Content Providers

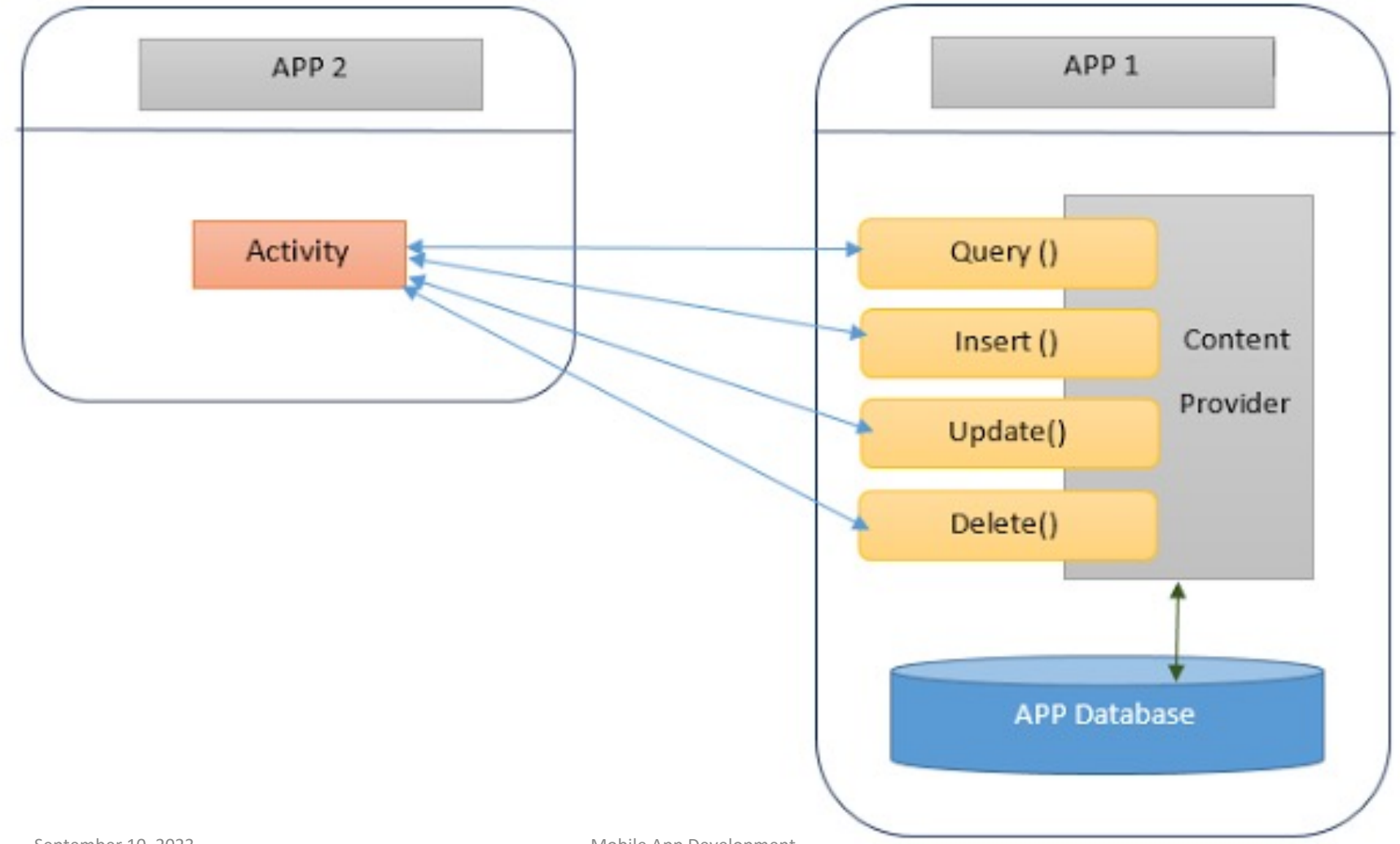
Outline

- Introduction
  - ContentProvider
  - Common Providers
  - Content URIs
- Common Content Providers
  - Contact Providers
  - CallLog Provider
  - SMS Provider
- ContentResolver
  - Retrieving Data from the Provider
  - Inserting, Updating, and Deleting Data
- Demonstration

Introduction

- In Android security model, one application cannot directly access (read/write) other application's data. Every application has its own id data directory and own protected memory area.
- Content provider is the best way to share data across applications. Content provider is a set of data wrapped up in a custom API to read and write.
- Applications have to register themselves as a provider of data. Other applications can request Android to read/write that data through a fixed API.
- In this module, you will learn how to design and develop content providers.

Introduction



Outline

- Introduction
  - ContentProvider
  - **Common Providers**
  - Content URIs
- Common Content Providers
  - Contact Providers
  - CallLog Provider
  - SMS Provider
- ContentResolver
  - Retrieving Data from the Provider
  - Inserting, Updating, and Deleting Data
- Demonstration

Common Content Providers

https://developer.android.com/training/permissions/requesting.html

Provider	SDK	Usage
Browser	1	Manages your web-searches, bookmarks and browsing-history.
CalendarContract	14	Manages the calendars on the user's device.
CallLog	1	Keeps track of your call history.
Contacts	1	The old and deprecated content provider for managing contacts. You should only use this provider if you need to support an SDK prior to SDK 5!
ContactsContract	5	Deals with all aspects of contact management. Supersedes the Contacts-content provider.
MediaStore	1	The content provider responsible for all your media files like music, video and pictures.
Settings	1	Manages all global settings of your device.
UserDictionary	3	Keeps track of words you add to the default dictionary.

Common Content Providers

- Please make use of the standard providers whenever you need data they provide.
- For example there are apps that ignore the ContactsContract. So the user might end up adding the same contact in multiple apps – which is pretty annoying. Something like this will not help your rating in Android's Market.
- Of course you should always keep in mind that not all of the standard providers might be present on certain devices. E.g. a tablet might have no CallLog.

Outline

- Introduction
  - ContentProvider
  - Common Providers
  - **Content URIs**
- Common Content Providers
  - Contact Providers
  - CallLog Provider
  - SMS Provider
- ContentResolver
  - Retrieving Data from the Provider
  - Inserting, Updating, and Deleting Data
- Demonstration

Content URIs

- The most important concept to understand when dealing with content providers is the content URI
- Whenever you want to access data from a content provider you have to specify a URI. URIs for content providers look like this:

```
content://authority/optionalPath/optionalId
```

Content URIs (Examples)

- The **scheme** for content providers is always "content". The colon and double-slash "://" are a fixed part and separate the scheme from the authority.
- The next part is the **authority** for the content provider. Authorities have to be unique for every content provider. Thus the naming conventions should follow the Java package name rules.

```
content://authority/optionalPath/optionalId
```

```
content://com.mediastore
```

Content URIs

- The third part, the optional **path**, is used to distinguish the kinds of data your content provider offers.
- The content provider for Android's mediastore, for example, distinguishes between audio files, video files and images using different paths for each of these types of media.
- This way a content provider can support different types of data that are nevertheless related.
- For totally unrelated data though you should use different content providers – and thus different authorities.

