

# Data Storage in Android



Lecturer: Trần Hồng Nghi  
Email: [ngarith@uit.edu.vn](mailto:ngarith@uit.edu.vn)

# Android Files

- Android uses the same file constructions found in a typical Java application.
- Files can be stored in the device's (small) main memory or in the much larger SD card. They can also be obtained from the network (as we will see later).
- Files stored in the device's memory, stay together with other application's resources (such as icons, pictures, music, ...).
- We will call this type: Resource Files.

# Storage Options

- Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.
- Your data storage options are the following:
  - Shared Preferences: Store private primitive data in key-value pairs.
  - Internal Storage: Store private data on the device memory.
  - External Storage: Store public data on the shared external storage.
  - SQLite Databases: Store structured data in a private database.
  - Network Connection: Store data on the web with your own network server.

# Shared Preferences

- Preferences is an Android lightweight mechanism to store and retrieve <key-value> pairs of primitive data types: booleans, floats, ints, longs, and strings
- PREFERENCES are typically used to keep state information and shared data among several activities of an application.
- In each entry of the form <key-value> the key is a string and the value must be a primitive data type.
- Preferences are similar to Bundles however they are persistent while Bundles are not.

# Shared Preferences(cont.)

- To get access to the preferences, you can use the following APIs:
  - `getPreferences()` from within your Activity, to access activityspecific preferences
  - `getSharedPreferences()` from within your Activity (or other application Context), to access application-level preferences
  - `getDefaultSharedPreferences()`, on `PreferencesManager`, to get the shared preferences that work in concert with Android's overall preference framework
- All of these methods return an instance of `SharedPreferences`, which offers a series of getters to access named preferences, returning a suitably typed result (e.g., `getBoolean()` to return a Boolean preference).

# Preference access permissions

- You can open and create SharedPreferences with any combination of several Context mode constants. Because these values are int types, you can add them, to combine permissions. The following mode constants are supported:
  - Context.MODE\_PRIVATE (value 0)
  - Context.MODE\_WORLD\_READABLE (value 1)
  - Context.MODE\_WORLD\_WRITEABLE (value 2)

# Using Preferences API calls(cont.)

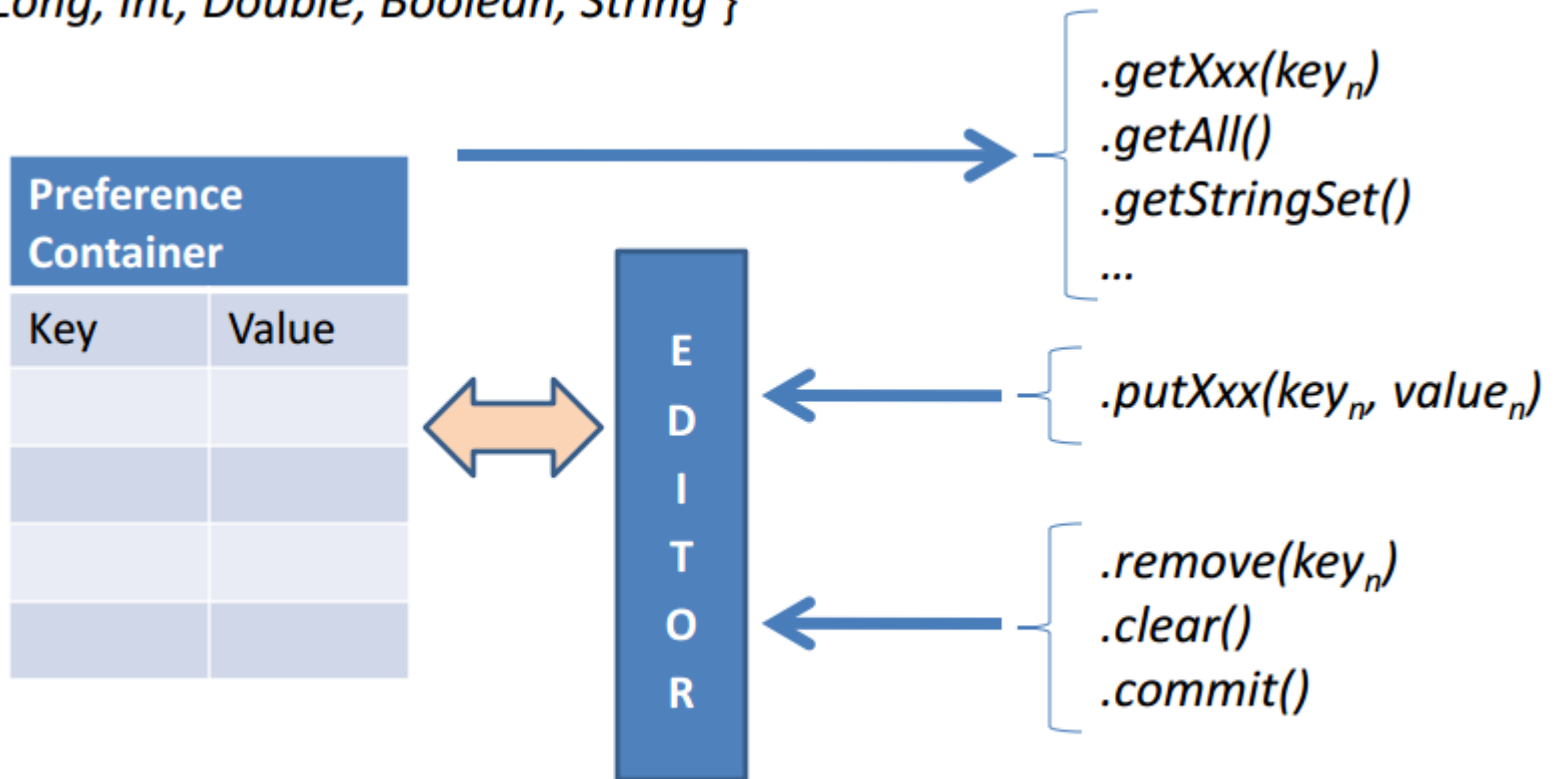
- To write values:
  1. Call edit() to get a SharedPreferences.Editor.
  2. Add values with methods such as putBoolean() and putString().
  3. Commit the new values with commit()
- To read values, use SharedPreferences methods such as getBoolean() and getString().

```
//get the preferences, then editor, set a data item
SharedPreferences appPrefs =
getSharedPreferences("MyAppPrefs", 0);
SharedPreferences.Editor prefsEd = appPrefs.edit();
prefsEd.putString("dataString", "some string data");
prefsEd.commit();
```

```
//get the preferences then retrieve saved
data, specifying a default value
SharedPreferences appPrefs =
getSharedPreferences("MyAppPrefs", 0);
String savedData =
appPrefs.getString("dataString", "");
```

