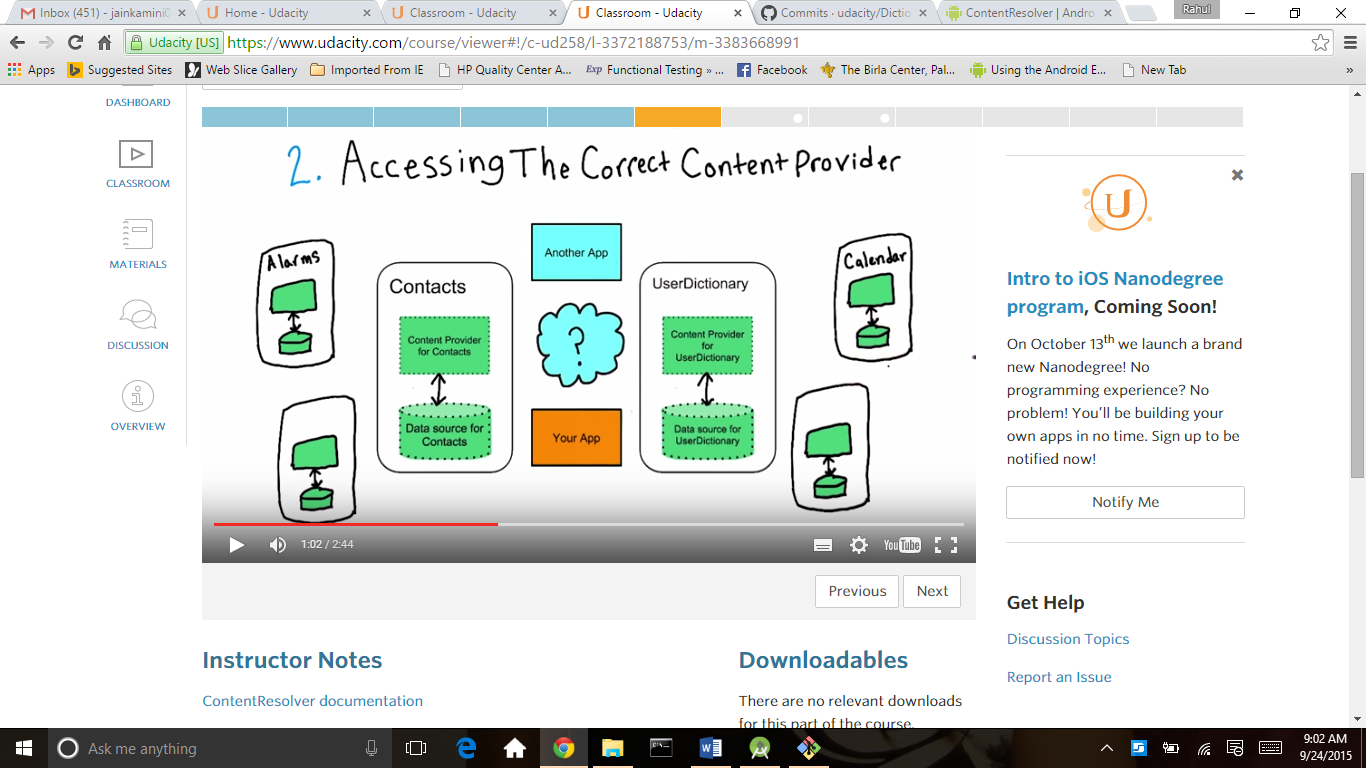
For content provider we need permission so we add permission in AndroidManifest.xml

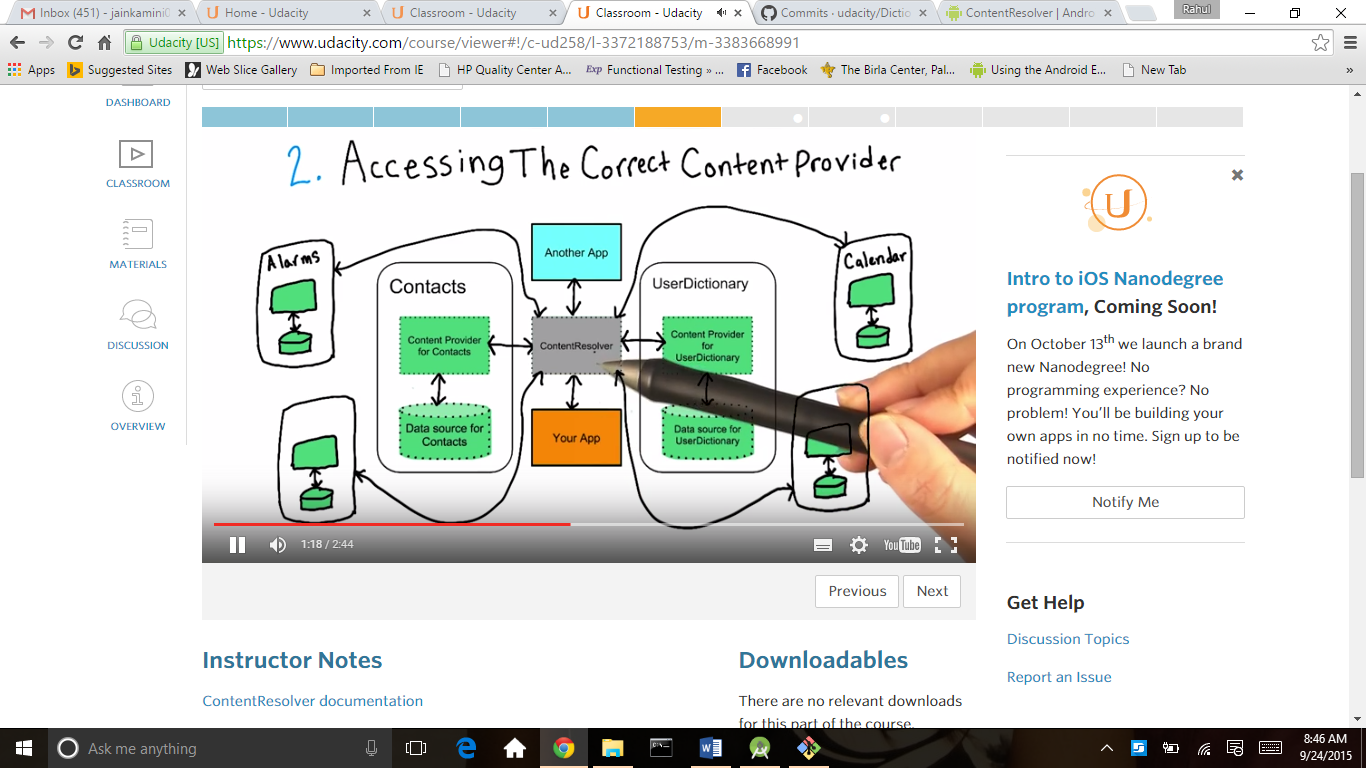


ContentResolver :

<http://developer.android.com/reference/android/content/ContentResolver.html>

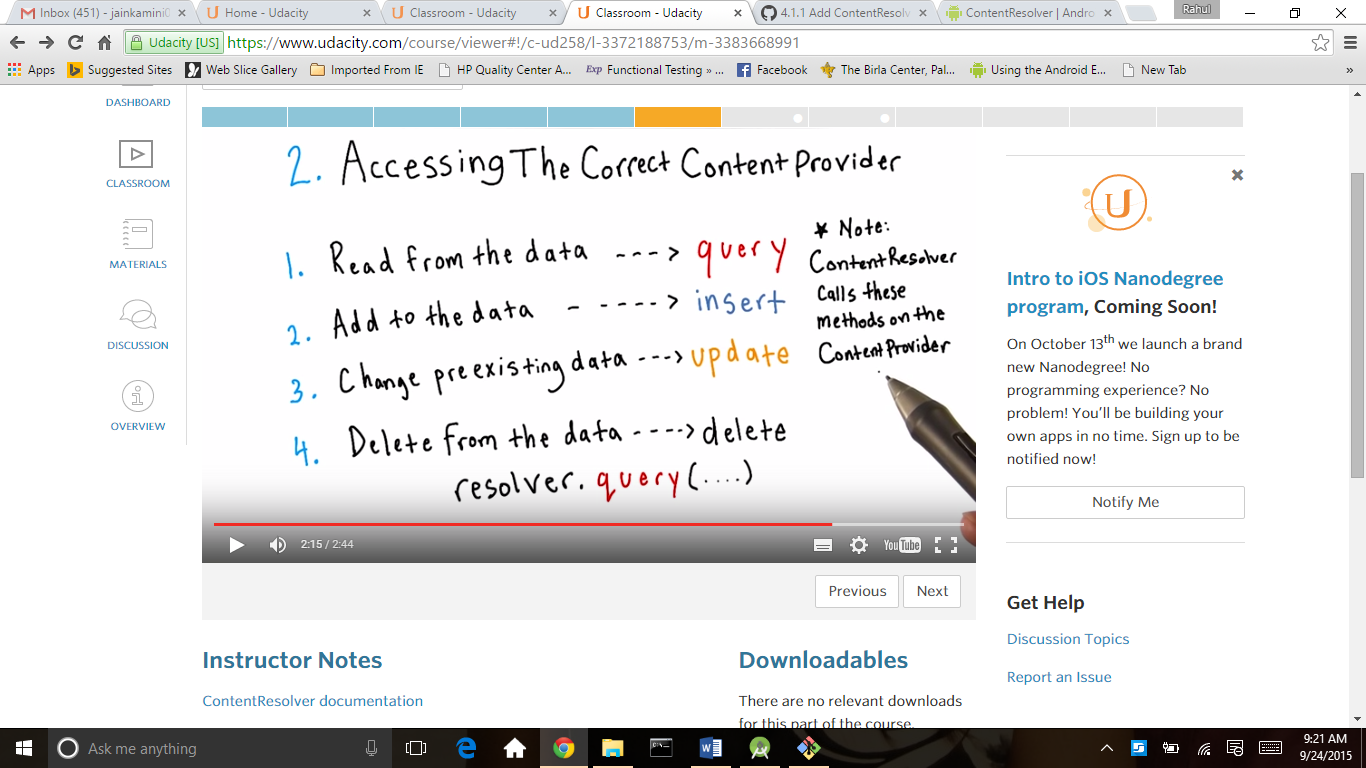
There are lot of contentproviders like contacts,calander,alarms and handful some standards ContentProviders. also your app is not only app on the system because all the app is communicate with different content provider so the contentresolver is provide correct content provider to your app it is the job of content resolver