Content URIs (Examples)

```
content://com.mediastore/video
```

```
content://com.mediastore/image
```

```
content://com.mediastore/audio
```

```
content://com.mediastore/food
```

```
content://com.mediastore/money
```

Content URIs (Examples)

- The last element is the optional **id**, which – if present – must be numeric. The id is used whenever you want to access a single record (e.g. a specific video file).

```
content://authority/optionalPath/optionalId
```

```
content://com.mediastore/video/1
```

```
content://com.mediastore/image/2
```

```
content://com.mediastore/audio/10
```

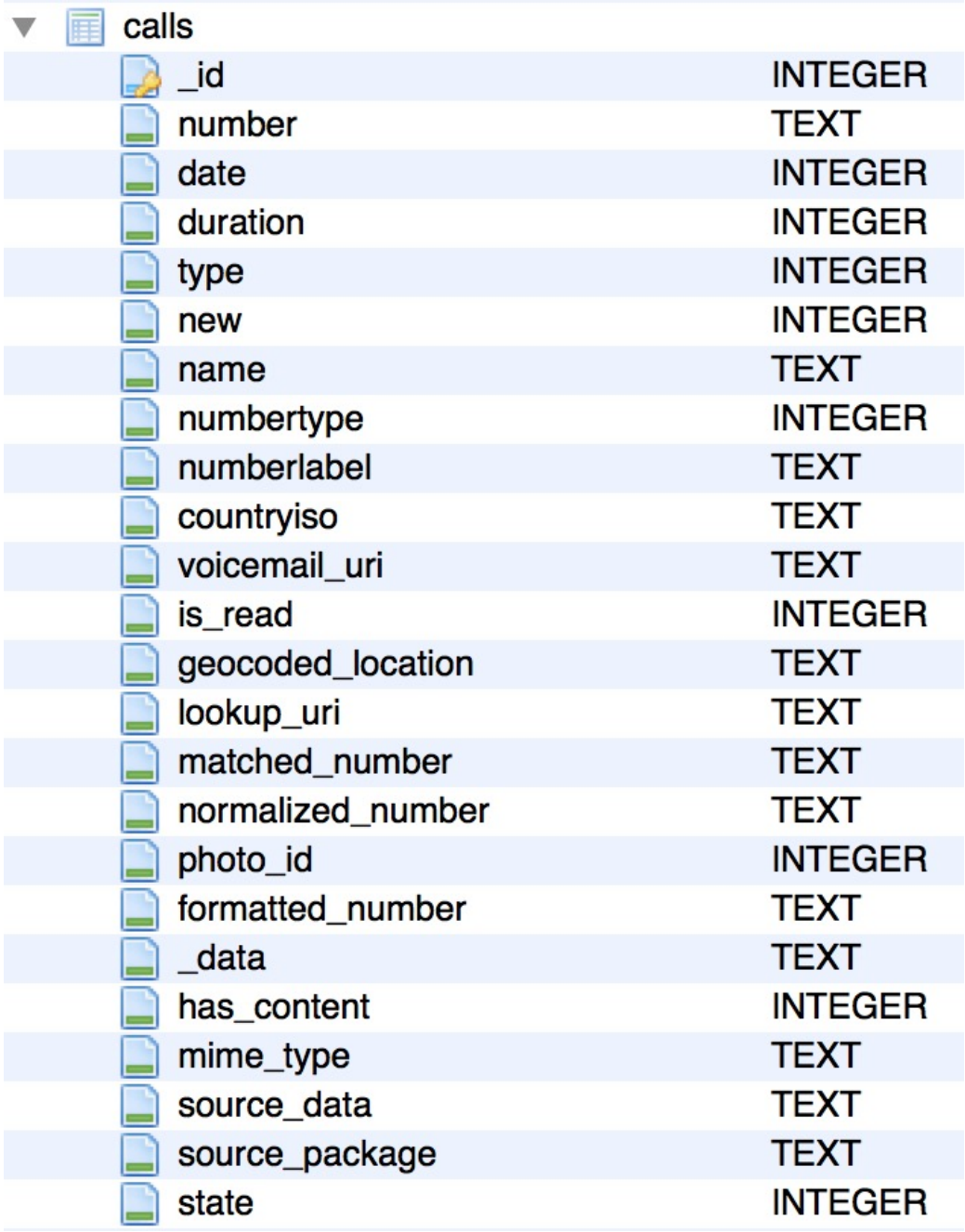
Content URIs (Common Content URIs)

Provider	CONTENT_URI	CONTENT_URI Values
CallLog.Calls	**CallLog.Calls. CONTENT_URI**	content://call_log/calls
ContactsContract.Contacts	**ContactsContract .Contacts.CONTENT_URI**	content://com.android.contacts/contacts
ContactsContract.RawContacts	**ContactsContract .RawContacts.CONTENT_URI**	content://com.android.contacts/raw_contacts
ContactsContract.Data	**ContactsContract .Data.CONTENT_URI**	content://com.android.contacts/data
Telephony.Sms	**Telephony.Sms.CONTENT_URI**	content://sms

Outline

- Introduction
  - ContentProvider
  - Common Providers
  - Content URIs
- Common Content Providers
  - **CallLog.Calls Provider**
  - Contact Providers
  - SMS Provider
- ContentResolver
  - Retrieving Data from the Provider
  - Inserting, Updating, and Deleting Data
- Demonstration

CallLog Provider (*calls* SQLite Table)



CallLog Provider (*calls* SQLite Table)