# Using Preferences API calls(cont.)

$Xxx = \{ Long, Int, Double, Boolean, String \}$

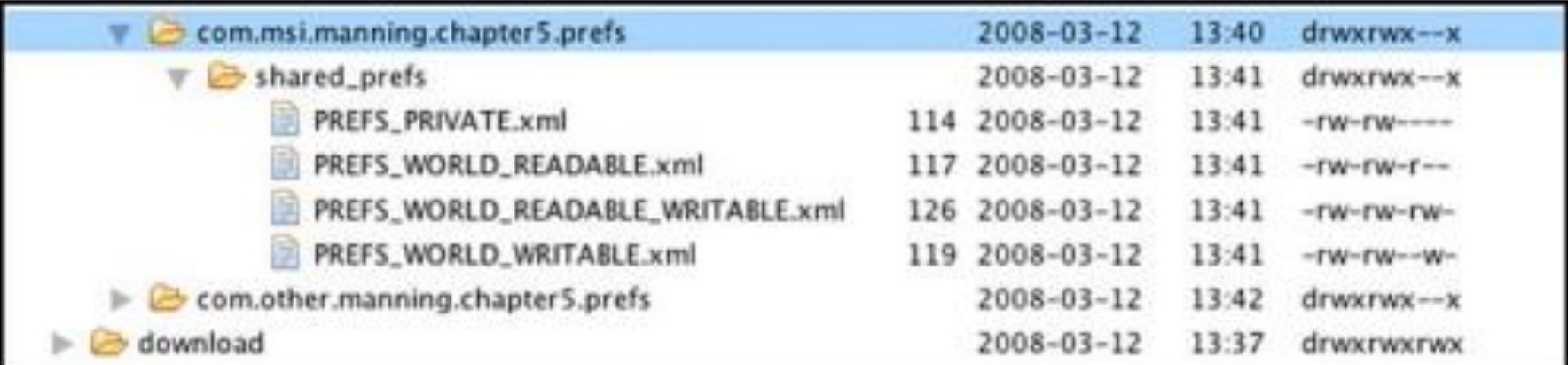




# Path of Preferences xml file.

- Android puts SharedPreferences XML files in the  
/data/data/YOUR\_PACKAGE\_NAME/shared\_prefs/YOUR\_P  
REFS\_NAME.xml
- OR  
/data/data/YOUR\_PACKAGE\_NAME/shared\_prefs/YOUR\_P  
ACKAGE\_NAME\_preferences.xml

An application or package usually has its own:

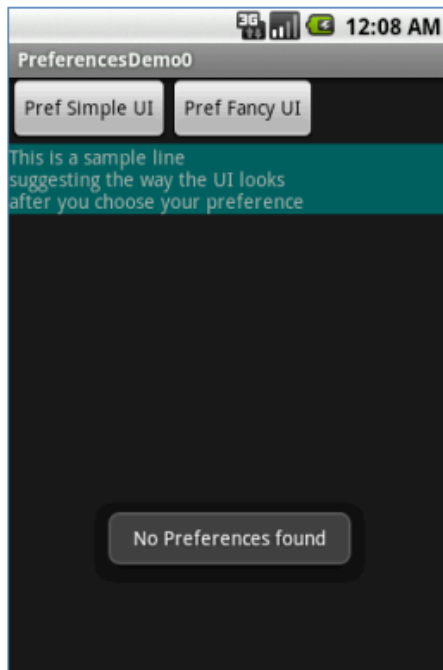


The screenshot shows a file manager interface with a list of files and folders. The first folder is 'com.msi.manning.chapter5.prefs', which contains a sub-folder 'shared\_prefs'. Inside 'shared\_prefs', there are four XML files: 'PREFS\_PRIVATE.xml', 'PREFS\_WORLD\_READABLE.xml', 'PREFS\_WORLD\_READABLE\_WRITABLE.xml', and 'PREFS\_WORLD\_WRITABLE.xml'. Below these, there is another folder 'com.other.manning.chapter5.prefs' and a 'download' folder. Each entry in the list is accompanied by its size, date, time, and permissions.

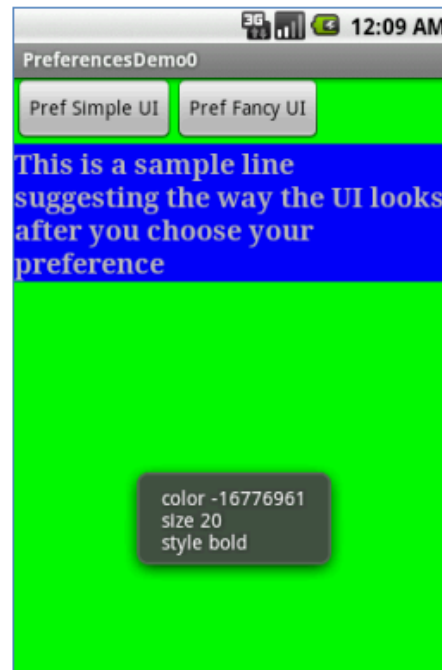
Item	Size	Date	Time	Permissions
com.msi.manning.chapter5.prefs		2008-03-12	13:40	drwxrwx--x
shared_prefs		2008-03-12	13:41	drwxrwx--x
PREFS_PRIVATE.xml	114	2008-03-12	13:41	-rw-rw----
PREFS_WORLD_READABLE.xml	117	2008-03-12	13:41	-rw-rw-r--
PREFS_WORLD_READABLE_WRITABLE.xml	126	2008-03-12	13:41	-rw-rw-rw-
PREFS_WORLD_WRITABLE.xml	119	2008-03-12	13:41	-rw-rw--w-
com.other.manning.chapter5.prefs		2008-03-12	13:42	drwxrwx--x
download		2008-03-12	13:37	drwxrwxrwx

# Example

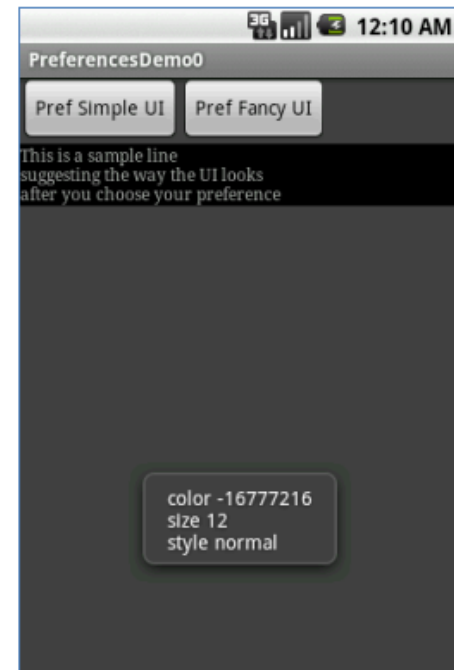
- Saving/Retrieving a SharedPreferences Object holding UI user choices.