This is code we use for Content resolver

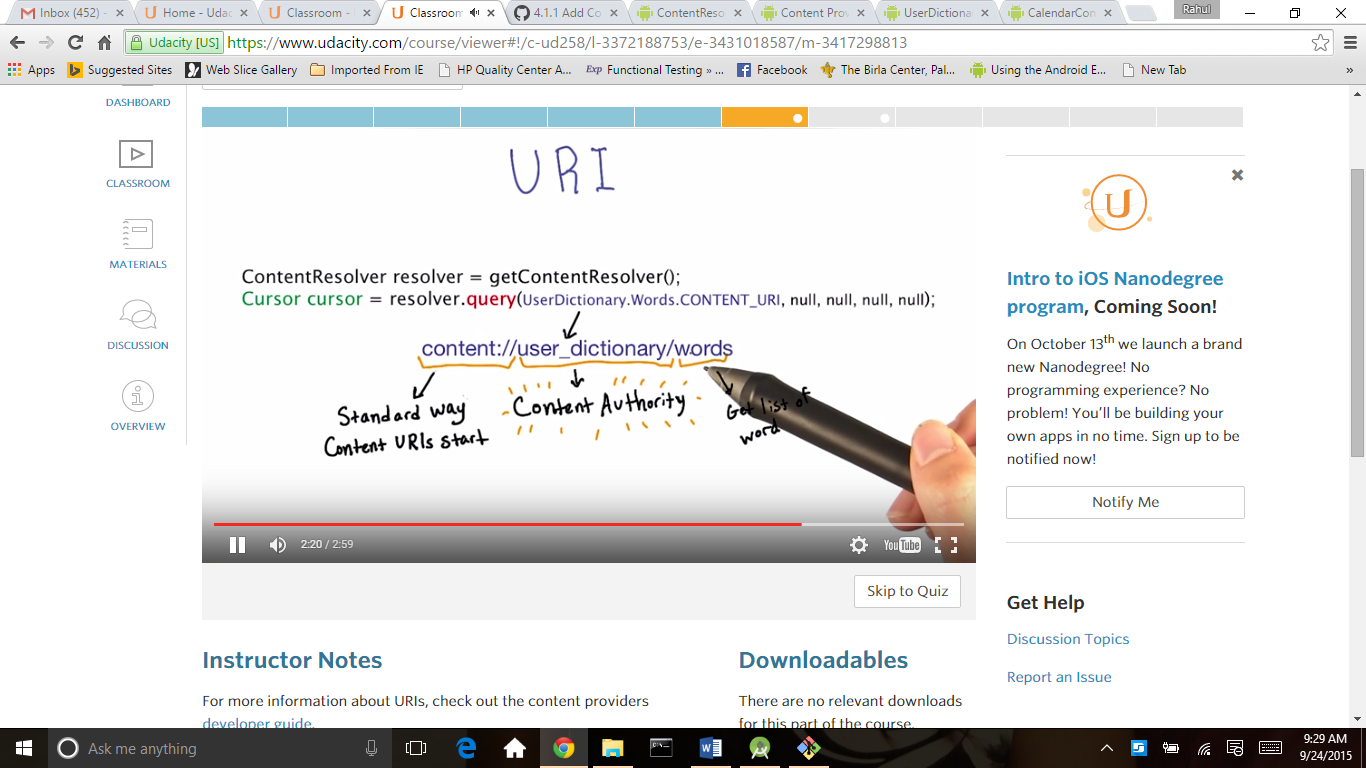
ContentResolver resolver = getContentResolver();

Cursor cursor = resolver.query(UserDictionary.Words.CONTENT\_URI, null, null, null, null); 

<https://stuff.mit.edu/afs/sipb/project/android/sdk/android-sdk-linux/docs//guide/topics/providers/content-provider-basics.html#ContentURIs>

<http://developer.android.com/reference/android/provider/UserDictionary.Words.html>