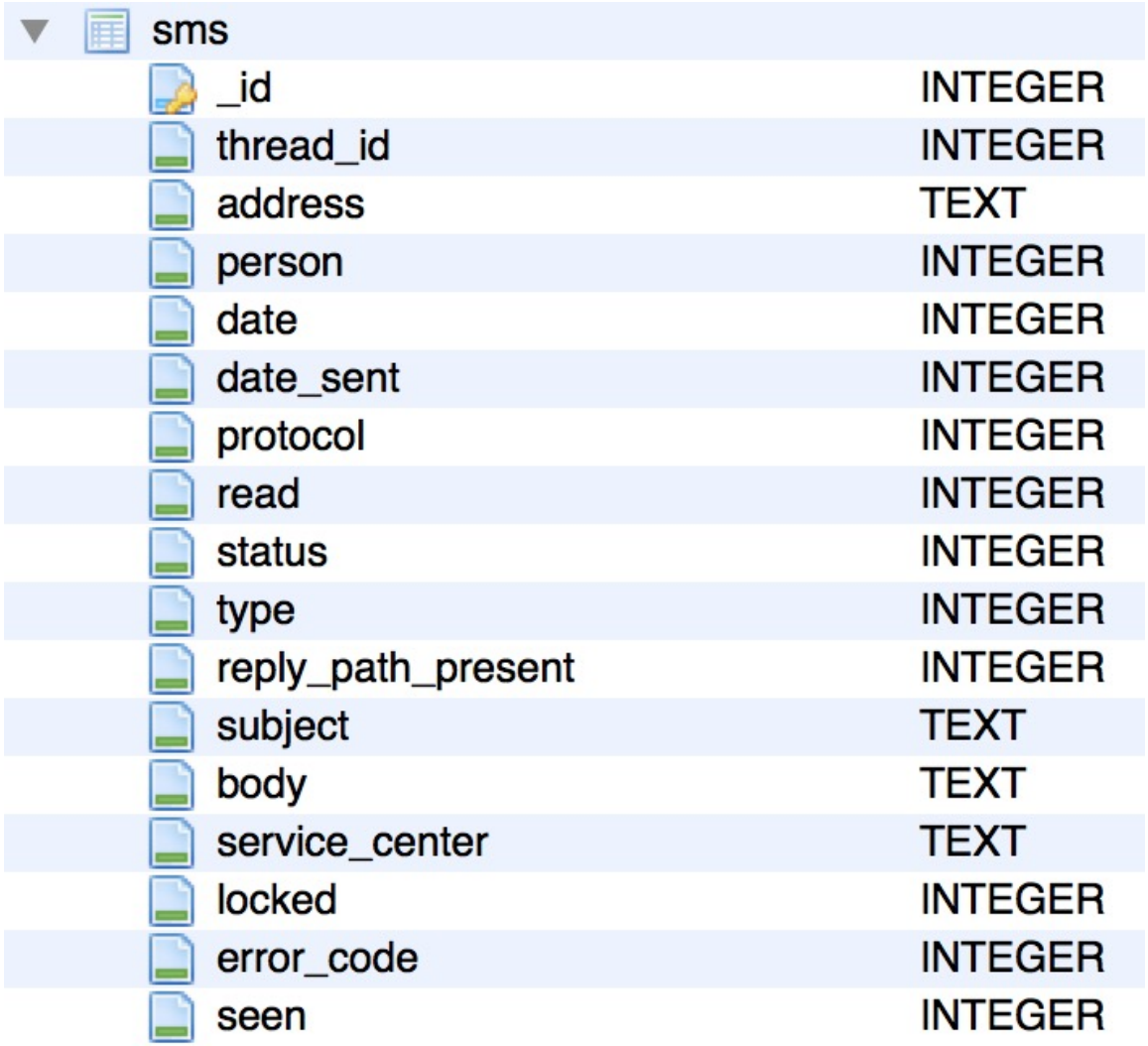
Column name	Static Variable Alias
number	`CallLog.Calls.Number`
date	`CallLog.Calls.Date`
duration	`CallLog.Calls.Duration`

type	`CallLog.Calls.Type`
	Allowed values are:
	`CallLog.Calls.INCOMING_TYPE (1)`
	`CallLog.Calls.OUTGOING_TYPE (2)`
	`CallLog.Calls.MISSED_TYPE (3)`
	`CallLog.Calls.VOICEMAIL_TYPE (4)`

Outline

- Introduction
  - ContentProvider
  - Common Providers
  - Content URIs
- Common Content Providers
  - Contact Providers
  - CallLog.Calls Provider
  - **SMS Provider**
- ContentResolver
  - Retrieving Data from the Provider
  - Inserting, Updating, and Deleting Data
- Demonstration

SMS Provider (*sms* SQLite Table)



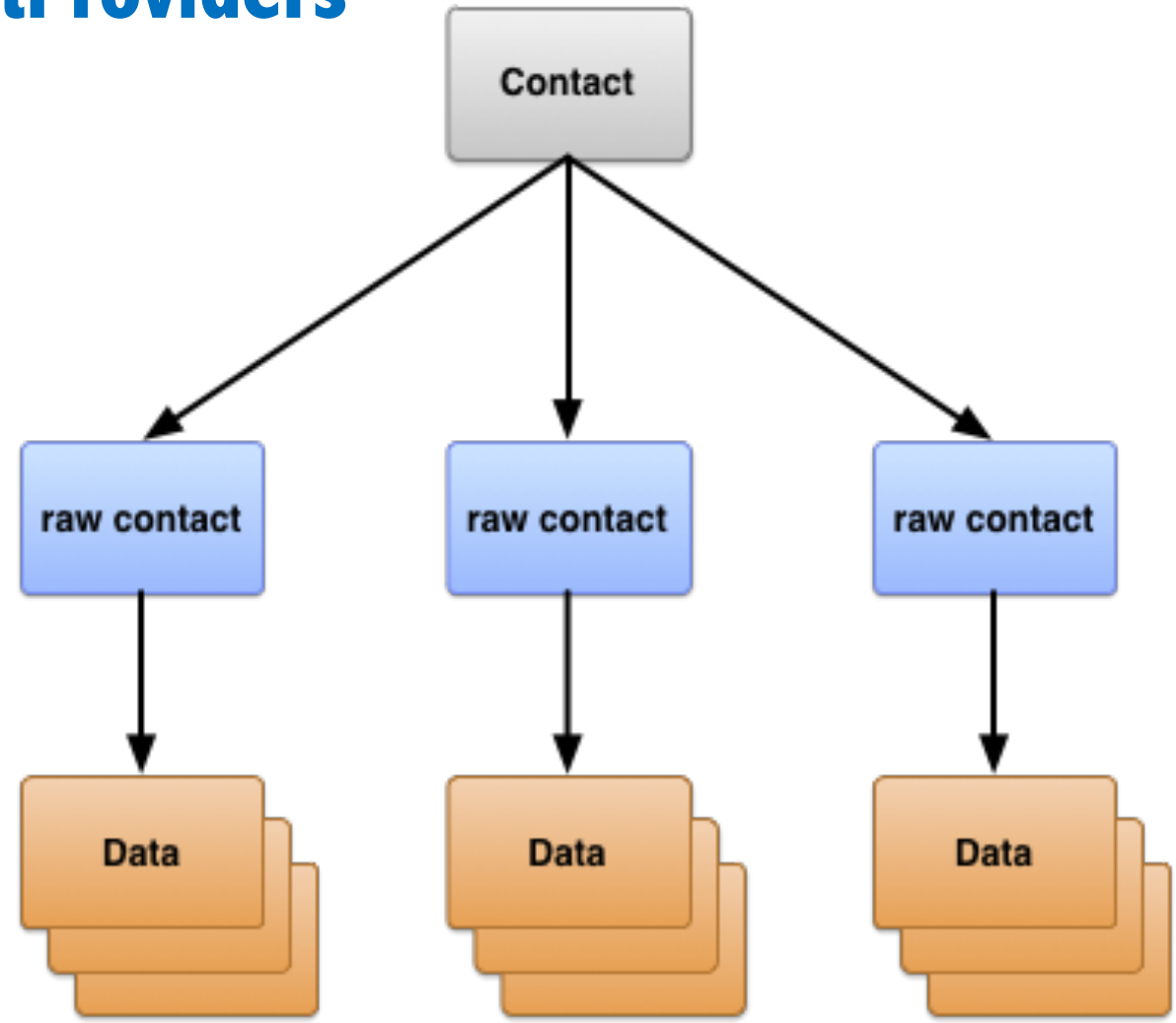
SMS Provider (*sms* SQLite Table)

