**Grow with Google Challenge Scholarship**

**Lesson 10 Notes**

**Which Match - Solution**

Given the following code:

**int** id = 4;

String idString = "" + id;

Uri uri = TaskContract.ItemEntry.CONTENT\_URI;

uri = uri.buildUpon().appendPath(idString).build();

getContentResolver().delete(uri, **null**, **null**);

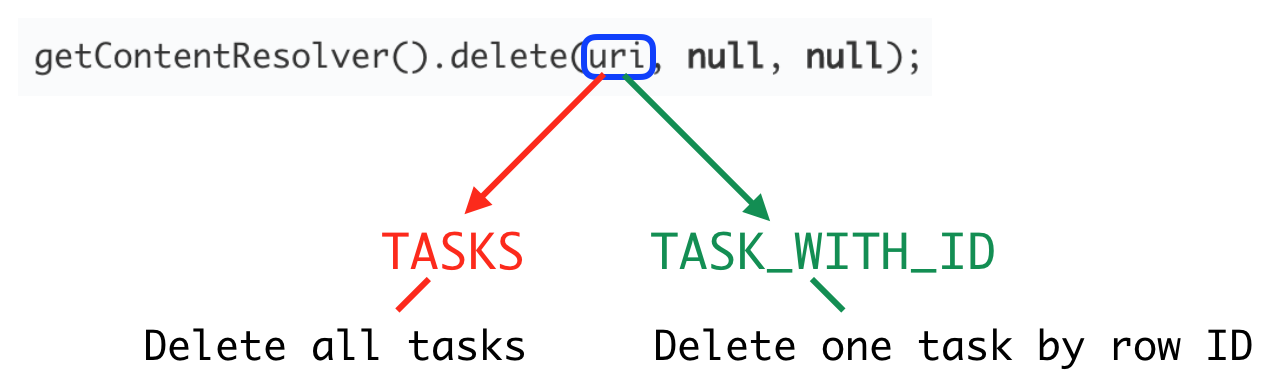
The constructed URI will be: content://com.example.android.todolist/tasks/4. The URIMatcher will recognize this URI as identifying a single row in the tasks database with a row id = 4, so it will match it to the **TASK\_WITH\_ID** integer constant, which is the correct answer.

Without this appended ID, this would match with the **TASKS** constant.

Next, let’s go through the details of the delete function call via a ContentResolver.

getContentResolver().delete(uri, **null**, **null**);

This delete call will get to our TaskContentProvider and then the UriMatcher will match the passed in URI to the TASK\_WITH\_ID integer constant. This is important for the delete method because the ContentProvider needs to be able to identify and delete just *the one row* with ID = 4. It shouldn’t delete *the entire directory* of tasks.

**[[](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/9a04c149-75f4-4234-bbdf-1a8802184f79)](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/9a04c149-75f4-4234-bbdf-1a8802184f79)**

**[Example of different delete behavior depending on the recognized URI.](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/9a04c149-75f4-4234-bbdf-1a8802184f79)**

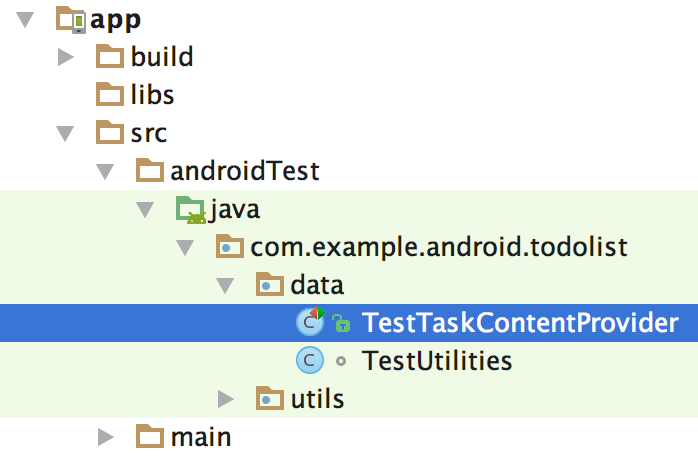
So, when a ContentProvider works with multiple types of data and corresponding URI’s, a UriMatcher plays an important role in accurate data management.