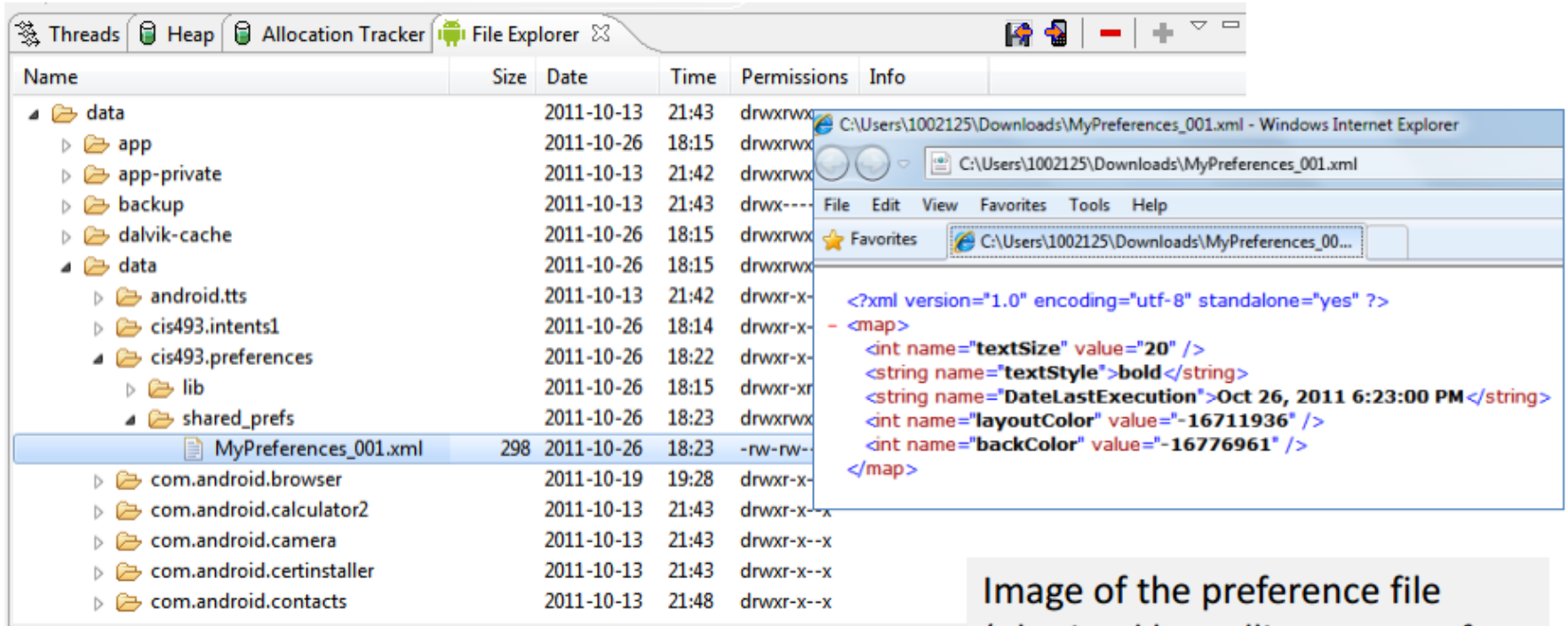
Initial UI with no choices made/save yet.



Images of the choices made by the user regarding the looks of the UI. The 'green screen' corresponds to the fancy layout, the 'grey screen' is the simple choice. Data is saved into the SharedPreferences object: *myPreferences 001*.



# Example(cont.)



The screenshot displays the DDMS File Explorer interface. The file list shows the following structure:

Name	Size	Date	Time	Permissions	Info
data		2011-10-13	21:43	drwxrwx	
app		2011-10-26	18:15	drwxrwx	
app-private		2011-10-13	21:42	drwxrwx	
backup		2011-10-13	21:43	drwx---	
dalvik-cache		2011-10-26	18:15	drwxrwx	
data		2011-10-26	18:15	drwxrwx	
android.tts		2011-10-13	21:42	drwxr-x	
cis493.intents1		2011-10-26	18:14	drwxr-x	
cis493.preferences		2011-10-26	18:22	drwxr-x	
lib		2011-10-26	18:15	drwxr-xr	
shared_prefs		2011-10-26	18:23	drwxrwx	
MyPreferences_001.xml	298	2011-10-26	18:23	-rw-rw-	
com.android.browser		2011-10-19	19:28	drwxr-x	
com.android.calculator2		2011-10-13	21:43	drwxr-x--x	
com.android.camera		2011-10-13	21:43	drwxr-x--x	
com.android.certinstaller		2011-10-13	21:43	drwxr-x--x	
com.android.contacts		2011-10-13	21:48	drwxr-x--x	

The inset window shows the XML content of 'MyPreferences\_001.xml':

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<map>
  <int name="textSize" value="20" />
  <string name="textStyle">bold</string>
  <string name="DateLastExecution">Oct 26, 2011 6:23:00 PM</string>
  <int name="layoutColor" value="-16711936" />
  <int name="backColor" value="-16776961" />
</map>
```

Image of the preference file  
(obtained by pulling a copy of  
the file out of the device).

- Using DDMS to explore the Device's memory map. Observe the choices made by the user are saved in the **data/data/Shared\_prefs/** folder as an XML file.

# File xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    android:id="@+id/linLayout1Vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android" >

    <LinearLayout
        android:id="@+id/linLayout2Horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" >
        <Button
            android:id="@+id/btnPrefSimple"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Pref Simple UI" />
        <Button
            android:id="@+id/btnPrefFancy"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Pref Fancy UI" />
    </LinearLayout>

    <TextView
        android:id="@+id/txtCaption1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="#ff006666"
        android:text="This is some sample text " />

</LinearLayout>
```

# Example(cont.)

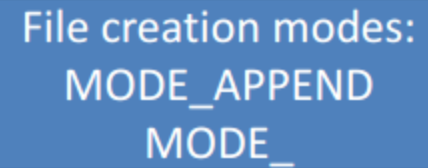
```
import ...

public class PreferenceDemo0 extends Activity implements OnClickListener {
    Button btnSimplePref;
    Button btnFancyPref;
    TextView txtCaption1;
    Boolean fancyPrefChosen = false;
    View    myLayout1Vertical;

    final int mode = Activity.MODE_PRIVATE;
    final String MYPREFS = "MyPreferences_001";

    // create a reference to the shared preferences object
    SharedPreferences mySharedPreferences;

    // obtain an editor to add data to my SharedPreferences object
    SharedPreferences.Editor myEditor;
```



File creation modes:  
MODE\_APPEND  
MODE\_

# Example(cont.)

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    myLayout1Vertical = (View)findViewById(R.id.linLayout1Vertical);
    txtCaption1 = (TextView) findViewById(R.id.txtCaption1);
    txtCaption1.setText("This is a sample line \n"
        + "suggesting the way the UI looks \n"
        + "after you choose your preference");

    // create a reference & editor for the shared preferences object
    mySharedPreferences = getSharedPreferences(MYPREFS, 0);
    myEditor = mySharedPreferences.edit();
    // has a Preferences file been already created?
    if (mySharedPreferences != null
        && mySharedPreferences.contains("backColor")) {
        // object and key found, show all saved values
        applySavedPreferences();
    } else {
        Toast.makeText(getApplicationContext(),
            "No Preferences found", 1).show();
    }
    btnSimplePref = (Button) findViewById(R.id.btnPrefSimple);
    btnSimplePref.setOnClickListener(this);
    btnFancyPref = (Button) findViewById(R.id.btnPrefFancy);
    btnFancyPref.setOnClickListener(this);
} // onCreate
```

# Example(cont.)

```
@Override
public void onClick(View v) {
    // clear all previous selections
    myEditor.clear();

    // what button has been clicked?
    if (v.getId() == btnSimplePref.getId()) {
        myEditor.putInt("backColor", Color.BLACK); // black background
        myEditor.putInt("textSize", 12); // humble small font
    } else { // case btnFancyPref
        myEditor.putInt("backColor", Color.BLUE); // fancy blue
        myEditor.putInt("textSize", 20); // fancy big
        myEditor.putString("textStyle", "bold"); // fancy bold
        myEditor.putInt("layoutColor", Color.GREEN); // fancy green
    }
    myEditor.commit();
    applySavedPreferences();
}
```

```
@Override
protected void onPause() {
    // warning: activity is on its last state of visibility!.
    // It's on the edge of being killed! Better save all current
    // state data into Preference object (be quick!)
    myEditor.putString("DateLastExecution", new Date().toLocaleString());
    myEditor.commit();
    super.onPause();
}
```



# Example(cont.)