Column name	Static Variable Alias
address	`Telephony.TextBasedSmsColumns.ADDRESS`
body	`Telephony.TextBasedSmsColumns.BODY`
date	`Telephony.TextBasedSmsColumns.DATE`
date_sent	`Telephony.TextBasedSmsColumns.DATE_SENT`

Outline

- Introduction
  - ContentProvider
  - Common Providers
  - Content URIs
- Common Content Providers
  - CallLog Provider
  - SMS Provider
  - **Contact Providers**
- ContentResolver
  - Retrieving Data from the Provider
  - Inserting, Updating, and Deleting Data
- Demonstration

ContactProviders



https://developer.android.com/reference/android/provider/ContactsContract.Contacts

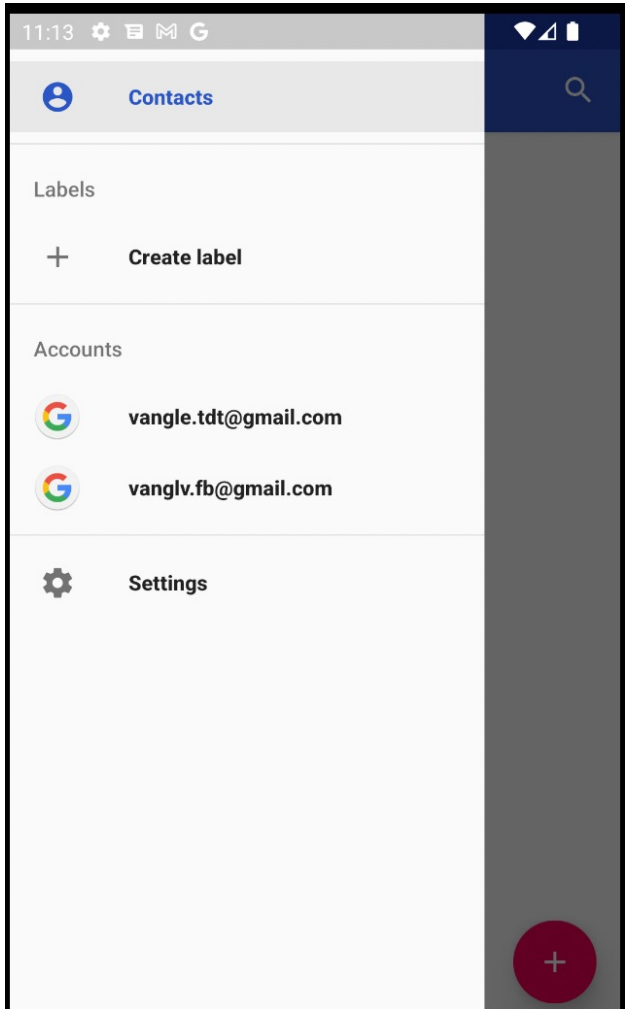
ContactProviders

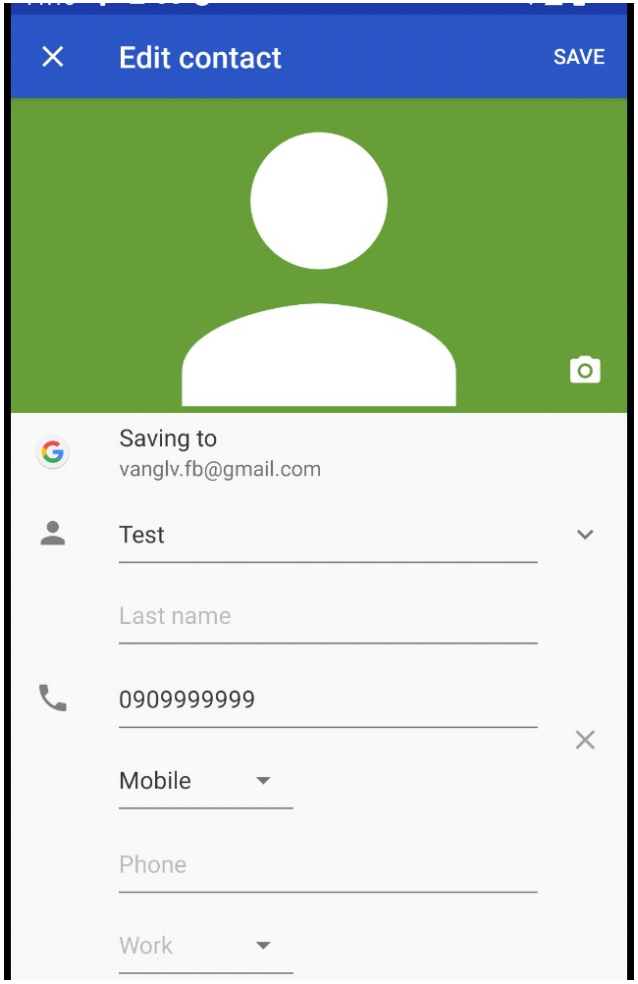
- The Contacts database is divided into 3 tables ***contacts***, ***raw contacts***, and ***data***.
- Each table contains column (_ID) which is an auto incremented primary key.
- ***data*** table contains all the contact info like phone number, mail id, address etc.
- The ***raw contacts*** points to the actual contact created. Hence we use the ***raw contacts*** while adding a contact.
- The user cannot add any data in the **contacts** table. The data in this table is populated internally due to aggregation of contacts.

