MCA253 - Mobile Applications

Unit:3.3
Content Provider

Dr. Siddesha S MCA, M.Sc Tech (by Research) , Ph.D
Asst. Professor,
Dept. of Computer Applications,
JSS Science and Technology University
Mysuru – 570 006

Introduction

- ✓ What are content providers ?
- ✓ How to use a content provider in Android ?
- ✓ How to create and use your own content provider ?

- ❑ Previous section explains the various ways to persist data—using shared preferences, files, as well as SQLite databases.
- ❑ Although using the database approach is the recommended way to save structured and complex data, sharing data is a challenge because the database is accessible to only the package that created it.

Sharing Data in Android

- In Android, using a content provider is the recommended way to share data across packages.
- Think of a content provider as a data store.
- How it stores its data is not relevant to the application using it.
- However, the way in which packages can access the data stored in it using a consistent programming interface is important.
- A content provider behaves very much like a database—you can query it, edit its content, and add or delete content.
- However, unlike a database, a content provider can use different ways to store its data.
- The data can be stored in a database, in files, or even over a network.

Sharing Data in Android

- Android ships with many useful content providers, including the following:
  - **Browser**—Stores data such as browser bookmarks, browser history, and so on
  - **CallLog**—Stores data such as missed calls, call details, and so on
  - **Contacts**—Stores contact details
  - **MediaStore**—Stores media files such as audio, video, and images
  - **Settings**—Stores the device's settings and preferences
- Besides the many built-in content providers, you can also create your own content providers.
- To query a content provider, you specify the query string in the form of a Uniform Resource Identifier (URI), with an optional specifier for a particular row.

Sharing Data in Android

- The format of the query URI:

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

- The various parts of the URI are as follows,

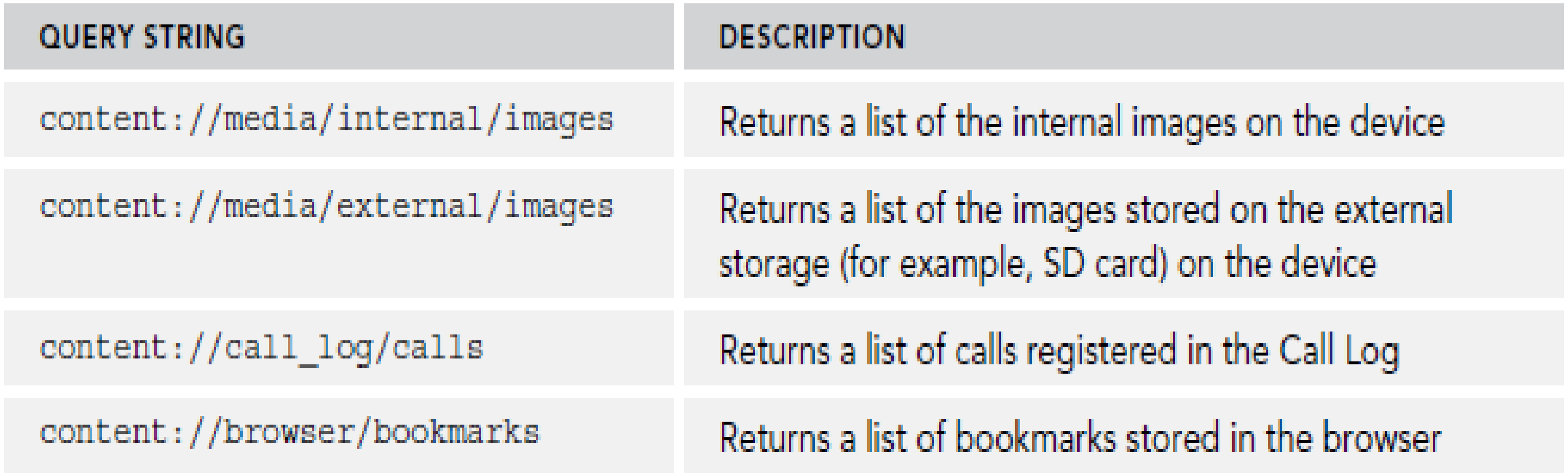
- The standard prefix for content providers is always content**://.**
  The authority specifies the name of the content provider. An example would be contacts for the built-in Contacts content provider. For third-party content providers, this could be the fully qualified name, such as **`com.wrox.provider`** or **`com.sidprojects.provider`**.