<http://developer.android.com/reference/android/provider/CalendarContract.html>

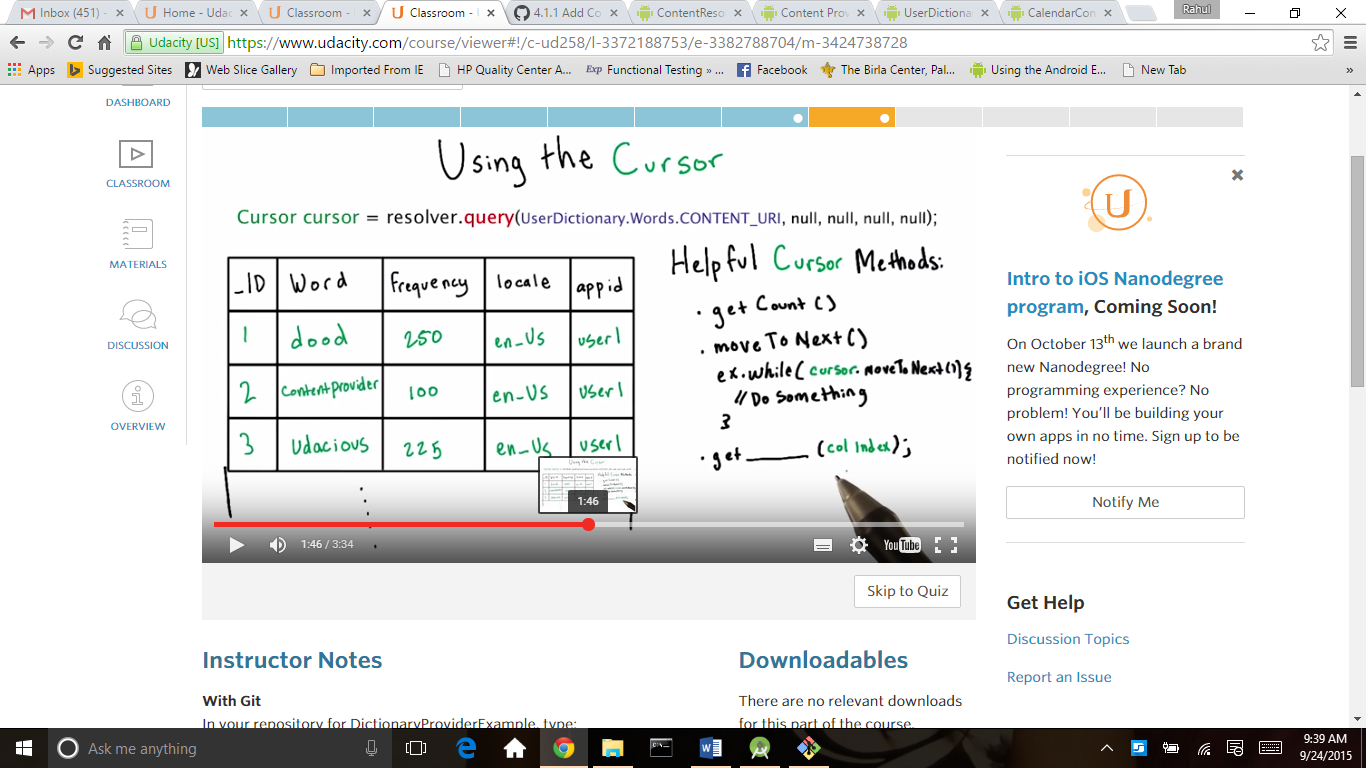


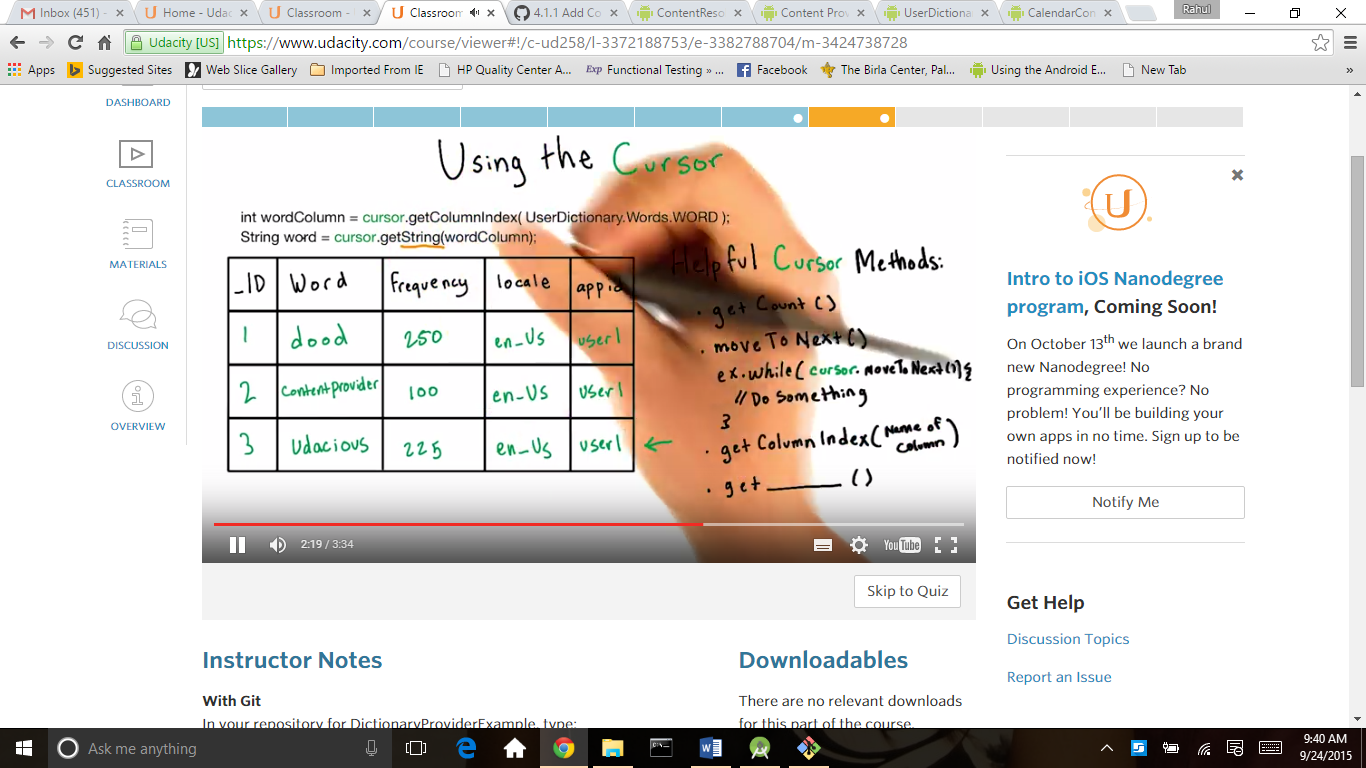
Cursor

http://www.tutorialspoint.com/java/java\_using\_iterator.htm

http://developer.android.com/reference/android/database/Cursor.html







Cursor is use for fetch data and here for dictionary we use face word frequency,columned, words (green code) always close the cursor because release the memory .the out put of this code is in below screen

package android.example.com.dictionaryproviderexample;

import android.content.ContentResolver;

import android.database.Cursor;

import android.os.Bundle;

import android.provider.UserDictionary;

import android.provider.UserDictionary.Words;

import android.support.v7.app.ActionBarActivity;

import android.widget.TextView;

/\*\*

\* This is the central activity for the Provider Dictionary Example App. The purpose of this app is

\* to show an example of accessing the {@link Words} list via its' Content Provider.

\*/

public class MainActivity extends ActionBarActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Get the TextView which will be populated with the Dictionary ContentProvider data.

TextView dictTextView = (TextView) findViewById(R.id.dictionary\_text\_view);

// Get the ContentResolver which will send a message to the ContentProvider

ContentResolver resolver = getContentResolver();

// Get a Cursor containing all of the rows in the Words table

Cursor cursor = resolver.query(UserDictionary.Words.CONTENT\_URI, null, null, null, null);

// Surround the cursor in a try statement so that the finally block will eventually execute

try {

dictTextView.setText("The UserDictionary contains " + cursor.getCount() + " words\n");

dictTextView.append("COLUMNS: " + Words.\_ID + " - " + Words.FREQUENCY +

" - " + Words.WORD);

// Get the index of the columns using UserDictionary.Words constants,

// which contain the headers of the columns.

int idColumn = cursor.getColumnIndex(UserDictionary.Words.\_ID);

int frequencyColumn = cursor.getColumnIndex(UserDictionary.Words.FREQUENCY);

int wordColumn = cursor.getColumnIndex(UserDictionary.Words.WORD);

// Iterates through all returned rows in the cursor.

while (cursor.moveToNext()) {

// Use that index to extract the String value of the word

// at the current row the cursor is on.

int id = cursor.getInt(idColumn);

int frequency = cursor.getInt(frequencyColumn);

String word = cursor.getString(wordColumn);

dictTextView.append(("\n" + id + " - " + frequency + " - " + word));

}

} finally {

// Always close your cursor to avoid memory leaks

cursor.close();

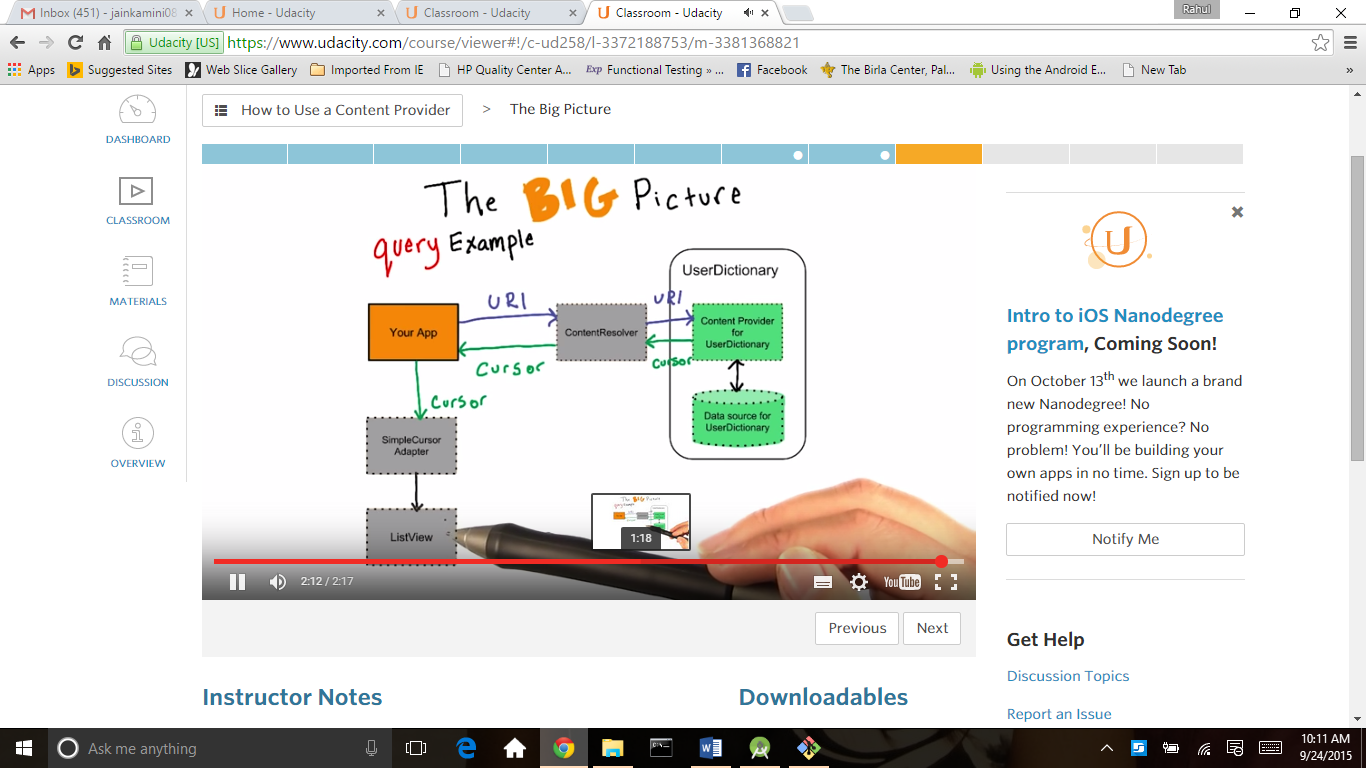
}

}

}

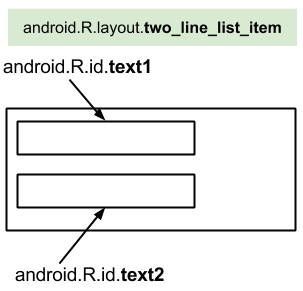
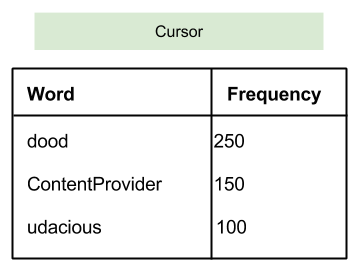
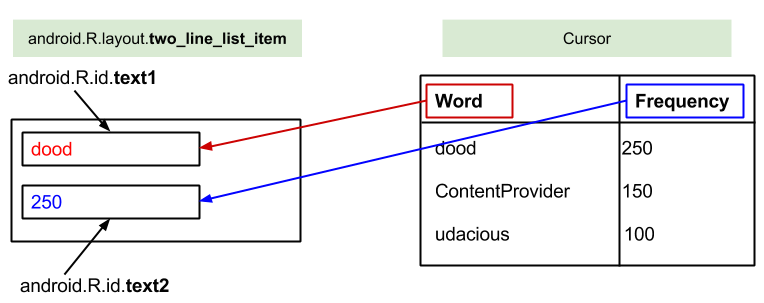






SimpleCursorAdapter

<http://developer.android.com/reference/android/widget/SimpleCursorAdapter.html>

1. What layout xml each list item will use. You could make your own xml list item layout, but for simplicity use android.R.layout.two\_line\_list\_item which is a built-in layout.
2. The Cursor.
3. The columns in the Cursor to use (their String headings), stored in a String array.
4. The id’s of text views contained in the layout that you passed as the first parameter, stored as an int array. These text views will be filled from the values of the columns specified in the third parameter.
5. An optional settings flag. Just set it to 0 for now.
6. Let’s look at how the first layout.xml parameter and the two arrays work together:
7. 
8. This is two\_line\_list\_item.xml, it has two TextViews, text1 and text2.
9. 
10. Over here, we have the cursor, which has two columns we care about: the word column and the frequency column. Remember we can get the headings for each of these columns from the UserDictionary.Words class.
11. 
12. The ordering of the String array with headings and the int array with text view id’s determines what data in the cursor is put in what TextView in the layout. So the data in the first column listed in the String array is put into the first TextView in the id array and so on.