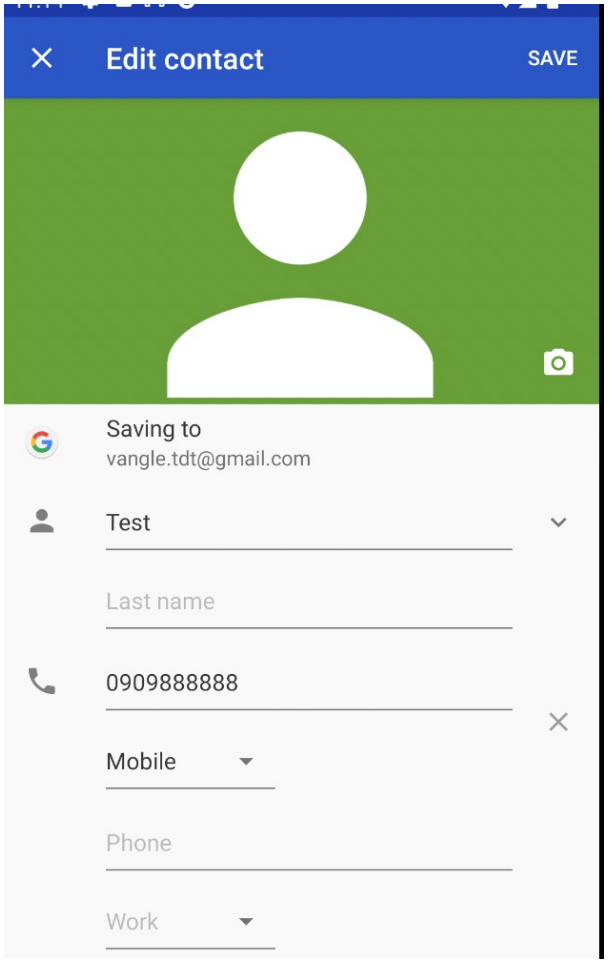
https://stackoverflow.com/questions/5151885/android-new-data-record-is-added-to-the-wrong-contact/5158059#5158059

Contacts vs RawContacts

- The distinction between Contacts and RawContacts becomes more apparent when you have contacts with multiple phone numbers or email addresses associated with **different accounts.**
- In those cases, the Android Contacts app will create separate RawContacts for each account and link them together to form a single Contact.


11:13
Contacts
Labels
Create label
Accounts
vangle.tdt@gmail.com
vanglv.fb@gmail.com
Settings


Edit contact
SAVE
Saving to
vanglv.fb@gmail.com
Test
Last name
0909999999
Mobile
Phone
Work


Edit contact
SAVE
Saving to
vangle.tdt@gmail.com
Test
Last name
0909888888
Mobile
Phone
Work

ContactsContract.Contacts Table

Constants		
String	CONTACT_LAST_UPDATED_TIMESTAMP	Timestamp (milliseconds since epoch) of when this contact was last updated.
String	DISPLAY_NAME	The display name for the contact.
String	HAS_PHONE_NUMBER	An indicator of whether this contact has at least one phone number.
String	IN_DEFAULT_DIRECTORY	Flag that reflects whether the contact exists inside the default directory.
String	IN_VISIBLE_GROUP	Flag that reflects the `GROUP_VISIBLE` state of any `ContactsContract.CommonDataKinds.GroupMembership` for this contact.
String	IS_USER_PROFILE	Flag that reflects whether this contact represents the user's personal profile entry.
String	LOOKUP_KEY	An opaque value that contains hints on how to find the contact if its row id changed as a result of a sync or aggregation.
String	NAME_RAW_CONTACT_ID	Reference to the row in the RawContacts table holding the contact name.
String	PHOTO_FILE_ID	Photo file ID of the full-size photo.
String	PHOTO_ID	Reference to the row in the data table holding the photo.
String	PHOTO_THUMBNAIL_URI	A URI that can be used to retrieve a thumbnail of the contact's photo.
String	PHOTO_URI	A URI that can be used to retrieve the contact's full-size photo.

ContactsContract.RawContacts Table

Column name	Use
ACCOUNT_NAME	The account name for the account type that's the source of this raw contact. For example, the account name of a Google account is one of the device owner's Gmail addresses.
ACCOUNT_TYPE	The account type that's the source of this raw contact. For example, the account type of a Google account is com.google. Always qualify your account type with a domain identifier for a domain you own or control. This will ensure that your account type is unique.

ContactsContract.Data Table

- Display name, phone number, email, postal address, photo, and website detail rows are all found in the ContactsContract.Data table.
- To help manage this, the ContactsContract.Data table has some columns with ***descriptive names***, and others with ***generic names***.