# A Small Note on Testing

In lesson the Storing Data in SQLite lesson, you were introduced to the idea of [**unit tests**](https://classroom.udacity.com/courses/ud851/lessons/b5ab398b-2d15-4840-b3ff-0171e84d1c8d/concepts/94302e18-e24e-4eb0-8767-17dfaceb6c30) that let you test whether a small piece of code works as you expected.

In the exercises for this lesson, you’ve been provided with unit tests that test the functionality of your TaskContentProvider. There are tests for the individual functions you’ll implement (like insert and delete) as well as tests to see if you’ve registered the ContentProvider correctly and if your UriMatcher is working as it should.

All of these tests are kept in the **src** > **androidTest** folder in the TestTaskContentProvider class.

**[[](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/cb1c595b-506c-467a-8409-82ea7e048fb7)](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/cb1c595b-506c-467a-8409-82ea7e048fb7)**

**[Location of TestTaskContentProvider.](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/cb1c595b-506c-467a-8409-82ea7e048fb7)**

To enable a test for a certain function, just uncomment that particular test by highlighting the text and selecting Code > Comment with Line Comment, as in the image below.

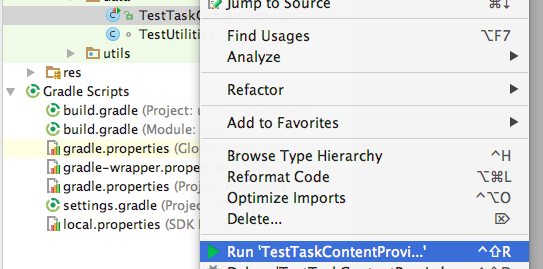
Note: If you uncomment some text code, and received some red lines indicating errors, you likely have to import some classes to complete the test code.

**[[](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/cb1c595b-506c-467a-8409-82ea7e048fb7)](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/cb1c595b-506c-467a-8409-82ea7e048fb7)**

**[(Left) Highlight desired test code. (Right) Uncomment the test you want to run.](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/cb1c595b-506c-467a-8409-82ea7e048fb7)**

## To Run a Test

To run all the tests in a class, right click on the class name of the test and select **Run <TestClassName>** as seen below:

**[[](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/cb1c595b-506c-467a-8409-82ea7e048fb7)](https://classroom.udacity.com/courses/ud851-gwg/lessons/88171055-d9e6-4da5-acc4-b92a302a75a8/concepts/cb1c595b-506c-467a-8409-82ea7e048fb7)**

## What Does the CustomCursorAdapter do?

The CustomCursorAdapter will inflate views using the xml layout file task\_layout, and create ViewHoldersthat will fill the main RecyclerView.

Each ViewHolder includes data about a single task: it’s text description and priority level. The priorityViewwill actually be a small colored circle that indicates the priority level 1-3 (1 is high and 3 is low).

The priority circle is a drawable resource, and its color is assigned to red, yellow, or green based on the priority level.

All of this code was included in your starter code, so no need to change anything in here.

## Query for One Item

Here's the code for querying a single task:

*// Implement query to handle requests for data by URI*

@Override

**public** Cursor **query**(@NonNull Uri uri, String[] projection, String selection,

String[] selectionArgs, String sortOrder) {

*// Get access to underlying database (read-only for query)*

**final** SQLiteDatabase db = mTaskDbHelper.getReadableDatabase();

*// Write URI match code*

*// Write a query for the tasks directory and default case*

**int** match = sUriMatcher.match(uri);

Cursor retCursor;

**switch** (match) {

*// Query for the tasks directory*

**case** TASKS:

retCursor = db.query(TABLE\_NAME,

projection,

selection,

selectionArgs,

**null**,

**null**,

sortOrder);

**break**;

*// Add a case to query for a single row of data by ID*

*// Use selections and selectionArgs to filter for that ID*

**case** TASK\_WITH\_ID:

*// Get the id from the URI*

String id = uri.getPathSegments().get(1);

*// Selection is the \_ID column = ?, and the Selection args = the row ID from the URI*

String mSelection = "\_id=?";

String[] mSelectionArgs = **new** String[]{id};

*// Construct a query as you would normally, passing in the selection/args*

retCursor = db.query(TABLE\_NAME,

projection,

mSelection,

mSelectionArgs,

**null**,

**null**,

sortOrder);

**break**;

*// Default exception*

**default**:

**throw** **new** UnsupportedOperationException("Unknown uri: " + uri);

}

*// Set a notification URI on the Cursor*

retCursor.setNotificationUri(getContext().getContentResolver(), uri);

*// Return the desired Cursor*

**return** retCursor;

}

# Update

The sample code for update is below. We won't be using this functionality in this app, but if you plan to provide your app's data to other applications, you should implement all of the operations that you want developers to be able to use.