```
public void applySavedPreferences() {
    // extract the <key/value> pairs, use default param for missing data
    int backColor = mySharedPreferences.getInt("backColor",Color.BLACK);
    int textSize = mySharedPreferences.getInt("textSize", 12);
    String textStyle = mySharedPreferences.getString("textStyle", "normal");
    int layoutColor = mySharedPreferences.getInt("layoutColor",Color.DKGRAY);
    String msg = "color " + backColor + "\n"
                + "size " + textSize + "\n"
                + "style " + textStyle;
    Toast.makeText(getApplicationContext(), msg, 1).show();

    txtCaption1.setBackgroundColor(backColor);
    txtCaption1.setTextSize(textSize);
    if (textStyle.compareTo("normal")==0) {
        txtCaption1.setTypeface(Typeface.SERIF,Typeface.NORMAL);
    }
    else {
        txtCaption1.setTypeface(Typeface.SERIF,Typeface.BOLD);
    }
    myLayout1Vertical.setBackgroundColor(layoutColor);
} // applySavedPreferences

} // class
```



# Internal Storage:

- You can save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed.
- Android provides a convenience method on Context to get a `FileOutputStream`— namely `openFileOutput(String name, int mode)` create and write a private file to the internal storage:
  1. Call `openFileOutput()` with the name of the file and the operating mode. This returns a `FileOutputStream`. That file will ultimately be stored at the `data/data/[PACKAGE_NAME]/files/file.name` path on the platform.
  2. Write to the file with `write()`.
  3. Close the stream with `close()`.

# MODE\_PRIVATE

- MODE\_PRIVATE will create the file (or replace a file of the same name) and make it private to your application. Other modes available are: MODE\_APPEND, MODE\_WORLD\_READABLE, MODE\_WORLD\_WRITEABLE.
- **Tip:** If you want to save a static file in your application at compile time, save the file in your project res/raw/directory. You can open it with openRawResource(), passing the R.raw.<filename> resource ID. This method returns an InputStream that you can use to read the file (but you cannot write to the original file).

# Saving cache file

- If you'd like to cache some data, rather than store it persistently, you should use `getCacheDir()` to open a `File` that represents the internal directory where your application should save temporary cache files.
- When the device is low on internal storage space, Android may delete these cache files to recover space. However, you should not rely on the system to clean up these files for you. You should always maintain the cache files yourself and stay within a reasonable limit of space consumed, such as 1M.
- When the user uninstalls your application, these files are removed.

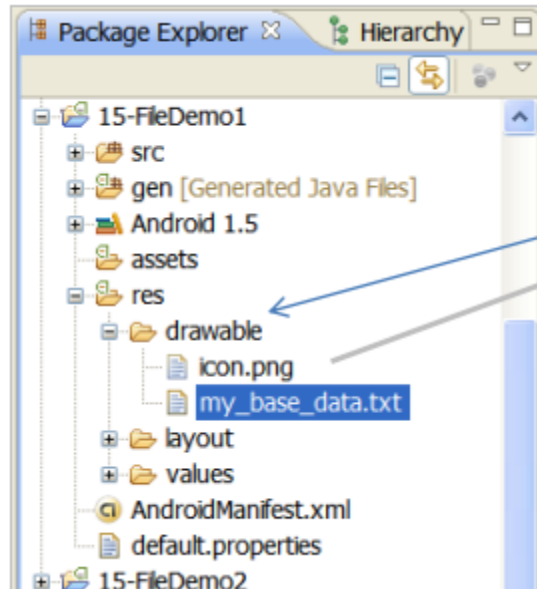
# Other useful methods

- getFilesDir() Gets the absolute path to the filesystem directory where your internal files are saved.
- getDir() Creates (or opens an existing) directory within your internal storage space.
- deleteFile() Deletes a file saved on the internal storage.
- fileList() Returns an array of files currently saved by your application.

# Using Android Resource Files

- When an application's .apk bytecode is deployed it may store in memory: code, drawables, and other raw resources (such as files). Acquiring those resources could be done using a statement such as:

```
InputStream is = this.getResources()  
                .openRawResource(R.drawable.my_base_data);
```



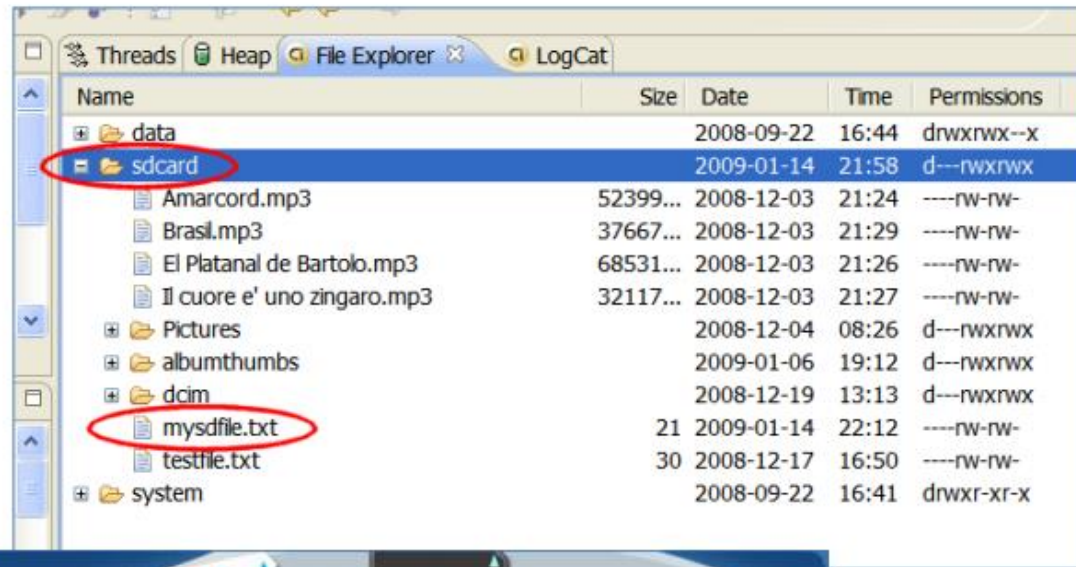
Use drag/drop to place file **my\_base\_data.txt** in res folder. It will be stored in the device's memory as part of the .apk

# External Storage

- Every Android-compatible device supports a shared "external storage" that you can use to save files. This can be a removable storage media (such as an SD card) or an internal (non-removable) storage.
- Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.
- Before you do any work with the external storage, you should always call `getExternalStorageState()` to check whether the media is available. The media might be mounted to a computer, missing, read-only, or in some other state.
- **Caution:** External storage can become unavailable if the user mounts the external storage on a computer or removes the media, and there's no security enforced upon files you save to the external storage. All applications can read and write files placed on the external storage and the user can remove them.

# External Storage(cont.)

Storing data into the SD card has the obvious advantage of a larger working space.



Name	Size	Date	Time	Permissions
data		2008-09-22	16:44	drwxrwx--x
sdcard		2009-01-14	21:58	d--rwxrwx
Amarcord.mp3	52399...	2008-12-03	21:24	----rw-rw-
Brasil.mp3	37667...	2008-12-03	21:29	----rw-rw-
El Platanal de Bartolo.mp3	68531...	2008-12-03	21:26	----rw-rw-
Il cuore e' uno zingaro.mp3	32117...	2008-12-03	21:27	----rw-rw-
Pictures		2008-12-04	08:26	d--rwxrwx
albumthumbs		2009-01-06	19:12	d--rwxrwx
dcim		2008-12-19	13:13	d--rwxrwx
mysdfile.txt	21	2009-01-14	22:12	----rw-rw-
testfile.txt	30	2008-12-17	16:50	----rw-rw-
system		2008-09-22	16:41	drwxr-xr-x



SDK1.6 it is necessary to request permission to write to the SD card. Add the following clause to your AndroidManifest.xml

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

# Example: Checking media availability