- The data path specifies the kind of data requested. For example, if you are getting all the contacts from the Contacts content provider then the data path would be people, and the URI would look like this: **`content://contacts/people.`**

Sharing Data in Android

- The id specifies the specific record requested. For example, if you are looking for contact number 2 in the Contacts content provider, the URI would look like this: **`content://contacts/people/2.`**

- **Example Query Strings:**



QUERY STRING	DESCRIPTION
`content://media/internal/images`	Returns a list of the internal images on the device
`content://media/external/images`	Returns a list of the images stored on the external storage (for example, SD card) on the device
`content://call_log/calls`	Returns a list of calls registered in the Call Log
`content://browser/bookmarks`	Returns a list of bookmarks stored in the browser

Using a Content Provider

Program: ContentProvider

- This example, retrieves the contacts stored in the Contacts application and displayed them in the ListView.
- First, specify the URI for accessing the Contacts application:

```
Uri allContacts = Uri.parse("content://contacts/people");
```

- Check that your app has permission to access the Contacts
- If the application does not have permission, a request for permission is issued (causing Android to pop the permission dialog).
- If the application does have permission, the **ListContacts()** method is called.
- The **getContentResolver()** method returns a **ContentResolver** object, which helps to resolve a content URI with the appropriate content provider.

Using a Content Provider

- The CursorLoader class (only available beginning with Android API level 11 and later) performs the cursor query on a background thread and therefore does not block the application UI.
- The SimpleCursorAdapter object maps a cursor to TextViews (or ImageViews) defined in your XML file (activity_main.xml).
- It maps the data (as represented by columns) to views (as represented by views)
- Like the managedQuery() method, one of the constructors for the SimpleCursorAdapter class has been deprecated.
- For devices running Honeycomb or later versions, you need to use the new constructor for the SimpleCursorAdapter class with one additional argument.

Using a Content Provider

- The flag registers the adapter to be informed when there is a change in the content provider.
- Note that for your application to access the Contacts application, you need to have the READ_CONTACTS permission in AndroidManifest.xml file.

Predefined Query String Constants

- Besides using the query URI, you can use a list of predefined query string constants in Android to specify the URI for the different data types. For example, besides using the query content

`//contacts/people`, can be rewritten as

```
Uri allContacts = Uri.parse("content://contacts/people");
```

using one of the predefined constants in Android

```
Uri allContacts = ContactsContract.Contacts.CONTENT_URI;
```

Projections

- The third parameter for the CursorLoader class controls how many columns are returned by the query.
- This parameter is known as the projection.
- You can specify the exact columns to return by creating an array containing the name of the column to return, like this

```
String[] projection = new String[]
       {ContactsContract.Contacts._ID,
       ContactsContract.Contacts.DISPLAY_NAME,
       ContactsContract.Contacts.HAS_PHONE_NUMBER};
Cursor c;
       CursorLoader cursorLoader = new CursorLoader(
       this,
       allContacts,
       projection,
       null,
       null,
       null,
       c = cursorLoader.loadInBackground();
```

Filtering

- The fourth and fifth parameters for the CursorLoader class enable you to specify a SQL WHERE clause to filter the result of the query.

```
Cursor c;
 CursorLoader cursorLoader = new CursorLoader(
   this,allContacts,projection,
ContactsContract.Contacts.DISPLAY_NAME + " LIKE '%Lee'", null ,
null);
c = cursorLoader.loadInBackground();
```

Sorting

- The last parameter of the CursorLoader class enables you to specify a SQL ORDER BY clause to sort the result of the query,

```
Cursor c;
CursorLoader cursorLoader = new CursorLoader( this,allContacts,
projection,
ContactsContract.Contacts.DISPLAY_NAME + " LIKE ?",
new String[] {"%Lee"},
ContactsContract.Contacts.DISPLAY_NAME + " ASC");
c = cursorLoader.loadInBackground();
```

Creating Your Own Content Providers

- ❑ Creating your own content provider in Android is relatively simple.
- ❑ All you need to do is extend the abstract ContentProvider class and override the various methods defined within it.

Java Class Name: BooksProvider

- ❑ In this example, you first create a class named BooksProvider that extends the ContentProvider base class.
- ❑ The various methods to override in this class are as follows:

- ➢ getType()—Returns the MIME type of the data at the given URI.
- ➢ onCreate()—Called when the provider is started.
- ➢ query()—Receives a request from a client. The result is returned as a Cursor object.