### Step 3 : Set the ListView’s Adapter to the SimpleCursorAdapter

1. Don’t forget to do this! If you don’t know how, check out the [documentation about ListViews](http://developer.android.com/reference/android/widget/ListView.html) or this Android documentation about [Building Layouts with an Adapter](http://developer.android.com/guide/topics/ui/declaring-layout.html#AdapterViews).
2. When you're finished click **Next** to see my solution.

(Green code) is for simplecursoradapter

In this code first we want to bind listviwe and 2 textview first text for word and second text for COLUMNS\_TO\_BE\_BOUND is string array which is for word and frequency

and LAYOUT\_ITEMS\_TO\_FILL is int array for textview both are pass in simple array adapter

out put of this is shown in below screen

/\*

\* Copyright (C) 2014 The Android Open Source Project

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* http://www.apache.org/licenses/LICENSE-2.0

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package android.example.com.dictionaryproviderexample;

import android.content.ContentResolver;

import android.database.Cursor;

import android.os.Bundle;

import android.provider.UserDictionary;

import android.provider.UserDictionary.Words;

import android.support.v7.app.ActionBarActivity;

import android.widget.ListView;

import android.support.v4.widget.SimpleCursorAdapter;

/\*\*

\* This is the central activity for the Provider Dictionary Example App. The purpose of this app is

\* to show an example of accessing the {@link Words} list via its' Content Provider.

\*/

public class MainActivity extends ActionBarActivity {

// For the SimpleCursorAdapter to match the UserDictionary columns to layout items.

private static final String[] COLUMNS\_TO\_BE\_BOUND = new String[] {

UserDictionary.Words.WORD,

UserDictionary.Words.FREQUENCY

};

private static final int[] LAYOUT\_ITEMS\_TO\_FILL = new int[] {

android.R.id.text1,

android.R.id.text2

};

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

// Get the TextView which will be populated with the Dictionary ContentProvider data.

ListView dictListView = (ListView) findViewById(R.id.dictionary\_list\_view);

// Get the ContentResolver which will send a message to the ContentProvider.

ContentResolver resolver = getContentResolver();

// Get a Cursor containing all of the rows in the Words table.

Cursor cursor = resolver.query(UserDictionary.Words.CONTENT\_URI, null, null, null, null);

// Set the Adapter to fill the standard two\_line\_list\_item layout with data from the Cursor.

SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,

android.R.layout.two\_line\_list\_item,

cursor,

COLUMNS\_TO\_BE\_BOUND,

LAYOUT\_ITEMS\_TO\_FILL,

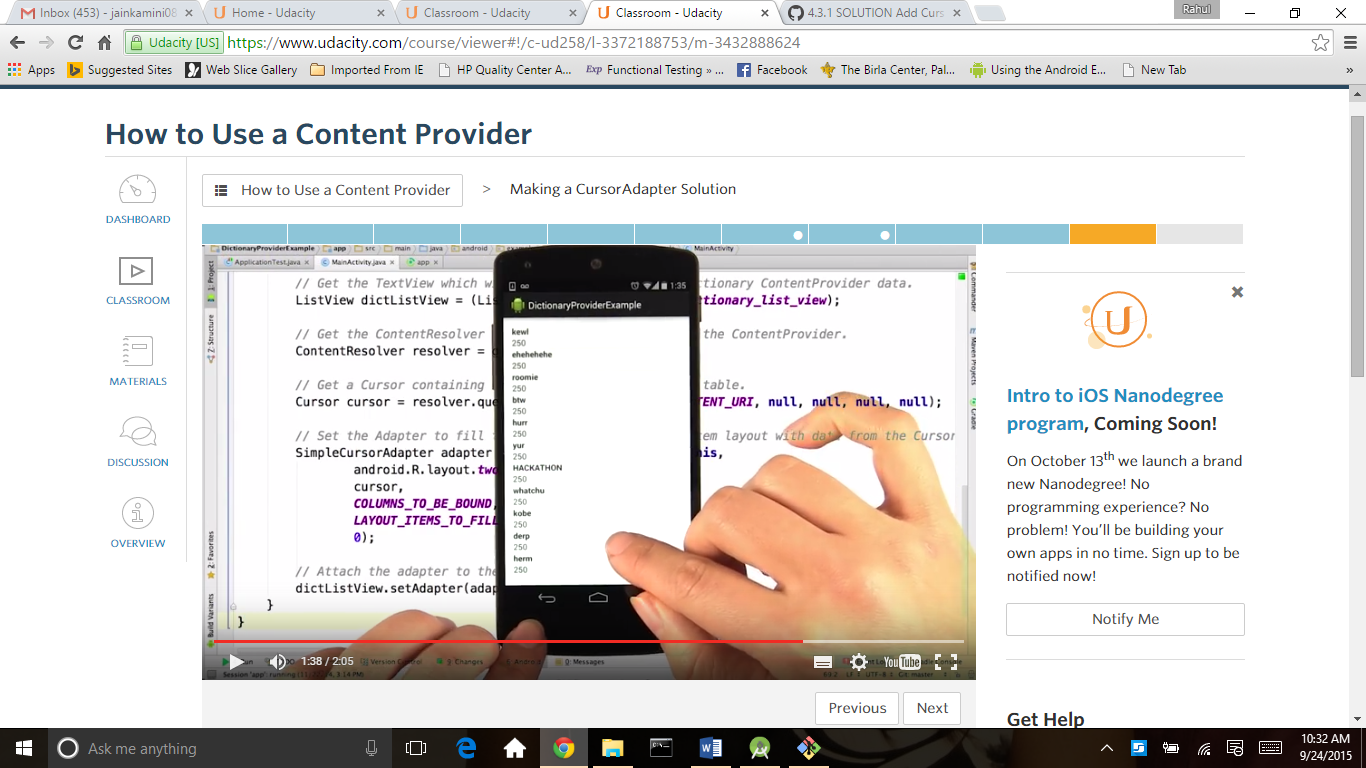
0);

// Attach the adapter to the ListView.

dictListView.setAdapter(adapter);