```
boolean mExternalStorageAvailable = false;
boolean mExternalStorageWriteable = false;
String state = Environment.getExternalStorageState();

if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // We can only read the media
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    // Something else is wrong. It may be one of many other states, but all we need
    // to know is we can neither read nor write
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}
```



# Example: Create file on External Storage

```
void createExternalStorageFile(Context context) {  
    //tao file moi 91  
    File file = new File(context.getExternalFilesDir(null), "DemoFile.txt ");  
    try {  
        file.createNewFile();  
        FileWriter fw = new FileWriter(file);  
        BufferedWriter bw = new BufferedWriter(fw);  
        bw.write("Hello World");  
        bw.newLine();  
        bw.close();  
    } catch (IOException e) {  
        Log.w("ExternalStorage", "Error writing " + file, e);  
    }  
}
```

# Accessing files on external storage

- Using API Level 8 or greater, use getExternalFilesDir() to open a File that represents the external storage directory .
- This method takes a type parameter that specifies the type of subdirectory you want, such as:
  - DIRECTORY\_MUSIC
  - DIRECTORY\_RINGTONES

This method will create the appropriate directory if necessary. By specifying the type of directory, you ensure that the Android's media scanner will properly categorize your files in the system (for example, ringtones are identified as ringtones and not music).

- Using API Level 7 or lower, use getExternalStorageDirectory(), to open a File representing the root of the external storage. You should then write your data in the following directory:

`/Android/data/ <package_name>/files/`

The `<package_name>` is your Java-style package name, such as "com.example.android.app".

- If the user uninstalls your application, this directory and all its contents will be deleted.

# Saving files that should be shared

- If you want to save files that are not specific to your application and that should *not* be deleted when your application is uninstalled, save them to one of the public directories on the external storage. These directories lay at the root of the external storage, such as Music/, Pictures/, Ringtones/, and others.
- In API Level 8 or greater, use [getExternalStoragePublicDirectory\(\)](#) else use [getExternalStorageDirectory\(\)](#) to open a [File](#).
- Save your shared files in one of the following directories:
  - Music/ - Media scanner classifies all media found here as user music.
  - Podcasts/ - Media scanner classifies all media found here as a podcast.
  - Ringtones/ - Media scanner classifies all media found here as a ringtone.
  - Alarms/ - Media scanner classifies all media found here as an alarm sound.
  - Notifications/ - Media scanner classifies all media found here as a notification sound.
  - Pictures/ - All photos (excluding those taken with the camera).
  - Movies/ - All movies (excluding those taken with the camcorder).
  - Download/ - Miscellaneous downloads.

# Saving cache files

- using API Level 8 or greater, use getExternalCacheDir() to open a File that represents the external storage directory where you should save cache files. If the user uninstalls your application, these files will be automatically deleted.
- If you're using API Level 7 or lower, use getExternalStorageDirectory() to open a File that represents the root of the external storage, then write your cache data in the following directory:
  - */Android/data/ <package\_name>/cache/*  
The *<package\_name>* is your Java-style package name, such as "com.example.android.app".

# Content Provider & SQLite

- Content Provider
- SQLite

# Content Provider

- Content providers allow programs access to data which is present on the device.
- A content provider manages access to a central repository of data.
- They encapsulate the data from tables, and provide mechanisms for defining data security and unified usage.
- Content providers are the standard interface that connects data in one process with code running in another process.
- Link Content Provider:  
<http://developer.android.com/reference/android/provider/package-summary.html>

# Content Providers available

Content Provider	Intended Data
Browser	Browser bookmarks, browser history, etc.
CallLog	Missed calls, call details, etc.
Contacts	Contact details
MediaStore	Media files such as audio, video and images
Settings	Device settings and preferences

# How they work

- Irrespective of how the data is stored, Content Providers give a uniform interface to access the data.
- Data is exposed as a simple table with rows and columns where row is a record and column is a particular data type with a specific meaning.
- Each record is identified by a unique **\_ID** field which is the key to the record.
- Each content provider exposes a unique **URI** that identifies its data set uniquely. URI == table-name




# Content Provider-Cursor

- Cursor is an interface that provides **random** read-write access to the result of a database query from a **content provider**
- A cursor is a collection of **rows**

ID	ISBN	Tên sách	NXB
1	123456	Lập trình Android	T3H
2	987654	Lập trình iPhone	T3H
3	134679	Lập trình Java	T3H1
4	258456	Lập trình C#	T3H2

Get books: T3H

Cursor



ID	ISBN	Title	NXB
1	123456	Lập trình Android	T3H
2	987654	Lập trình iPhone	T3H

# Method of Cursor:

- moveToFirst
- moveToNext
- moveToPrevious
- getCount
- getColumnIndexOrThrow
- getColumnName
- getColumnNames
- moveToPosition
- getPosition

# Content URI

- A **content URI** is a URI that identifies data in a provider.
  - `android.provider.Contacts.Phones.CONTENT_URI`
  - `android.provider.Contacts.Photos.CONTENT_URI`

# Content Provider

- ContentResolver

- *ContentResolver cr = getContentResolver();*

- To retrieve data from a provider, your application needs "read access permission" for the provider in androidmanifest.xml

- Cursor:

- *Cursor cursor = cr.query(...)*

- *Ex:* **Cursor** someRows = cr.**query**(MyProvider.CONTENT\_URI, null, where, null, order);

# Content Provider

```
ContentResolver cr = getContentResolver();
```

```
// Trả về các dòng
```

```
Cursor allRows = cr.query(MyProvider.CONTENT_URI, null, null, null,  
null);
```

```
// Trả về những dòng có cột thứ 3 bằng giá trị cho trước  
// và theo thứ tự của cột thứ 5
```

```
String where = KEY_COL3 + "=" + requiredValue;
```

```
String order = KEY_COL5;
```

```
Cursor someRows = cr.query(MyProvider.CONTENT_URI,  
null, where, null, order);
```

# Example: contact

Activity\_main.xml

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <ListView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/lv"
    ></ListView>

</RelativeLayout>
```

An  
(091) 923-7237

---

Hong  
(091) 937-5732

---

Hong  
(092) 231-3313

# Example

row.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >
  <TextView
    android:id="@+id/textName"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
  <TextView
    android:id="@+id/textValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

# Example

- Set permission to access data

```
<uses-permission  
android:name="android.permission.RE  
AD_CONTACTS"/>
```



# Example

MainActivity.java

```
ListView listView = (ListView) findViewById(R.id.lv);  
Cursor cursor = getContentResolver().query(  
    ContactsContract.CommonDataKinds.Phone.CONTENT_URI  
    , null, null, null, null);  
String[] from= new String[] {Phone.DISPLAY_NAME, Phone.NUMBER};  
int[] to = { R.id.textName, R.id.textValue };  
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this  
    ,R.layout.row, cursor, from, to,0);  
listView.setAdapter(adapter);
```