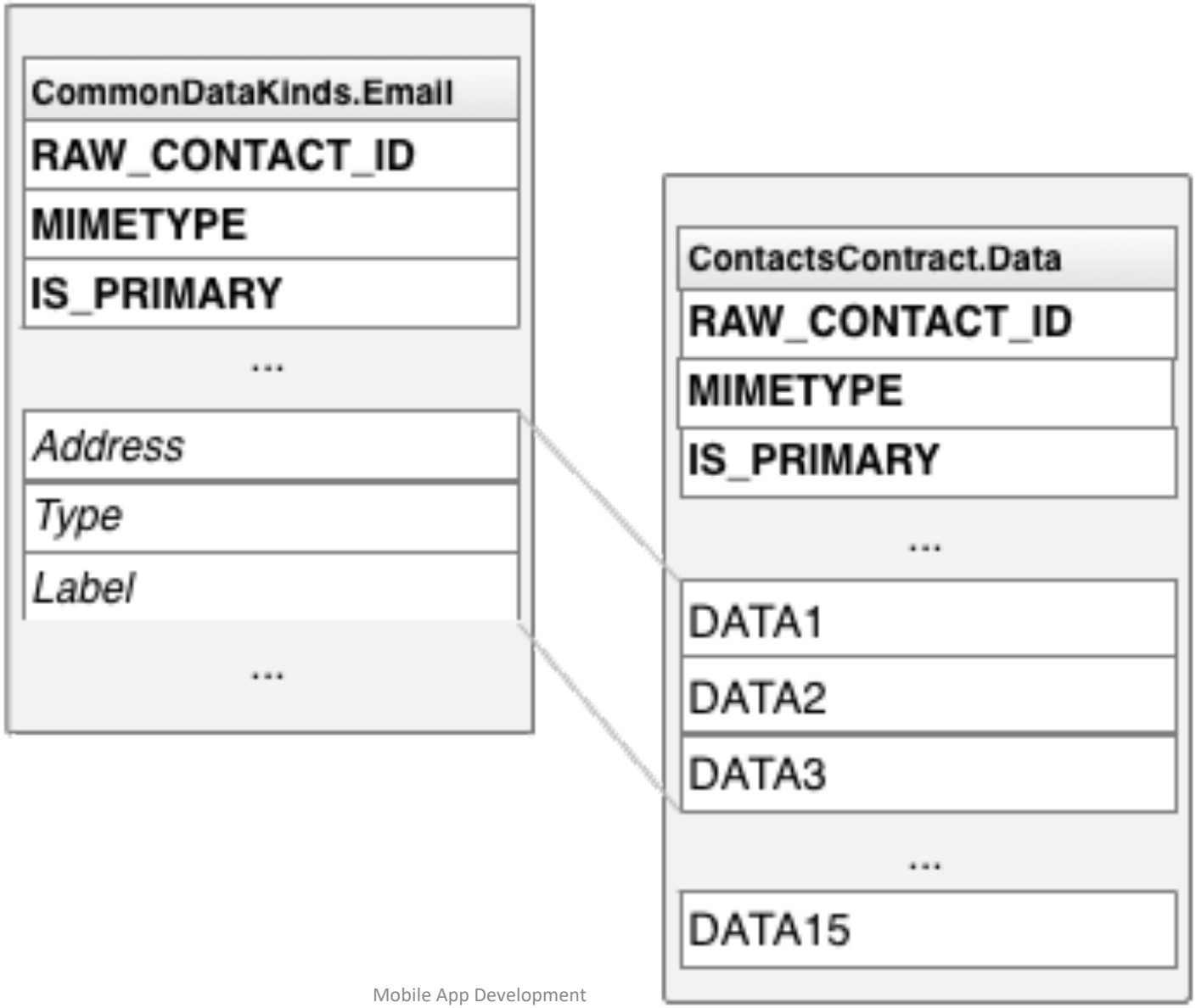
ContactsContract.Data Table (Descriptive column names)

- **RAW_CONTACT_ID** The value of the _ID column of the raw contact for this data.
- **MIMETYPE** The type of data stored in this row, expressed as a custom ***MIME type***.
- **IS_PRIMARY** column flags the data row that contains the primary data for the type.

ContactsContract.Data Table (Generic column names)

- There are 15 ***generic columns*** named DATA1 through DATA15 that are generally available
- Four generic columns SYNC1 through SYNC4 that should only be used by sync adapters.
- The DATA1 column is indexed. The Contacts Provider always uses this column for the data that the provider expects will be the most frequent target of a query. For example, in an email row, this column contains the actual email address.
- By convention, the column DATA15 is reserved for storing Binary Large Object (BLOB) data such as photo thumbnails.

ContactsContract.Data Table (Generic column names)



ContactsContract.Data Table (Generic column names)

RAW_CONTACT_ID	MIMETYPE	IS_PRIMARY	DATA1	DATA2	...	DATA 15
1	EMAIL	0	dickinson@gmail.com	TYPE_WORK		
1	PHONE	0	+84909999999	TYPE_HOME		
1	PHONE	1	+84988888888	TYPE_MOBILE		
…	…	…	…	…	…	…

ContactsContract.Data Table (MIMETYPEs)

MINETYPE	STATIC VARIABLE ALIAS	STRING VALUES
CONTACT'S NAME	ContactsContract.CommonDataKinds.StructuredName.CONTENT_ITEM_TYPE	vnd.android.cursor.item/name
CONTACT'S PHONE NUMBER	ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE	vnd.android.cursor.item/phone_v2
CONTACT'S EMAIL	ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE	vnd.android.cursor.item/email_v2
CONTACT'S IM	ContactsContract.CommonDataKinds.Im.CONTENT_ITEM_TYPE	vnd.android.cursor.item/im
CONTACT'S RELATION	ContactsContract.CommonDataKinds.Relation.CONTENT_ITEM_TYPE	vnd.android.cursor.item/relation

ContactsContract.Data Table (Generic column names VS CommonDataKinds)

- ContactsContract.CommonDataKinds.StructuredName
- ContactsContract.CommonDataKinds.Phone
- ContactsContract.CommonDataKinds.Email
- ContactsContract.CommonDataKinds.Im
- ContactsContract.CommonDataKinds.Relation

ContactsContract.Data Table - CommonDataKinds (ContactsContract.CommonDataKinds.StructuredName)

Type	Alias	Data column	Notes
String	DISPLAY_NAME	DATA1	
String	GIVEN_NAME	DATA2	
String	FAMILY_NAME	DATA3	
String	PREFIX	DATA4	Common prefixes in English names are "Mr", "Ms", "Dr" etc.
String	MIDDLE_NAME	DATA5	
String	SUFFIX	DATA6	Common suffixes in English names are "Sr", "Jr", "III" etc.
String	PHONETIC_GIVEN_NAME	DATA7	Used for phonetic spelling of the name, e.g. Pinyin, Katakana, Hiragana
...	...	...	...

ContactsContract.Data Table - CommonDataKinds (ContactsContract.CommonDataKinds.Phone)

Type	Alias	Data column	Notes
String	NUMBER	DATA1	
int	TYPE	DATA2	**Allowed values are:** **TYPE_CUSTOM. Put the actual type in LABEL.** **TYPE_HOME, TYPE_MOBILE** **TYPE_WORK, TYPE_FAX_WORK** **TYPE_FAX_HOME, TYPE_PAGER** **TYPE_OTHER, TYPE_CALLBACK** **TYPE_CAR, TYPE_COMPANY_MAIN** **TYPE_ISDN, TYPE_MAIN** **TYPE_OTHER_FAX, TYPE_RADIO** **TYPE_TELEX, TYPE_TTY_TDD** **TYPE_WORK_MOBILE, TYPE_WORK_PAGER** **TYPE_ASSISTANT, TYPE_MMS**
String	LABEL	DATA3	

ContactsContract.Data Table - CommonDataKinds (ContactsContract.CommonDataKinds.Email)