}

}

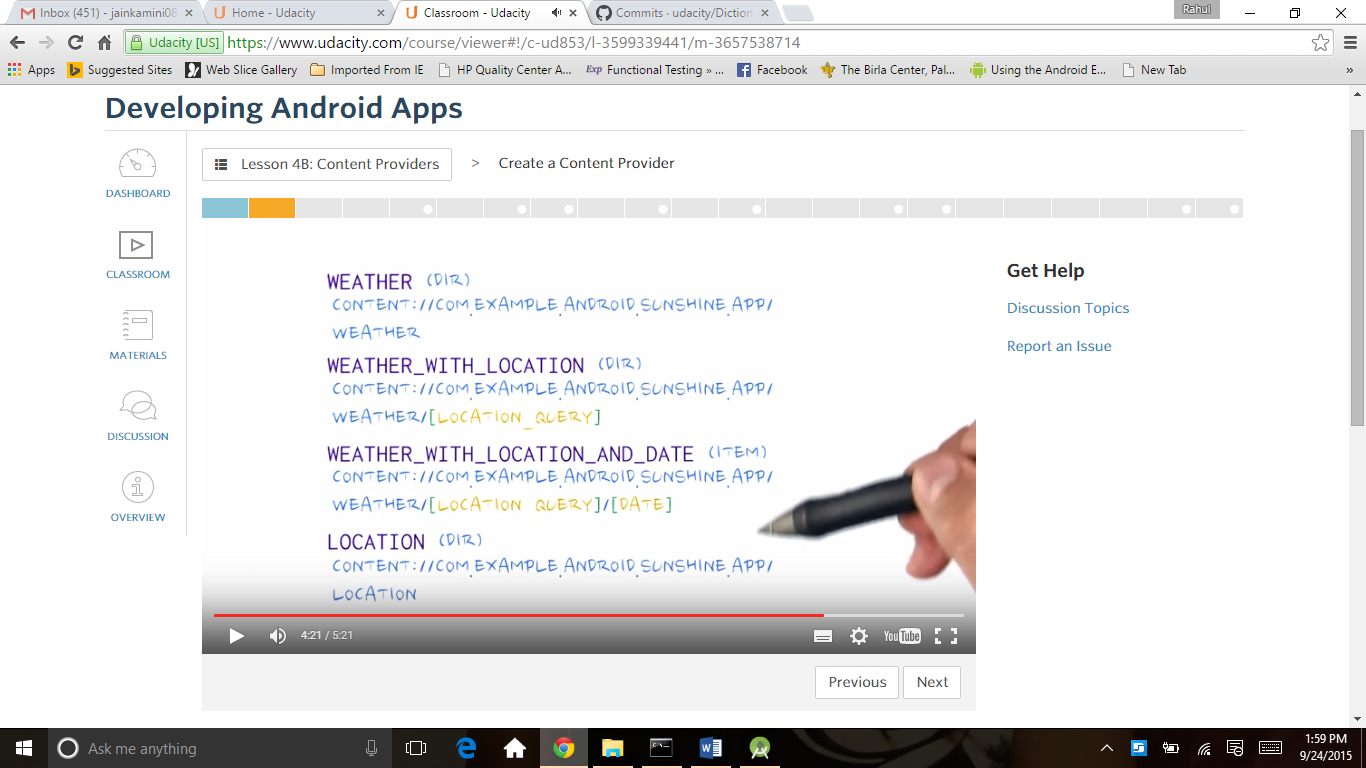


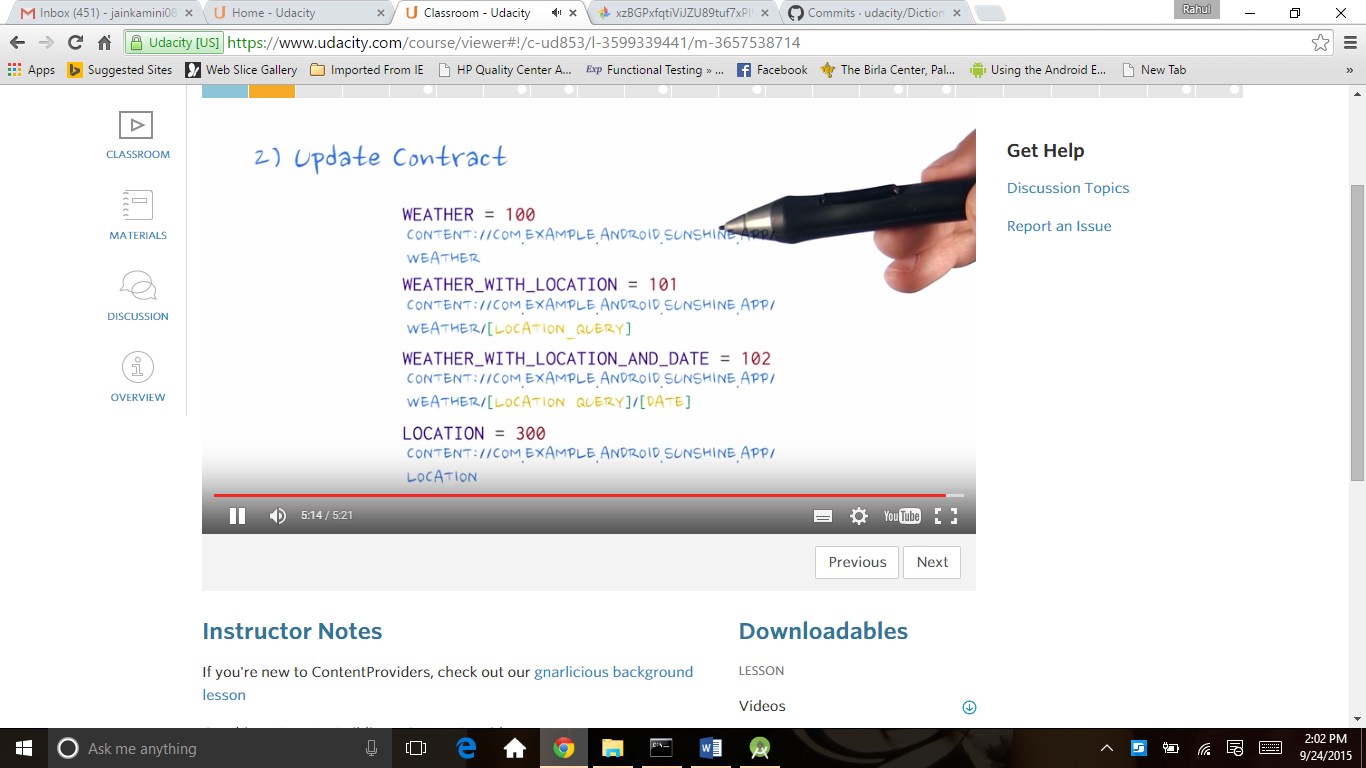
Content provider for sunshine app

https://www.udacity.com/course/viewer#!/c-ud853/l-3599339441/m-3657538714

<http://lh4.ggpht.com/xzBGPxfqtiViJZU89tuf7xPI9Xp7psIzg5bkGL2PPNmyxBjVYCSsCBQIeG20-OygFbd-1RmgO8iseVYB-Ae0=s0#w=1216&h=641>

<http://developer.android.com/reference/android/net/Uri.Builder.html>





First building a content provider with uri

For this see above screen for uri we build uri in below code

*// The "Content authority" is a name for the entire content provider, similar to the  
// relationship between a domain name and its website. A convenient string to use for the  
// content authority is the package name for the app, which is guaranteed to be unique on the  
// device.*

**public static final** String ***CONTENT\_AUTHORITY*** = **"com.example.android.sunshine.app"**;

*// Use CONTENT\_AUTHORITY to create the base of all URI's which apps will use to contact  
// the content provider.***public static final** Uri ***BASE\_CONTENT\_URI*** = Uri.*parse*(**"content://"** + ***CONTENT\_AUTHORITY***);  
  
*// Possible paths (appended to base content URI for possible URI's)  
// For instance, content://com.example.android.sunshine.app/weather/ is a valid path for  
// looking at weather data. content://com.example.android.sunshine.app/givemeroot/ will fail,  
// as the ContentProvider hasn't been given any information on what to do with "givemeroot".  
// At least, let's hope not. Don't be that dev, reader. Don't be that dev.*

*//this is the two tables weather and location***public static final** String ***PATH\_WEATHER*** = **"weather"**;  
**public static final** String ***PATH\_LOCATION*** = **"location"**;

**public static final** Uri ***CONTENT\_URI*** =  
 ***BASE\_CONTENT\_URI***.buildUpon().appendPath(***PATH\_WEATHER***).build();

//this is a code where uri is encoding

**public static** Uri buildWeatherUri(**long** id) {  
 **return** ContentUris.*withAppendedId*(***CONTENT\_URI***, id);

}

//build this with query parameters

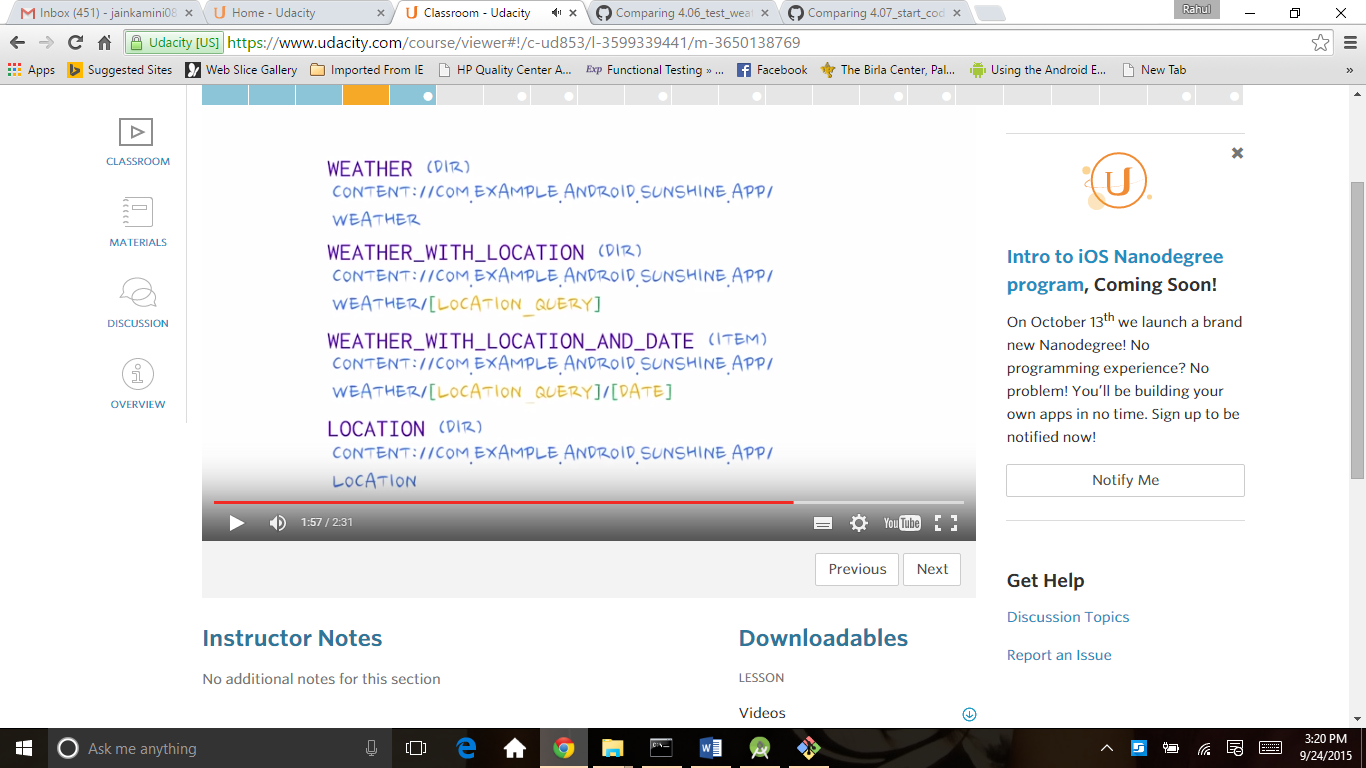
**public static** Uri buildWeatherLocationWithStartDate(  
 String locationSetting, **long** startDate) {  
 **long** normalizedDate = *normalizeDate*(startDate);  
 **return *CONTENT\_URI***.buildUpon().appendPath(locationSetting)  
 .appendQueryParameter(***COLUMN\_DATE***, Long.*toString*(normalizedDate)).build();  
}

it is a cursordir base type or cursoritem type

**public static final** String ***CONTENT\_TYPE*** =  
 ContentResolver.***CURSOR\_DIR\_BASE\_TYPE*** + **"/"** + ***CONTENT\_AUTHORITY*** + **"/"** + ***PATH\_WEATHER***;  
**public static final** String ***CONTENT\_ITEM\_TYPE*** =  
 ContentResolver.***CURSOR\_ITEM\_BASE\_TYPE*** + **"/"** + ***CONTENT\_AUTHORITY*** + **"/"** + ***PATH\_WEATHER***;