# SQLite

# SQLite Databases:

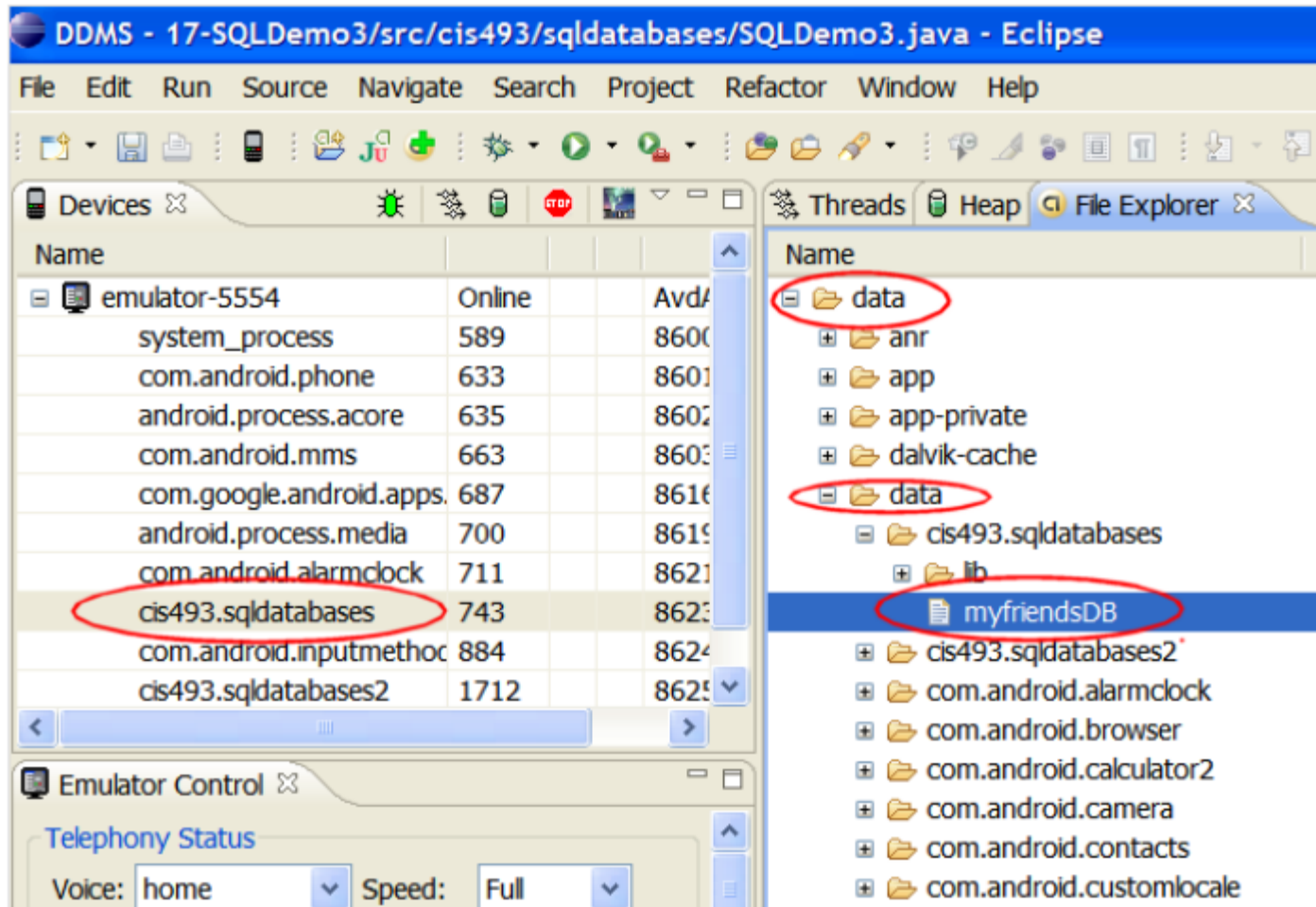
- SQLite is an open source database that is stable, and is popular on many small devices, including Android.
- SQLite is a great fit for Android app development:
  - It's a zero-configuration database.
  - It doesn't have a server.
  - It's a single-file database.
  - It's open source
- Available in all Android device
- No separate setup or administration
- Embedded in Android
- Database Dir:  
DATA/data/APP\_NAME/databases/FILENAME

# Android Package

- Packages
  - android.database
  - android.database.sqlite
- Classes
  - SQLiteOpenHelper  
To manage database creation, and version management.  
<http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>
  - SQLiteDatabase  
To manage SQLite DB  
<http://developer.android.com/reference/android/database/sqlite/SQLiteDatabase.html>

# Database Location

- Emulator's File Explorer showing the placement of the database



# SQLite statement

- Create table

```
create table <name table>(  Colum 1 <Data type >,  
                           Colum 2<Data type >,  
                           Colum 3<Data type >,  
                           . . . . .);
```

- Insert data

```
insert into <name table > [(column)] values(<value>);
```

- Delete data

```
Delete from < name table > [where <condition> ];
```

- Update

```
Update <name table> set <column name>=<value>[,< column  
  name t>=< value >,. . . ]  
[where <condition>];
```

# How to create a SQLite database?

- Method 1

```
public static SQLiteDatabase.openDatabase (  
String path, SQLiteDatabase.CursorFactory factory, int flags )
```


- Open the database according to the flags OPEN\_READWRITE, OPEN\_READONLY, CREATE\_IF\_NECESSARY . Sets the locale of the database to the the system's current locale.
- **Parameters**

path	to database file to open and/or create
factory	an optional factory class that is called to instantiate a cursor when query is called, or null for default
flags	to control database access mode
Returns	the newly opened database
Throws	SQLException if the database cannot be opened

# Example Create a SQLite Database

```
import android.app.Activity;
import android.database.sqlite.*;
import android.os.Bundle;
import android.widget.Toast;

public class SQLDemo1 extends Activity {
    SQLiteDatabase db;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        //  filePath is a complete destination of the form
        //  "/data/data/<namespace>/<databaseName>"
        //  "/sdcard/<databasename>"
        try {
            db = SQLiteDatabase.openDatabase(
                "/data/data/cis493.sqlitedatabases/myfriendsDB",
                null,
                SQLiteDatabase.CREATE_IF_NECESSARY);
            db.close();
        }
        catch (SQLException e) {
            Toast.makeText(this, e.getMessage(), 1).show();
        }
    } // onCreate
} // class
```