*// Update won't be used in the final ToDoList app but is implemented here for completeness*

*// This updates a single item (by it's ID) in the tasks directory*

@Override

**public** **int** **update**(@NonNull Uri uri, ContentValues values, String selection,

String[] selectionArgs) {

*//Keep track of if an update occurs*

**int** tasksUpdated;

*// match code*

**int** match = sUriMatcher.match(uri);

**switch** (match) {

**case** TASK\_WITH\_ID:

*//update a single task by getting the id*

String id = uri.getPathSegments().get(1);

*//using selections*

tasksUpdated = mTaskDbHelper.getWritableDatabase().update(TABLE\_NAME, values, "\_id=?", **new** String[]{id});

**break**;

**default**:

**throw** **new** UnsupportedOperationException("Unknown uri: " + uri);

}

**if** (tasksUpdated != 0) {

*//set notifications if a task was updated*

getContext().getContentResolver().notifyChange(uri, **null**);

}

*// return number of tasks updated*

**return** tasksUpdated;

}

# GetType

Here's an example of getType for the ToDo list app:

*/\* getType() handles requests for the MIME type of data*

*We are working with two types of data:*

*1) a directory and 2) a single row of data.*

*This method will not be used in our app, but gives a way to standardize the data formats*

*that your provider accesses, and this can be useful for data organization.*

*For now, this method will not be used but will be provided for completeness.*

*\*/*

@Override

**public** String **getType**(@NonNull Uri uri) {

**int** match = sUriMatcher.match(uri);

**switch** (match) {

**case** TASKS:

*// directory*

**return** "vnd.android.cursor.dir" + "/" + TaskContract.AUTHORITY + "/" + TaskContract.PATH\_TASKS;

**case** TASK\_WITH\_ID:

*// single item type*

**return** "vnd.android.cursor.item" + "/" + TaskContract.AUTHORITY + "/" + TaskContract.PATH\_TASKS;

**default**:

**throw** **new** UnsupportedOperationException("Unknown uri: " + uri);

}

}

# Setup Sunshine's Content Provider and Query Solution

In this exercise you modified the WeatherProvider class so that it could perform a query. This required registering the content provider in the AndroidManifest.xml, creating a URIMatcher and finally completing the query.

## Notes on Solution Code

### Add the Content Provider to the Manifest

First, we add the content provider to the manifest, using a provider tag:

<provider

android:name=".data.WeatherProvider"

android:authorities="@string/content\_authority"

android:exported="false"/>

### Setup the URIMatcher

It's important to note that a lot of code was provided for you. The WeatherContract was updated to include the new URIs you needed for this exercise, namely:

* **content://com.example.android.sunshine/weather/** - The directory of all weather data. This is the same as the CONTENT\_URI for the weather table..
* **content://com.example.android.sunshine/weather/#** - A single item of data. The number here is meant to match a **date**. For these URIs the WeatherProvider includes the **[buildWeatherUriWithDate](https://github.com/udacity/ud851-Sunshine/blob/S09.01-Exercise-ContentProviderFoundation/app/src/main/java/com/example/android/sunshine/data/WeatherContract.java" \t "_blank)**method.

The URIMatcher should be set up in such a way that it matches and maps these two types of URI to integer constants.

So first things first, you need to define two integer constants:

**public** **static** **final** **int** CODE\_WEATHER = 100;

**public** **static** **final** **int** CODE\_WEATHER\_WITH\_DATE = 101;

After this you should write a static method to build the URI matcher.