Creating Your Own Content Providers

- insert()—Inserts a new record into the content provider
- delete()—Deletes an existing record from the content provider.
- update()—Updates an existing record from the content provider

- Within your content provider, you are free to choose how you want to store your data—in a traditional file system, XML, a database, or even through web services. For this example, you use the SQLite database approach discussed in the previous section.
- Define the constants for the DB within the BooksProvider class
- Used an UriMatcher object to parse the content URI that is passed to the content provider through a ContentResolver.
- For example, the following content URI represents a request for all books in the content provider:

  content://com.example.mycontentprovider.Books/books

Creating Your Own Content Providers

- The following represents a request for a particular book with _id 5:

```
content://com.example.mycontentprovider.Books/books\5
```

- This content provider uses a SQLite database to store the books. Note that you use the SQLiteOpenHelper helper class to help manage database.
- Next, override the getType() method to uniquely describe the data type for the content provider.
- Using the UriMatcher object, vnd.android.cursor.item/vnd.<package name>.books is returned
- For a single book, and vnd.android.cursor.dir/vnd.<package name>.books is returned for multiple books

Creating Your Own Content Providers

- Next, override the onCreate() method to open a connection to the database when the content provider is started
- Next, override the query() method to allow clients to query for books.
- By default, the result of the query is sorted using the title field.
- The resulting query is returned as a Cursor object.
- To allow a new book to be inserted into the content provider, override the insert() method
- After the record is inserted successfully, call the notifyChange() method of the ContentResolver.
- This notifies registered observers that a row was updated.
- To delete a book, override the delete() method

Creating Your Own Content Providers

- Likewise, call the notifyChange() method of the ContentResolver after the deletion.
- This notifies registered observers that a row was deleted.
- To update a book, you override the update() method.
- As with the insert() and delete() methods, you called the notifyChange() method of the ContentResolver after the update.
- This notifies registered observers that a row was updated.
- Finally, to register your content provider with Android, modify the AndroidManifest.xml file by adding the <provider> element.

Using the Content Provider

Program name: MyContentProvider

- First, modify the activity so that users can enter a book’s ISBN and title to add to the content provider that just created **[BookProvider]**
- To add a book to the content provider, create a new ContentValues object and then populate it with the various information about a book.
- Notice that because your content provider is in the same package, you can use the BooksProvider .TITLE and the BooksProvider.ISBN constants to refer to the "title" and "isbn" fields, respectively.

```
ContentValues values = new ContentValues();
values.put(BooksProvider.TITLE, ((EditText)
findViewById(R.id.txtTitle)).getText().toString());
values.put(BooksProvider.ISBN, ((EditText)
findViewById(R.id.txtISBN)).getText().toString());
Uri uri = getContentResolver().insert(
BooksProvider.CONTENT_URI, values);
```

Using the Content Provider

- If you were accessing this content provider from another package, then you would not be able to use these constants.
- In that case, you need to specify the field name directly, like this:

```
ContentValues values = new ContentValues();
values.put("title", ((EditText)
findViewById(R.id.txtTitle)).getText().toString());
values.put("isbn", ((EditText)
findViewById(R.id.txtISBN)).getText().toString());
Uri uri = getContentResolver().insert(
Uri.parse(" content://com.example.mycontentprovider.Books/books"),
values);
```

- Also note that for external packages, you need to refer to the content URI using the fully qualified content URI,

```
Uri.parse("content://com.example.mycontentprovider.Books/books”)
```

Using the Content Provider

- Retrieve all the titles in the content provider
- This query returns the result sorted in descending order based on the title field.
- If you want to update a book’s detail, call the update() method with the content URI, indicating the book’s ID as follows,

```
ContentValues editedValues = new ContentValues();
editedValues.put(BooksProvider.TITLE, "Android Tips and
 Tricks");
 getContentResolver().update(
           Uri.parse(
           "content://com.jfdimarzio.provider.Books/books/2"
           ), editedValues, null, null);
```