Type	Alias	Data column	Notes
String	ADDRESS	DATA1	
int	TYPE	DATA2	**Allowed values are:** **TYPE_CUSTOM. Put the actual type in LABEL.** **TYPE_HOME** **TYPE_WORK** **TYPE_OTHER** **TYPE_MOBILE**
String	LABEL	DATA3	

ContactsContract.Data Table - CommonDataKinds (ContactsContract.CommonDataKinds.Im)

Type	Alias	Data column	Notes
String	DATA	DATA1	
int	TYPE	DATA2	**Allowed values are:** TYPE_CUSTOM. Put the actual type in LABEL. TYPE_HOME TYPE_WORK TYPE_OTHER
String	LABEL	DATA3	
String	PROTOCOL	DATA5	**Allowed values:** PROTOCOL_CUSTOM. PROTOCOL_AIM, PROTOCOL_MSN, PROTOCOL_YAHOO, PROTOCOL_SKYPE PROTOCOL_QQ, PROTOCOL_GOOGLE_TALK PROTOCOL_ICQ, PROTOCOL_JABBER, PROTOCOL_NETMEETING
String	CUSTOM_PROTOCOL	DATA6	

ContactsContract.Data Table - CommonDataKinds (ContactsContract.CommonDataKinds.Relation)

Type	Alias	Data column	Notes
String	DATA	DATA1	
int	TYPE	DATA2	**Allowed values are:** **TYPE_CUSTOM. Put the actual type in LABEL.** **TYPE_ASSISTANT, TYPE_BROTHER** **TYPE_CHILD, TYPE_DOMESTIC_PARTNER** **TYPE_FATHER, TYPE_FRIEND** **TYPE_MANAGER, TYPE_MOTHER** **TYPE_PARENT, TYPE_PARTNER** **TYPE_REFERRED_BY, TYPE_RELATIVE** **TYPE_SISTER, TYPE_SPOUSE**
String	LABEL	DATA3	

Outline

- Introduction
  - ContentProvider
  - Common Providers
  - Content URIs
- Common Content Providers
  - Contact Providers
  - CallLog Provider
  - SMS Provider
- **ContentResolver**
  - Retrieving Data from the Provider
  - Inserting, Updating, and Deleting Data
- Demonstration

ContentResolver

- When you want to access data in a content provider, you use the **ContentResolver** object in your application's Context to communicate with the provider as a client.
  - The API you use retrieve data from a content provider.
  - The API you use to insert, update, or delete data in a content provider.
- The **ContentResolver** object communicates with the provider object, an instance of a class that implements ContentProvider.
- The provider object receives data requests from clients, performs the requested action, and returns the results.

ContentResolver (Content Provider Permissions)

- To get the permissions needed to access a provider, an application requests them with a ***<uses-permission>*** element in its manifest file.
- When the Android Package Manager installs the application, a user must approve all of the permissions the application requests.
- The following **<uses-permission>** element requests read access to the User Dictionary Provider:

```
<uses-permission android:name="android.permission.READ_USER_DICTIONARY">
```

Outline

- Introduction
  - ContentProvider
  - Common Providers
  - Content URIs
- Common Content Providers
  - Contact Providers
  - CallLog Provider
  - SMS Provider
- ContentResolver
  - **Retrieving Data from the Provider**
  - Inserting, Updating, and Deleting Data
- Demonstration

ContentResolver
(Retrieving Data from the Provider)

```
mCursor = getContentResolver().query(
                                CONTENT_URI,
                                projection,
                                selectionClause
                                selectionArgs,
                                sortOrder
);
```

ContentResolver (Retrieving Data from the Provider)

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

```
ContentResolver cr = getContentResolver();
Cursor rs = cr.query(
        ContactsContract.Contacts.CONTENT_URI,
        null,
        null,
        null,
        null);
```

```
int colIndex = rs.getColumnIndex(
        ContactsContract.Contacts.DISPLAY_NAME);
rs.moveToFirst();
do {
    String contactName = rs.getString(colIndex);
    Log.d("ContactDemonstration", contactName);
}while (rs.moveToNext());
rs.close();
```

```
ContentResolver cr = getContentResolver();
Cursor rs = cr.query(
        ContactsContract.Contacts.CONTENT_URI,
        null,
        null,
        null,
        null);
```

```
int colIndex = rs.getColumnIndex(
        ContactsContract.CommonDataKinds.Phone.NUMBER);
rs.moveToFirst();
do {
    String phoneNumber = rs.getString(colIndex);
    Log.d("ContactDemonstration", phoneNumber);
}while (rs.moveToNext());
rs.close();
```

(Using ContactsContract.Data)

```
ContentResolver cr = getContentResolver();
String selection = ContactsContract.Data.MIMETYPE + "=?";
String[] selectionArgs = {
        ContactsContract.CommonDataKinds.Phone.CONTENT_ITEM_TYPE
};
Cursor rs = cr.query(
        ContactsContract.Data.CONTENT_URI,
        null,
        selection,
        selectionArgs,
        null);
int colIndex = rs.getColumnIndex(ContactsContract.Data.DATA1);
rs.moveToFirst();
do {
    String phoneNumber = rs.getString(colIndex);
    Log.d("ContactDemonstration", phoneNumber);
}while (rs.moveToNext());
rs.close();
```

(Using CommonDataKinds.Phone)

```
ContentResolver cr = getContentResolver();
Cursor rs = cr.query(
        ContactsContract.CommonDataKinds.Phone.CONTENT_URI,
        null,
        null,
        null,
        null);
int colIndex = rs.getColumnIndex(
        ContactsContract.CommonDataKinds.Phone.NUMBER);
rs.moveToFirst();
do {
    String phoneNumber = rs.getString(colIndex);
    Log.d("ContactDemonstration", phoneNumber);
}while (rs.moveToNext());
rs.close();
```

(using ContactsContract.Data)

```
ContentResolver cr = getContentResolver();
String selection = ContactsContract.Data.MIMETYPE + "=?";
String[] selectionArgs = {
        ContactsContract.CommonDataKinds.Email.CONTENT_ITEM_TYPE
};
Cursor rs = cr.query(
        ContactsContract.Data.CONTENT_URI,
        null,
        selection,
        selectionArgs,
        null);
int colIndex = rs.getColumnIndex(ContactsContract.Data.DATA1);
rs.moveToFirst();
do {
    String email = rs.getString(colIndex);
    Log.d("ContactDemonstration", email);
}while (rs.moveToNext());
rs.close();
```