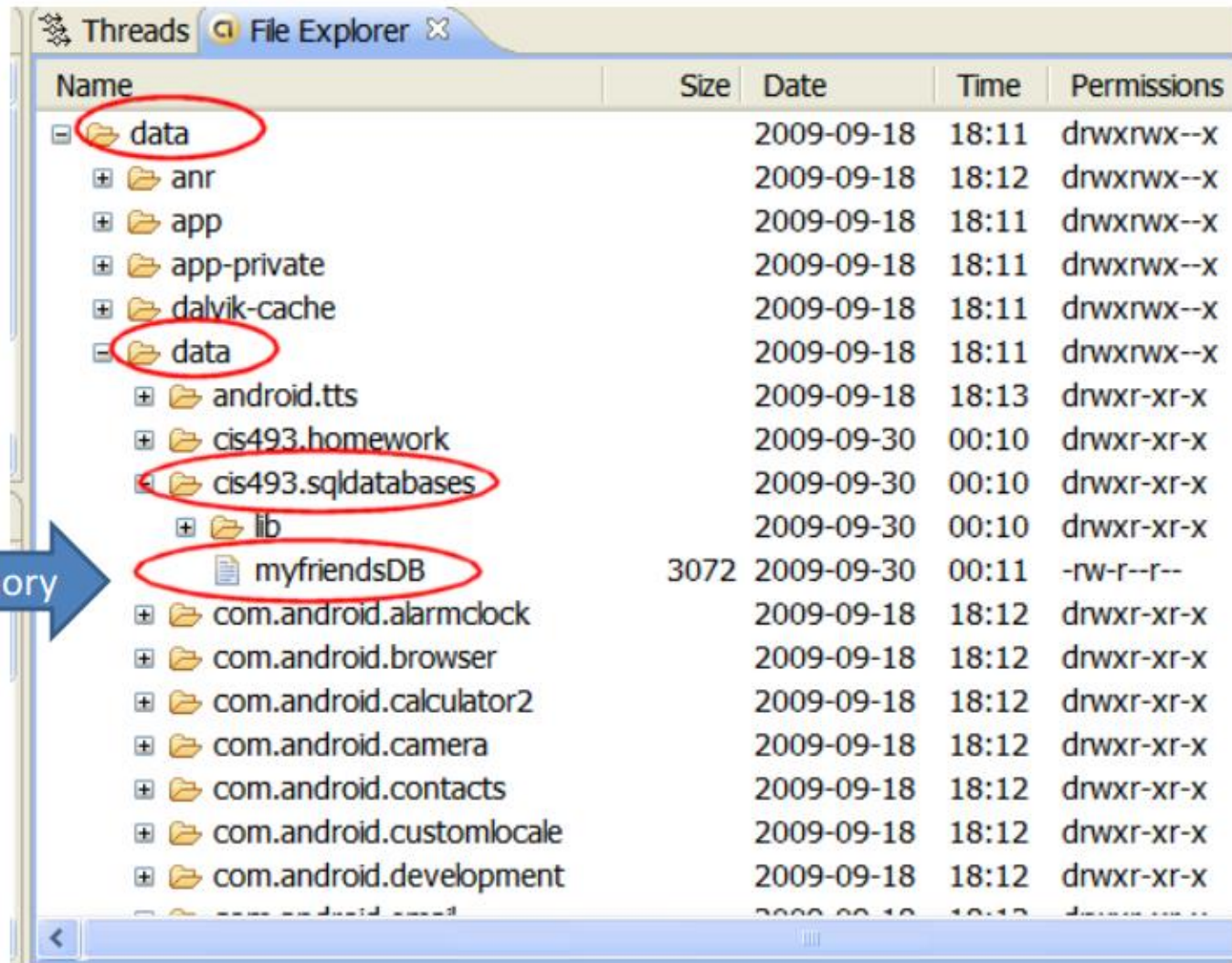




# Example Create a SQLite Database(cont.)

Android's  
System  
Image

Device's memory



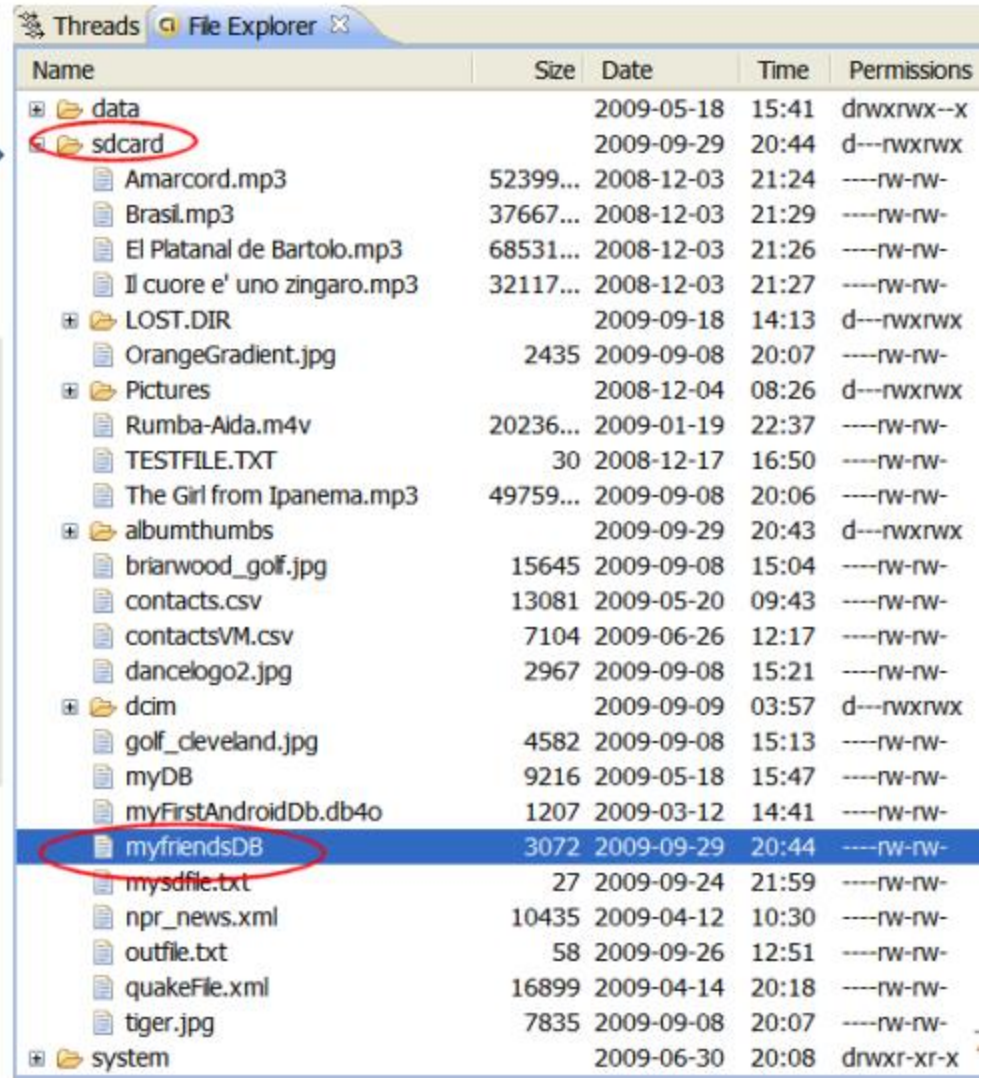
Name	Size	Date	Time	Permissions
data		2009-09-18	18:11	drwxrwx--x
anr		2009-09-18	18:12	drwxrwx--x
app		2009-09-18	18:11	drwxrwx--x
app-private		2009-09-18	18:11	drwxrwx--x
dalvik-cache		2009-09-18	18:11	drwxrwx--x
data		2009-09-18	18:11	drwxrwx--x
android.tts		2009-09-18	18:13	drwxr-xr-x
cis493.homework		2009-09-30	00:10	drwxr-xr-x
cis493.sqldatabases		2009-09-30	00:10	drwxr-xr-x
lib		2009-09-30	00:10	drwxr-xr-x
myfriendsDB	3072	2009-09-30	00:11	-rw-r--r--
com.android.alarmclock		2009-09-18	18:12	drwxr-xr-x
com.android.browser		2009-09-18	18:12	drwxr-xr-x
com.android.calculator2		2009-09-18	18:12	drwxr-xr-x
com.android.camera		2009-09-18	18:12	drwxr-xr-x
com.android.contacts		2009-09-18	18:12	drwxr-xr-x
com.android.customlocale		2009-09-18	18:12	drwxr-xr-x
com.android.development		2009-09-18	18:12	drwxr-xr-x
com.android.email		2009-09-18	18:12	drwxr-xr-x

# Example Create a SQLite Database(cont.)

Creating the  
database file in  
the SD card

Using:

```
db = SQLiteDatabase.openDatabase(  
    "sdcard/myfriendsDB",  
    null,  
    SQLiteDatabase.CREATE_IF_NECESSARY);
```