Code for this in below

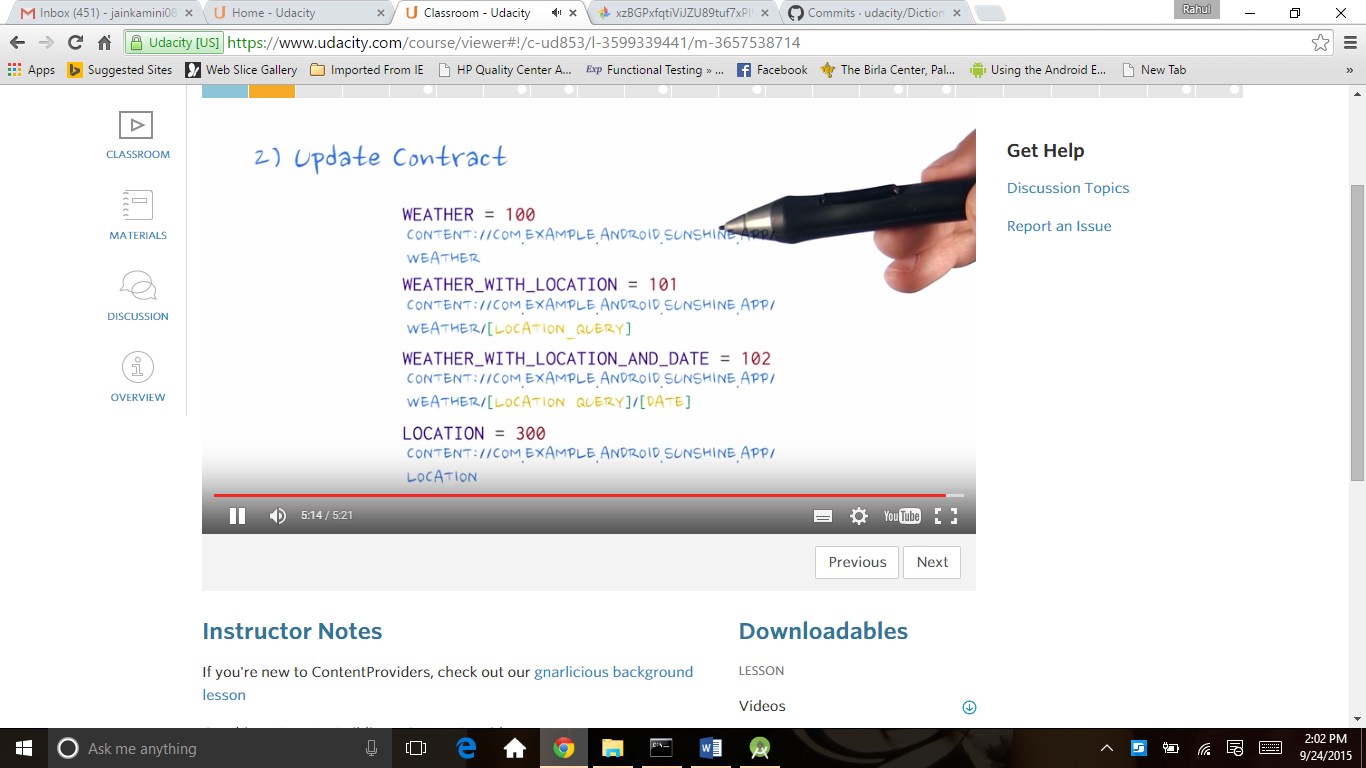
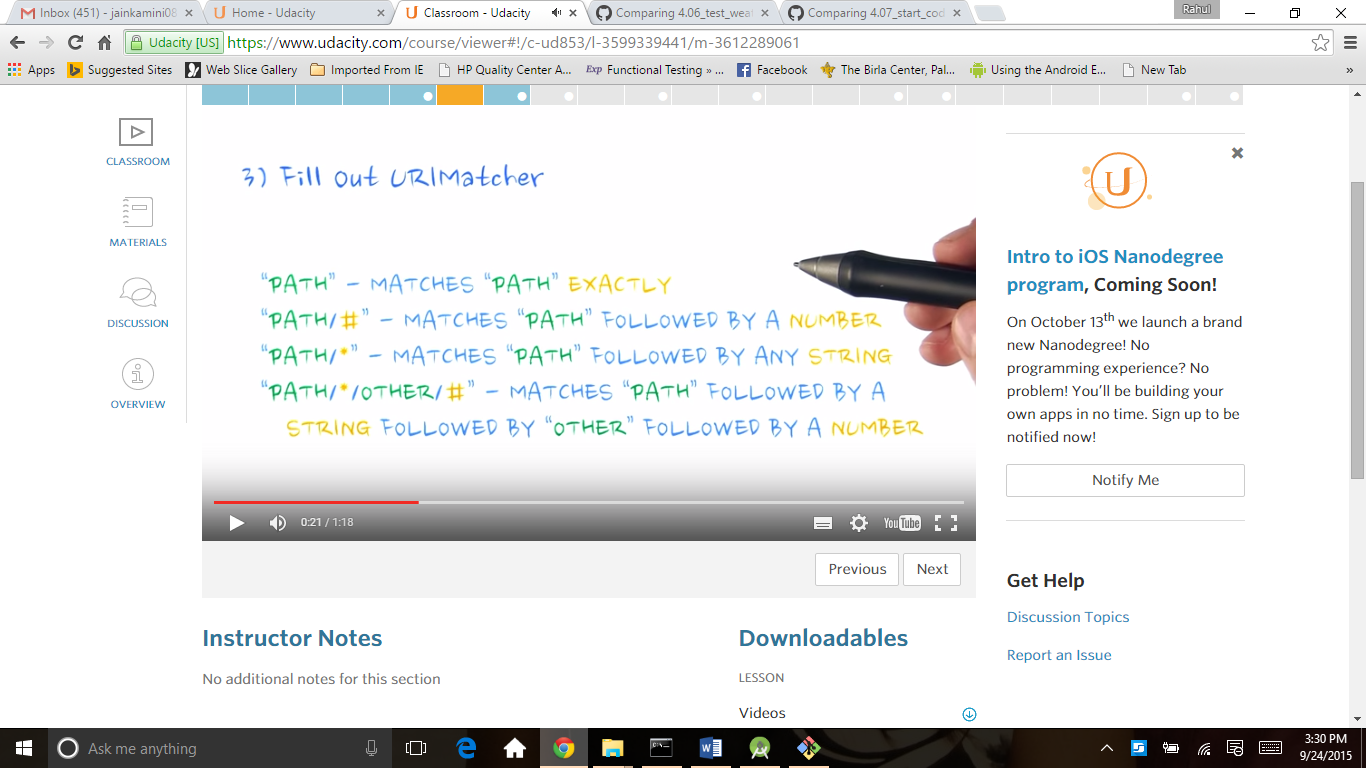
uri Screen