**public** **static** UriMatcher **buildUriMatcher**() {

**final** UriMatcher matcher = **new** UriMatcher(UriMatcher.NO\_MATCH);

**final** String authority = WeatherContract.CONTENT\_AUTHORITY;

matcher.addURI(authority, WeatherContract.PATH\_WEATHER, CODE\_WEATHER);

matcher.addURI(authority, WeatherContract.PATH\_WEATHER + "/#", CODE\_WEATHER\_WITH\_DATE);

**return** matcher;

}

### Initialize the Content Provider

In this case because the underlying data structure is a SQLite database, you need to make a connection to that database in the onCreate method:

mOpenHelper = **new** WeatherDbHelper(getContext());

### Code Query

To code query, you'll need to use your URI matcher to take the incoming URI and figure out what it is

**public** Cursor **query**(@NonNull Uri uri, String[] projection, String selection,

String[] selectionArgs, String sortOrder) {

Cursor cursor;

**switch** (sUriMatcher.match(uri)) {

**case** CODE\_WEATHER\_WITH\_DATE: {

*// Code for querying with a date*

**break**;

}

**case** CODE\_WEATHER: {

*// Code for querying the weather table*

**break**;

}

**default**:

**throw** **new** UnsupportedOperationException("Unknown uri: " + uri);

}

cursor.setNotificationUri(getContext().getContentResolver(), uri);

**return** cursor;

}

The simpler of the two cases is querying the entire directory of weather, seen below:

cursor = mOpenHelper.getReadableDatabase().query(

WeatherContract.WeatherEntry.TABLE\_NAME,

projection,

selection,

selectionArgs,

**null**,

**null**,

sortOrder);

When you're querying with a date, you can use getLastPathSegment to get the date string, then pass it in as a selection argument. The selection parameter should reference the date column, as seen below:

String normalizedUtcDateString = uri.getLastPathSegment();

String[] selectionArguments = **new** String[]{normalizedUtcDateString};

cursor = mOpenHelper.getReadableDatabase().query(

*/\* Table we are going to query \*/*

WeatherContract.WeatherEntry.TABLE\_NAME,

projection,

WeatherContract.WeatherEntry.COLUMN\_DATE + " = ? ",

selectionArguments,

**null**,

**null**,

sortOrder);

**break**;

Finally, it's important to set the notification URI for the cursor. We'll use this later when we implement a class called the CursorLoader.

cursor.setNotificationUri(getContext().getContentResolver(), uri);

See the full list of edits in the solution diff below.

### Solution Code

**Solution:** [[**S09.01-Solution-ContentProviderFoundation**](https://github.com/udacity/ud851-Sunshine/tree/student/S09.01-Solution-ContentProviderFoundation)][[**Diff**](https://github.com/udacity/ud851-Sunshine/compare/S09.01-Exercise-ContentProviderFoundation...S09.01-Solution-ContentProviderFoundation)]

# Add Delete Functionality Solution

In this exercise you implemented the code to delete an entire database worth of weather data. Wow!

## Notes on Solution Code

Note that in the solution code, you only need to implementing delete for the **CODE\_WEATHER** case because you're deleting the entire database. You also don't have to actually use this delete functionality in the app, so it's up to you to run the tests and make sure everything was completed correctly. With that said, here's the code for delete:

@Override

**public** **int** **delete**(@NonNull Uri uri, String selection, String[] selectionArgs) {

**int** numRowsDeleted;

**if** (**null** == selection) selection = "1";

**switch** (sUriMatcher.match(uri)) {

**case** CODE\_WEATHER:

numRowsDeleted = mOpenHelper.getWritableDatabase().delete(

WeatherContract.WeatherEntry.TABLE\_NAME,

selection,

selectionArgs);

**break**;

**default**:

**throw** **new** UnsupportedOperationException("Unknown uri: " + uri);

}

**if** (numRowsDeleted != 0) {

getContext().getContentResolver().notifyChange(uri, **null**);

}

**return** numRowsDeleted;

}

### Solution Code

**Solution:** [[**S09.03-Solution-ContentProviderDelete**](https://github.com/udacity/ud851-Sunshine/tree/student/S09.03-Solution-ContentProviderDelete)][[**Diff**](https://github.com/udacity/ud851-Sunshine/compare/S09.03-Exercise-ContentProviderDelete...S09.03-Solution-ContentProviderDelete)]

# Using a CursorLoader Solution

In this exercise you replaced the AsyncTaskLoader with a CursorLoader that gets data from the ContentProvider. Great job!!