(using CommonDataKinds.Email)

```
ContentResolver cr = getContentResolver();
Cursor rs = cr.query(
        ContactsContract.CommonDataKinds.Email.CONTENT_URI,
        null,
        null,
        null,
        null);
int colIndex = rs.getColumnIndex(
        ContactsContract.CommonDataKinds.Email.ADDRESS);
rs.moveToFirst();
do {
    String email = rs.getString(colIndex);
    Log.d("ContactDemonstration", email);
}while (rs.moveToNext());
rs.close();
```

Retrieve call logs

```
Cursor cursor = getContentResolver().query(
        CallLog.Calls.CONTENT_URI,
        null,
        null,
        null,
        null);

int colIndex = cursor.getColumnIndex(
        CallLog.Calls.NUMBER);
cursor.moveToFirst();
do {
    String number = cursor.getString(colIndex);
    Log.d("CallLogDemonstration", number);
}while (cursor.moveToNext());
```

Outline

- Introduction
  - ContentProvider
  - Common Providers
  - Content URIs
- Common Content Providers
  - Contact Providers
  - CallLog Provider
  - SMS Provider
- ContentResolver
  - Retrieving Data from the Provider
  - **Inserting, Updating, and Deleting Data**
- Demonstration

ContentResolver (Insert Data)

```
getContentResolver(). insert(
                        CONTENT_URI,
                        contentValues,
);
```

ContentResolver (Insert Data)

```
getContentResolver(). update(
                        CONTENT_URI,
                        contentValues,
                        where,
                        selectionArgs
);
```

ContentResolver (Insert Data)

```
getContentResolver(). delete(
                        CONTENT_URI,
                        where,
                        selectionArgs
);
```

```
ContentValues call = new ContentValues();
call.put(CallLog.Calls.TYPE,
        CallLog.Calls.INCOMING_TYPE);
call.put(CallLog.Calls.NUMBER,
        "18001090");
call.put(CallLog.Calls.DATE,
        Calendar.getInstance().getTime().getTime());
getContentResolver().insert(
        CallLog.Calls.CONTENT_URI,
        call);
```

```
ContentValues missedCall = new ContentValues();
missedCall.put(CallLog.Calls.TYPE,
        CallLog.Calls.MISSED_TYPE);
missedCall.put(CallLog.Calls.NUMBER,
        "77777777");
Calendar c = Calendar.getInstance();
missedCall.put(CallLog.Calls.DATE,
        c.getTime().toString());
getContentResolver().insert(
        CallLog.Calls.CONTENT_URI,
        missedCall);
```

```
ContentValues sms = new ContentValues();
sms.put(Telephony.Sms.BODY,
        "automatic message");
sms.put(Telephony.Sms.ADDRESS,
        "090000000");
sms.put(Telephony.Sms.DATE,
        Calendar.getInstance().getTime().toString());
getContentResolver().insert(
        Telephony.Sms.Sent.CONTENT_URI,
        sms);
```

ContentObserver

Public methods	
boolean	**deliverSelfNotifications()** Returns true if this observer is interested receiving self-change notifications.
final void	**dispatchChange(boolean selfChange)** *This method was deprecated in API level 16. Use* **dispatchChange(boolean, android.net.Uri)** *instead.*
final void	**dispatchChange(boolean selfChange, Uri uri)** Dispatches a change notification to the observer.
void	**onChange(boolean selfChange)** This method is called when a content change occurs.
void	**onChange(boolean selfChange, Uri uri)** This method is called when a content change occurs.

ContentResolver (Batch Processing)

```
getContentResolver().applyBatch(
      AUTHORITY,
      ArrayList<ContentProviderOperation>
);
```

ContentResolver (Batch Processing)

```
ArrayList<ContentProviderOperation> ops =
        new ArrayList<ContentProviderOperation>();

ContentProviderOperation op1 =
        ContentProviderOperation.newInsert(CONTENT_URI)
                .withValue (key1, value1)
                .....
                .withValue(keyN, valueN)
                .build();

ops.add(op1)
```

ContentResolver (Batch Processing)

```
ContentProviderOperation op2 =
        ContentProviderOperation.newInsert(CONTENT_URI)
                .withValue (key1, value1)
                                        .....
                .withValue(keyN, valueN)
                .build();

ops.add(op2);

getContentResolver().applyBatch(
        AUTHORITY,
        ops);
```

ContentResolver (Batch Processing)

```
ContentProviderOperation op3 =
        ContentProviderOperation.newInsert(CONTENT_URI)
                .withValueBackReference(key1, 0)
                .withValueBackReference(key2, 1)
                .withValue(key2, value2)
                                        .....
                .withValue(keyN, valueN)
                .build();
```

```
.withValueBackReference(key, indexValue)
```

ContentResolver

```
ArrayList<ContentProviderOperation> ops =
                new ArrayList<ContentProviderOperation>();

ops.add(ContentProviderOperation.newInsert (
                        ContactsContract.RawContacts.CONTENT_URI)
        .withValue(ContactsContract.RawContacts.ACCOUNT_TYPE, null)
        .withValue(ContactsContract.RawContacts.ACCOUNT_NAME,null )
        .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,
                "0912345678")
        .build());
```

ContentResolver

```
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,
                "Mike Wave")
        .build());
try {
    ContentProviderResult[] res = getContentResolver().applyBatch(
                                    ContactsContract.AUTHORITY,
                                    ops);
} catch (RemoteException e) {
    e.printStackTrace();
} catch (OperationApplicationException e) {
    e.printStackTrace();
}
```

Bottom line

- What's content provider?
- How is the structure of ContactsContract content provider?
- How to make a fake/missed call?
- How to make a fake message?

Exercise 1

- Show the entire SMS sent by phone number 0909999999
- Show the entire SMS which contains string “Hello, how are you?”
- Show the time of all calls to contacts name “Donald Trump”

https://inthecheesefactory.com/blog/things-you-need-to-know-about-android-m-permission-developer-edition/en

Exercise 2

- Show the entire contacts with the friend‘s relationship
- Show the entire contacts who has at least one yahoo account
- Show the entire contacts with phone number is 098888888 and display name is “Obama”

https://developer.android.com/reference/android/provider/ContactsContract.PhoneLookup.html

Exercise 3

- Use **ContentResolver**
  - Show the entire contacts with the friend‘s relationship
  - Show the entire contacts who has at least one yahoo account
  - Show the entire contacts with phone number is 098888888 and display name is “Obama”
  - Show the entire SMS sent by phone number 0908888888
  - Show the entire SMS which contains string “I love you”