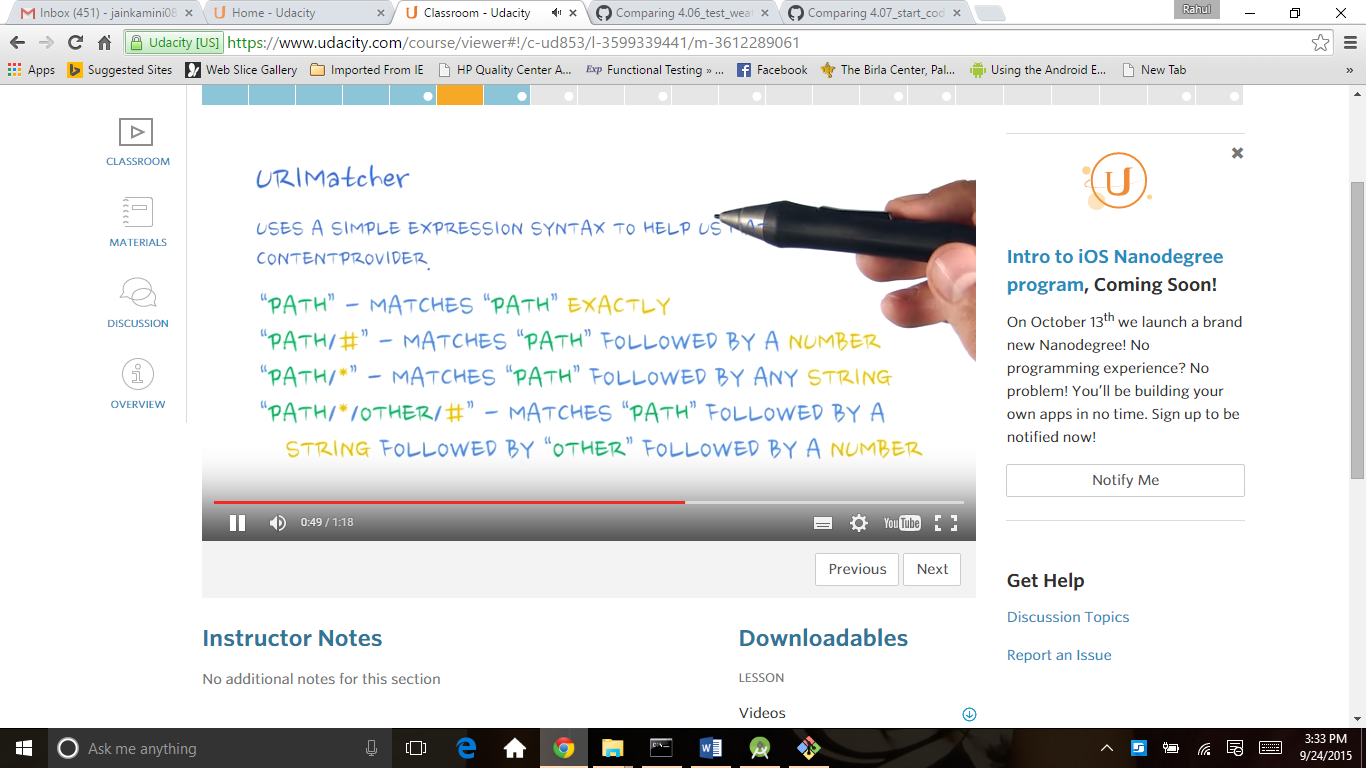


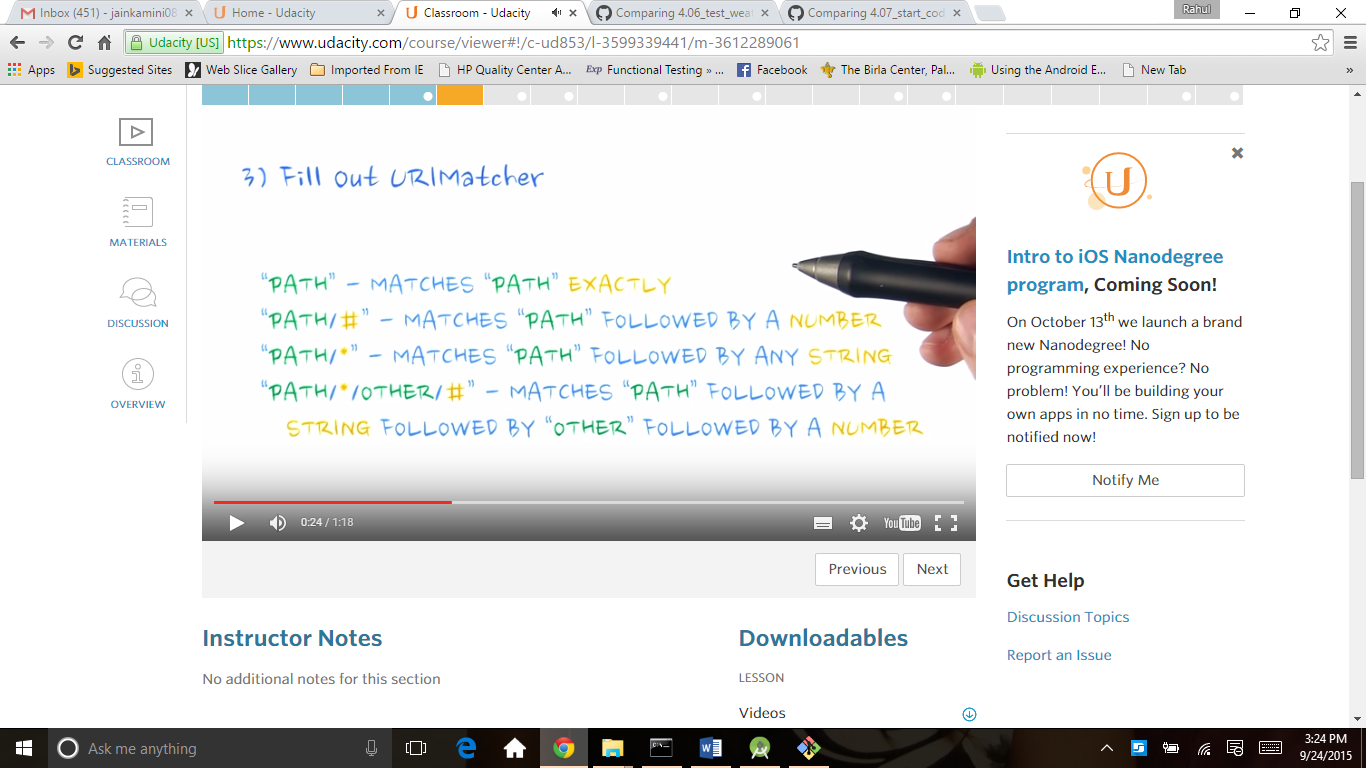
*/\*  
 \* Copyright (C) 2014 The Android Open Source Project  
 \*  
 \* Licensed under the Apache License, Version 2.0 (the "License");  
 \* you may not use this file except in compliance with the License.  
 \* You may obtain a copy of the License at  
 \*  
 \* http://www.apache.org/licenses/LICENSE-2.0  
 \*  
 \* Unless required by applicable law or agreed to in writing, software  
 \* distributed under the License is distributed on an "AS IS" BASIS,  
 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
 \* See the License for the specific language governing permissions and  
 \* limitations under the License.  
 \*/***package** com.example.android.sunshine.app.data;  
  
**import** android.content.ContentResolver;  
**import** android.content.ContentUris;  
**import** android.net.Uri;  
**import** android.provider.BaseColumns;  
**import** android.text.format.Time;  
  
*/\*\*  
 \* Defines table and column names for the weather database.  
 \*/***public class** WeatherContract {  
  
 *// The "Content authority" is a name for the entire content provider, similar to the  
 // relationship between a domain name and its website. A convenient string to use for the  
 // content authority is the package name for the app, which is guaranteed to be unique on the  
 // device.* **public static final** String ***CONTENT\_AUTHORITY*** = **"com.example.android.sunshine.app"**;  
  
 *// Use CONTENT\_AUTHORITY to create the base of all URI's which apps will use to contact  
 // the content provider.* **public static final** Uri ***BASE\_CONTENT\_URI*** = Uri.*parse*(**"content://"** + ***CONTENT\_AUTHORITY***);  
  
 *// Possible paths (appended to base content URI for possible URI's)  
 // For instance, content://com.example.android.sunshine.app/weather/ is a valid path for  
 // looking at weather data. content://com.example.android.sunshine.app/givemeroot/ will fail,  
 // as the ContentProvider hasn't been given any information on what to do with "givemeroot".  
 // At least, let's hope not. Don't be that dev, reader. Don't be that dev.* **public static final** String ***PATH\_WEATHER*** = **"weather"**;  
 **public static final** String ***PATH\_LOCATION*** = **"location"**;  
  
 *// To make it easy to query for the exact date, we normalize all dates that go into  
 // the database to the start of the the Julian day at UTC.* **public static long** normalizeDate(**long** startDate) {  
 *// normalize the start date to the beginning of the (UTC) day* Time time = **new** Time();  
 time.set(startDate);  
 **int** julianDay = Time.*getJulianDay*(startDate, time.**gmtoff**);  
 **return** time.setJulianDay(julianDay);  
 }  
  
 */\* Inner class that defines the table contents of the location table \*/* **public static final class** LocationEntry **implements** BaseColumns {  
  
 **public static final** Uri ***CONTENT\_URI*** =  
 ***BASE\_CONTENT\_URI***.buildUpon().appendPath(***PATH\_LOCATION***).build();  
  
 **public static final** String ***CONTENT\_TYPE*** =  
 ContentResolver.***CURSOR\_DIR\_BASE\_TYPE*** + **"/"** + ***CONTENT\_AUTHORITY*** + **"/"** + ***PATH\_LOCATION***;  
 **public static final** String ***CONTENT\_ITEM\_TYPE*** =  
 ContentResolver.***CURSOR\_ITEM\_BASE\_TYPE*** + **"/"** + ***CONTENT\_AUTHORITY*** + **"/"** + ***PATH\_LOCATION***;  
  
 *// Table name* **public static final** String ***TABLE\_NAME*** = **"location"**;  
  
 *// The location setting string is what will be sent to openweathermap  
 // as the location query.* **public static final** String ***COLUMN\_LOCATION\_SETTING*** = **"location\_setting"**;  
  
 *// Human readable location string, provided by the API. Because for styling,  
 // "Mountain View" is more recognizable than 94043.* **public static final** String ***COLUMN\_CITY\_NAME*** = **"city\_name"**;  
  
 *// In order to uniquely pinpoint the location on the map when we launch the  
 // map intent, we store the latitude and longitude as returned by openweathermap.* **public static final** String ***COLUMN\_COORD\_LAT*** = **"coord\_lat"**;  
 **public static final** String ***COLUMN\_COORD\_LONG*** = **"coord\_long"**;  
  
 **public static** Uri buildLocationUri(**long** id) {  
 **return** ContentUris.*withAppendedId*(***CONTENT\_URI***, id);  
 }  
 }  
  
 */\* Inner class that defines the table contents of the weather table \*/* **public static final class** WeatherEntry **implements** BaseColumns {  
  
 **public static final** Uri ***CONTENT\_URI*** =  
 ***BASE\_CONTENT\_URI***.buildUpon().appendPath(***PATH\_WEATHER***).build();  
  
 **public static final** String ***CONTENT\_TYPE*** =  
 ContentResolver.***CURSOR\_DIR\_BASE\_TYPE*** + **"/"** + ***CONTENT\_AUTHORITY*** + **"/"** + ***PATH\_WEATHER***;  
 **public static final** String ***CONTENT\_ITEM\_TYPE*** =  
 ContentResolver.***CURSOR\_ITEM\_BASE\_TYPE*** + **"/"** + ***CONTENT\_AUTHORITY*** + **"/"** + ***PATH\_WEATHER***;  
  
 **public static final** String ***TABLE\_NAME*** = **"weather"**;  
  
 *// Column with the foreign key into the location table.* **public static final** String ***COLUMN\_LOC\_KEY*** = **"location\_id"**;  
 *// Date, stored as long in milliseconds since the epoch* **public static final** String ***COLUMN\_DATE*** = **"date"**;  
 *// Weather id as returned by API, to identify the icon to be used* **public static final** String ***COLUMN\_WEATHER\_ID*** = **"weather\_id"**;  
  
 *// Short description and long description of the weather, as provided by API.  
 // e.g "clear" vs "sky is clear".* **public static final** String ***COLUMN\_SHORT\_DESC*** = **"short\_desc"**;  
  
 *// Min and max temperatures for the day (stored as floats)* **public static final** String ***COLUMN\_MIN\_TEMP*** = **"min"**;  
 **public static final** String ***COLUMN\_MAX\_TEMP*** = **"max"**;  
  