## Notes on Solution Code

One of the biggest changes to the Sunshine code was to update the ForecastAdapter and change its onBindViewHolder method so that it takes all of the data from a cursor and uses that to populate the views.

The final implementation of onBindViewHolder should look like this:

@Override

**public** **void** **onBindViewHolder**(ForecastAdapterViewHolder forecastAdapterViewHolder, **int** position) {

*// Move the cursor to the appropriate position*

mCursor.moveToPosition(position);

*// Generate a weather summary with the date, description, high and low*

*/\* Read date from the cursor \*/*

**long** dateInMillis = mCursor.getLong(MainActivity.INDEX\_WEATHER\_DATE);

*/\* Get human readable string using our utility method \*/*

String dateString = SunshineDateUtils.getFriendlyDateString(mContext, dateInMillis, **false**);

*/\* Use the weatherId to obtain the proper description \*/*

**int** weatherId = mCursor.getInt(MainActivity.INDEX\_WEATHER\_CONDITION\_ID);

String description = SunshineWeatherUtils.getStringForWeatherCondition(mContext, weatherId);

*/\* Read high temperature from the cursor (in degrees celsius) \*/*

**double** highInCelsius = mCursor.getDouble(MainActivity.INDEX\_WEATHER\_MAX\_TEMP);

*/\* Read low temperature from the cursor (in degrees celsius) \*/*

**double** lowInCelsius = mCursor.getDouble(MainActivity.INDEX\_WEATHER\_MIN\_TEMP);

String highAndLowTemperature =

SunshineWeatherUtils.formatHighLows(mContext, highInCelsius, lowInCelsius);

String weatherSummary = dateString + " - " + description + " - " + highAndLowTemperature;

*// Display the summary that you created above*

forecastAdapterViewHolder.weatherSummary.setText(weatherSummary);

}

To view all the changes to the ForecastAdapter and the MainActivity loader code, see the solution and comparison code linked below.

### Solution Code

**Solution:** [[**S09.04-Solution-UsingCursorLoader**](https://github.com/udacity/ud851-Sunshine/tree/student/S09.04-Solution-UsingCursorLoader)][[**Diff**](https://github.com/udacity/ud851-Sunshine/compare/S09.04-Exercise-UsingCursorLoader...S09.04-Solution-UsingCursorLoader)]

# More Weather Details Solution

In this exercise, you've uses CursorLoaders to display more weather information in the Detail Layout.

Here's the solution code for the onCreateLoader:

@Override

**public** Loader<Cursor> **onCreateLoader**(**int** loaderId, Bundle loaderArgs) {

**switch** (loaderId) {

**case** ID\_DETAIL\_LOADER:

**return** **new** CursorLoader(**this**,

mUri,

WEATHER\_DETAIL\_PROJECTION,

**null**,

**null**,

**null**);

**default**:

**throw** **new** RuntimeException("Loader Not Implemented: " + loaderId);

}

}

onLoadFinished should start by checking if cursor has valid data:

**boolean** cursorHasValidData = **false**;

**if** (data != **null** && data.moveToFirst()) {

*/\* We have valid data, continue on to bind the data to the UI \*/*

cursorHasValidData = **true**;

}

**if** (!cursorHasValidData) {

*/\* No data to display, simply return and do nothing \*/*

**return**;

}

Then for each piece of weather information, retrieve it from the cursor and display it in the appropriate view. For example, the fist text view should the display the date as follows:

**long** localDateMidnightGmt = data.getLong(INDEX\_WEATHER\_DATE);

String dateText = SunshineDateUtils.getFriendlyDateString(**this**, localDateMidnightGmt, **true**);

mDateView.setText(dateText);

To view all the changes to the ForecastAdapter, MainActivity and the new layout code, see the solution and comparison code linked below.

### Solution Code

**Solution:** [[**S09.05-Solution-MoreDetails**](https://github.com/udacity/ud851-Sunshine/tree/student/S09.05-Solution-MoreDetails)][[**Diff**](https://github.com/udacity/ud851-Sunshine/compare/S09.05-Exercise-MoreDetails...S09.05-Solution-MoreDetails)]