Using the Content Provider

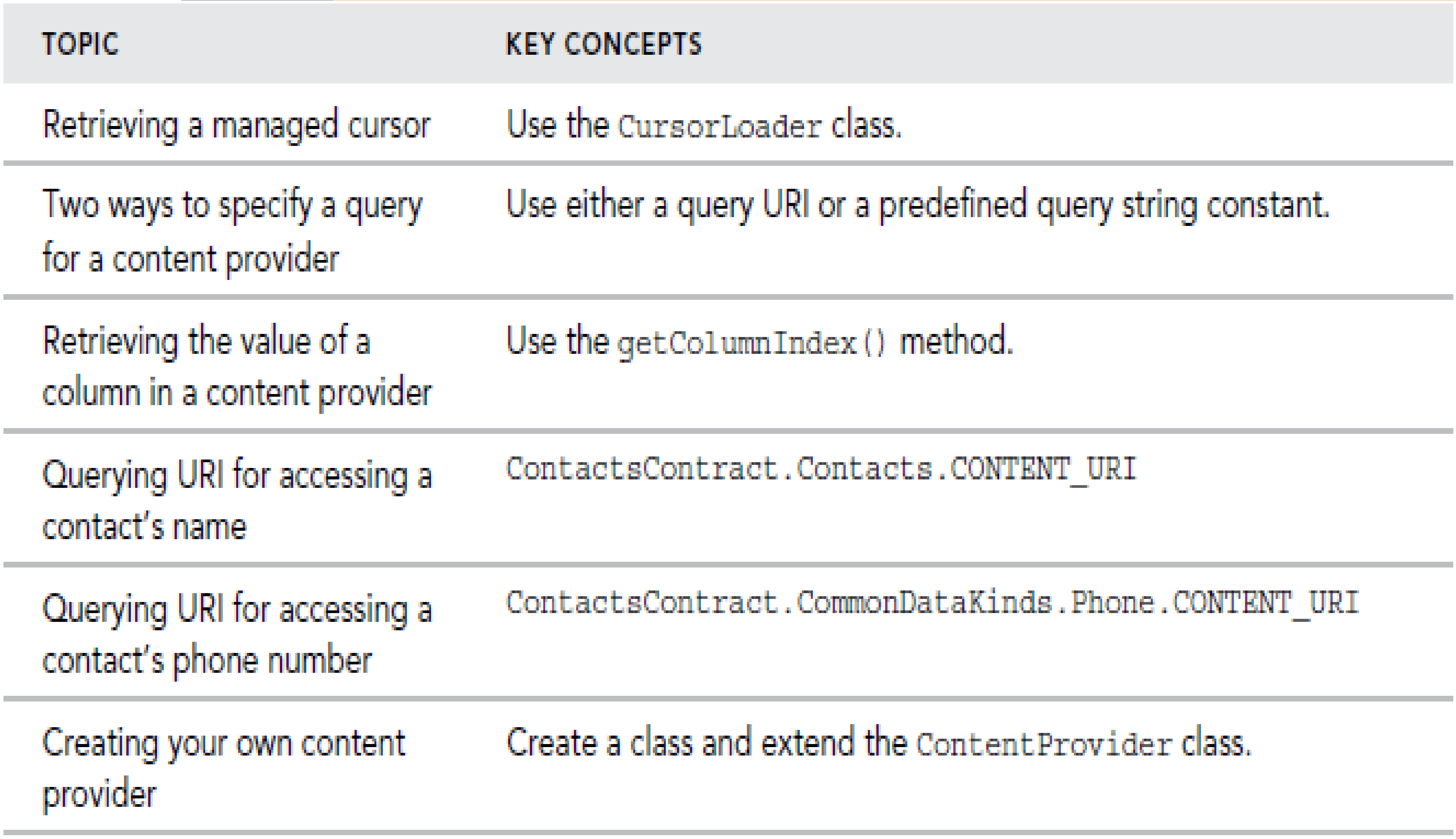
- To delete a book, use the delete() method with the content URI, indicating the book’s ID:

```
//---delete a title---
getContentResolver().delete(
Uri.parse("content://com.jfdimarzio.provider.Books/books/2"),
null, null);
```

- To delete all books, simply omit the book’s ID in your content URI:

```
//---delete all titles---
getContentResolver().delete(
Uri.parse("content://com.jfdimarzio.provider.Books/books"),
null, null);
```

Summary



TOPIC	KEY CONCEPTS
Retrieving a managed cursor	Use the `CursorLoader` class.
Two ways to specify a query for a content provider	Use either a query URI or a predefined query string constant.
Retrieving the value of a column in a content provider	Use the `getColumnIndex()` method.
Querying URI for accessing a contact's name	`ContactsContract.Contacts.CONTENT_URI`
Querying URI for accessing a contact's phone number	`ContactsContract.CommonDataKinds.Phone.CONTENT_URI`
Creating your own content provider	Create a class and extend the `ContentProvider` class.