 *// Humidity is stored as a float representing percentage* **public static final** String ***COLUMN\_HUMIDITY*** = **"humidity"**;  
  
 *// Humidity is stored as a float representing percentage* **public static final** String ***COLUMN\_PRESSURE*** = **"pressure"**;  
  
 *// Windspeed is stored as a float representing windspeed mph* **public static final** String ***COLUMN\_WIND\_SPEED*** = **"wind"**;  
  
 *// Degrees are meteorological degrees (e.g, 0 is north, 180 is south). Stored as floats.* **public static final** String ***COLUMN\_DEGREES*** = **"degrees"**;  
  
 **public static** Uri buildWeatherUri(**long** id) {  
 **return** ContentUris.*withAppendedId*(***CONTENT\_URI***, id);  
 }  
  
 */\*  
 Student: Fill in this buildWeatherLocation function  
 \*/* **public static** Uri buildWeatherLocation(String locationSetting) {  
 **return *CONTENT\_URI***.buildUpon().appendPath(locationSetting).build();  
 }  
  
 **public static** Uri buildWeatherLocationWithStartDate(  
 String locationSetting, **long** startDate) {  
 **long** normalizedDate = *normalizeDate*(startDate);  
 **return *CONTENT\_URI***.buildUpon().appendPath(locationSetting)  
 .appendQueryParameter(***COLUMN\_DATE***, Long.*toString*(normalizedDate)).build();  
 }  
  
 **public static** Uri buildWeatherLocationWithDate(String locationSetting, **long** date) {  
 **return *CONTENT\_URI***.buildUpon().appendPath(locationSetting)  
 .appendPath(Long.*toString*(*normalizeDate*(date))).build();  
 }  
  
 **public static** String getLocationSettingFromUri(Uri uri) {  
 **return** uri.getPathSegments().get(1);  
 }  
  
 **public static long** getDateFromUri(Uri uri) {  
 **return** Long.*parseLong*(uri.getPathSegments().get(2));  
 }  
  
 **public static long** getStartDateFromUri(Uri uri) {  
 String dateString = uri.getQueryParameter(***COLUMN\_DATE***);  
 **if** (**null** != dateString && dateString.length() > 0)  
 **return** Long.*parseLong*(dateString);  
 **else  
 return** 0;  
 }  
 }  
}

Uri matcher this is a third type of our contentprovider

http://developer.android.com/reference/android/content/UriMatcher.html





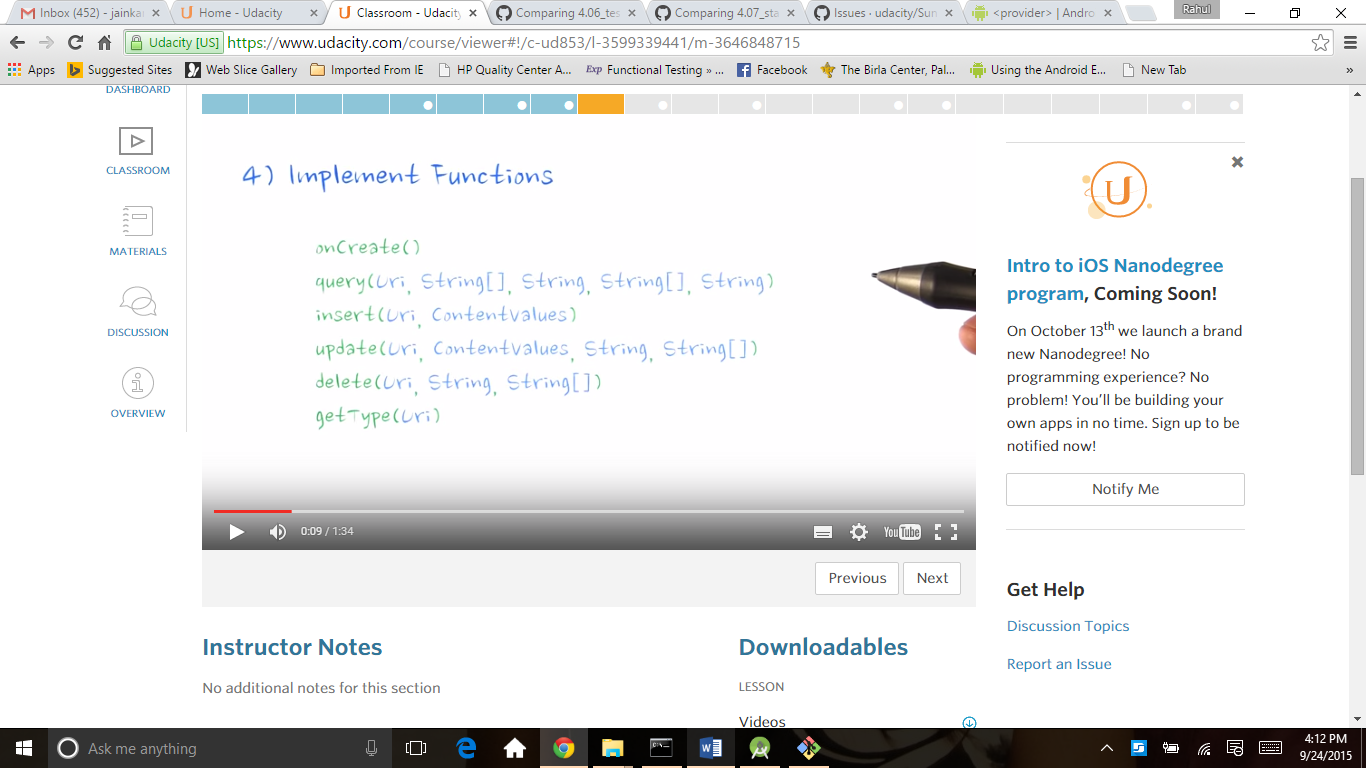
**static** UriMatcher buildUriMatcher() {  
 *// I know what you're thinking. Why create a UriMatcher when you can use regular  
 // expressions instead? Because you're not crazy, that's why.  
  
 // All paths added to the UriMatcher have a corresponding code to return when a match is  
 // found. The code passed into the constructor represents the code to return for the root  
 // URI. It's common to use NO\_MATCH as the code for this case.* **final** UriMatcher matcher = **new** UriMatcher(UriMatcher.***NO\_MATCH***);  
 **final** String authority = WeatherContract.***CONTENT\_AUTHORITY***;  
  
 *// For each type of URI you want to add, create a corresponding code.* matcher.addURI(authority, WeatherContract.***PATH\_WEATHER***, ***WEATHER***);  
 matcher.addURI(authority, WeatherContract.***PATH\_WEATHER*** + **"/\*"**, ***WEATHER\_WITH\_LOCATION***);  
 matcher.addURI(authority, WeatherContract.***PATH\_WEATHER*** + **"/\*/#"**, ***WEATHER\_WITH\_LOCATION\_AND\_DATE***);  
  
 matcher.addURI(authority, WeatherContract.***PATH\_LOCATION***, ***LOCATION***);  
 **return** matcher;  
}

<http://developer.android.com/guide/topics/manifest/provider-element.html>

<http://developer.android.com/guide/topics/providers/content-provider-basics.html#ClientProvider>

we add this code in androidmanifest.xml

<**provider  
 android:authorities="com.example.android.sunshine.app"  
 android:name=".data.WeatherProvider"** />



<http://developer.android.com/reference/android/content/ContentUris.html#parseId(android.net.Uri)>

This method get the uri and return content\_type

**public** String getType(Uri uri) {  
  
 *// Use the Uri Matcher to determine what kind of URI this is.* **final int** match = ***sUriMatcher***.match(uri);  
  
 **switch** (match) {  
 *// Student: Uncomment and fill out these two cases  
// case WEATHER\_WITH\_LOCATION\_AND\_DATE:  
// case WEATHER\_WITH\_LOCATION:* **case *WEATHER\_WITH\_LOCATION\_AND\_DATE***:  
 **return** WeatherContract.WeatherEntry.***CONTENT\_ITEM\_TYPE***;  
 **case *WEATHER\_WITH\_LOCATION***:  
 **return** WeatherContract.WeatherEntry.***CONTENT\_TYPE***;  
 **case *WEATHER***:  
 **return** WeatherContract.WeatherEntry.***CONTENT\_TYPE***;  
 **case *LOCATION***:  
 **return** WeatherContract.LocationEntry.***CONTENT\_TYPE***;  
 **default**:  
 **throw new** UnsupportedOperationException(**"Unknown uri: "** + uri);  
 }  
 }

@Override

**public** Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs,  
 String sortOrder) {  
 *// Here's the switch statement that, given a URI, will determine what kind of request it is,  
 // and query the database accordingly.* Cursor retCursor;  
 **switch** (***sUriMatcher***.match(uri)) {  
 *// "weather/\*/\*"* **case *WEATHER\_WITH\_LOCATION\_AND\_DATE***:  
 {  
 retCursor = getWeatherByLocationSettingAndDate(uri, projection, sortOrder);  
 **break**;  
 }  
 *// "weather/\*"* **case *WEATHER\_WITH\_LOCATION***: {  
 retCursor = getWeatherByLocationSetting(uri, projection, sortOrder);  
 **break**;  
 }  
 *// "weather"* **case *WEATHER***: {  
 retCursor = **null**;  
 **break**;  
 }  
 *// "location"* **case *LOCATION***: {  
 retCursor = **null**;  
 **break**;  
 }  
  
 **default**:  
 **throw new** UnsupportedOperationException(**"Unknown uri: "** + uri);  
 }  
 retCursor.setNotificationUri(getContext().getContentResolver(), uri);  
 **return** retCursor;  
}