Name	Size	Date	Time	Permissions
data		2009-05-18	15:41	drwxrwx--x
sdcard		2009-09-29	20:44	d---rwxrwx
Amarcord.mp3	52399...	2008-12-03	21:24	----rw-rw-
Brasil.mp3	37667...	2008-12-03	21:29	----rw-rw-
El Platanal de Bartolo.mp3	68531...	2008-12-03	21:26	----rw-rw-
Il cuore e' uno zingaro.mp3	32117...	2008-12-03	21:27	----rw-rw-
LOST.DIR		2009-09-18	14:13	d---rwxrwx
OrangeGradient.jpg	2435	2009-09-08	20:07	----rw-rw-
Pictures		2008-12-04	08:26	d---rwxrwx
Rumba-Aida.m4v	20236...	2009-01-19	22:37	----rw-rw-
TESTFILE.TXT	30	2008-12-17	16:50	----rw-rw-
The Girl from Ipanema.mp3	49759...	2009-09-08	20:06	----rw-rw-
albumthumbs		2009-09-29	20:43	d---rwxrwx
brianwood_golf.jpg	15645	2009-09-08	15:04	----rw-rw-
contacts.csv	13081	2009-05-20	09:43	----rw-rw-
contactsVM.csv	7104	2009-06-26	12:17	----rw-rw-
dancelogo2.jpg	2967	2009-09-08	15:21	----rw-rw-
dcim		2009-09-09	03:57	d---rwxrwx
golf_cleveland.jpg	4582	2009-09-08	15:13	----rw-rw-
myDB	9216	2009-05-18	15:47	----rw-rw-
myFirstAndroidDb.db4o	1207	2009-03-12	14:41	----rw-rw-
myfriendsDB	3072	2009-09-29	20:44	----rw-rw-
mysdfile.txt	27	2009-09-24	21:59	----rw-rw-
npr_news.xml	10435	2009-04-12	10:30	----rw-rw-
outfile.txt	58	2009-09-26	12:51	----rw-rw-
quakeFile.xml	16899	2009-04-14	20:18	----rw-rw-
tiger.jpg	7835	2009-09-08	20:07	----rw-rw-
system		2009-06-30	20:08	drwxr-xr-x

# create a SQLite database(cont.)

- An alternative way of opening/creating a SQLITE database in your local Android's System Image is given below:

```
SQLiteDatabase db = this.openOrCreateDatabase(  
    "myfriendsDB2",  
    MODE_PRIVATE,  
    null);
```

1. "myFriendsDB2" is the abbreviated file path. The prefix is assigned by Android as: /data/data/<app namespace>/databases/myFriendsDB2.
2. MODE could be: MODE\_PRIVATE, MODE\_WORLD\_READABLE, and MODE\_WORLD\_WRITEABLE. Meaningful for apps consisting of multiples activities.
3. null refers to optional factory class parameter

# Cursors

- Android cursors are used to gain (random) access to tables produced by SQL select statements.
- Cursors primarily provide one row-at-the-time operations on a table.
- Cursors include several types of operator, among them:
  - 1. *Positional awareness operators*** (isFirst(), isLast(), isBeforeFirst(), isAfterLast() ),
  - 2. *Record Navigation*** (moveToFirst(), moveToLast(), moveToNext(), moveToPrevious(), move(n) )
  - 3. *Field extraction*** (getInt, getString, getFloat, getBlob, getDate, etc.)
  - 4. *Schema inspection*** (getColumnName, getColumnNames, getColumnIndex, getColumnCount, getCount)

# Creating-Populating a Table

- SQL Syntax for the creating and populating of a table looks like this:

recID	name	phone
1	AAA	555
2	BBB	777
3	CCC	999

```
create table tblAMIGO (  
    recID integer PRIMARY KEY autoincrement,  
    name text,  
    phone text );
```

```
insert into tblAMIGO(name, phone) values ('AAA', '555' );
```

# Creating-Populating a Table(cont.)

- We will use the `execSQL(...)` method to manipulate SQL action queries. The following example creates a new table called `tblAmigo`.
- **The table has three fields:** a numeric unique identifier called `recID`, and two string fields representing our friend's name and phone. If a table with such a name exists it is first dropped and then created anew. Finally three rows are inserted in the table.

```
db.execSQL("create table tblAMIGO ("
           + " recID integer PRIMARY KEY autoincrement, "
           + " name  text, "
           + " phone text ); " );
```

```
db.execSQL( "insert into tblAMIGO(name, phone) values ('AAA', '555' );" );
db.execSQL( "insert into tblAMIGO(name, phone) values ('BBB', '777' );" );
db.execSQL( "insert into tblAMIGO(name, phone) values ('CCC', '999' );" );
```



# Comments

1. The field `recID` is defined as `PRIMARY KEY` of the table. The “autoincrement” feature guarantees that each new record will be given a unique serial number (0,1,2,...).
2. The database data types are very simple, for instance we will use: `text`, `varchar`, `integer`, `float`, `numeric`, `date`, `time`, `timestamp`, `blob`, `boolean`, and so on.
3. In general, any well-formed SQL action command (`insert`, `delete`, `update`, `create`, `drop`, `alter`, etc.) could be framed inside an `execSQL(...)` method.
4. You should make the call to `execSQL` inside of a try-catch-finally block. Be aware of potential `SQLiteException` situations thrown by the method.

# QUERY vs RAW QUERY

- **Raw Query** take for input a syntactically correct SQL-select statement. The select query could be as complex as needed and involve any number of tables (remember that outer joins are not supported).
- **Query** are compact parametrized select statements that operate on a single table (for developers who prefer not to use SQL).
- **rawQuery**(String query , String[] array);
- **Query**(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)
- **execSQL**(String sql)

Execute a single SQL statement that is NOT a SELECT or any other SQL statement that returns data.



# QUERY vs RAW QUERY

## RAW QUERY

```
String query = "select * from  
student_demo where _id = ?  
And age > ? Order by s_name  
limit ?";  
Cursor c = db.rawQuery(  
query, new  
String[] {"1","23","5"}  
);
```

## QUERY

```
String[] cols =  
{"_id","s_name"};  
String[] params = {"1","23"};  
Cursor c =  
db.query("student_demo",  
cols,"_id = ? And age =  
?",params,null,  
Null,5  
);
```

# RAW QUERY

- Consider the following code fragment

```
Cursor c1 = db.rawQuery(  
    "select count(*) as Total from tblAMIGO",  
    null);
```

- The previous rawQuery contains a select-statement that counts the rows in the table tblAMIGO.
- The result of this count is held in a table having only one row and one column. The column is called "Total".
- The cursor c1 will be used to traverse the rows (one!) of the resulting table.
- Fetching a row using cursor c1 requires advancing to the next record in the answer set.
- Later the (singleton) field total must be bound to a local Java variable.

# RAW QUERY(Cont.)

- Assume we want to count how many friends are there whose name is 'BBB' and their recID > 1. We could use the following construction

```
String mySQL = "select count(*) as Total "  
               + " from tblAmigo "  
               + " where recID > ? "  
               + " and name = ? ";
```



```
String[] args = {"1", "BBB"};
```

```
Cursor c1 = db.rawQuery(mySQL, args);
```

# SIMPLE QUERY

- The signature of the Android's simple query method is:

```
query( String    table,  
       String[] columns,  
       String    selection,  
       String[] selectionArgs,  
       String    groupBy,  
       String    having,  
       String    orderBy )
```

```
String[] columns =  
    {"Dno", "Avg(Salary) as AVG"};  
  
String[] conditionArgs =  
    {"F", "123456789"};  
  
Cursor c = db.query(  
    "EmployeeTable",  
    columns,  
    "sex = ? And superSsn = ? " ,  
    conditionArgs,  
    "Dno",  
    "Count(*) > 2",  
    "AVG Desc "  
);
```

```
← table name  
← columns  
← condition  
← condition args  
← group by  
← having  
← order by
```

# Using SQLITE Command Line

- The Android SDK contains a command line interface to SQLITE databases.
- To open/Create a database use the command  
**C:> sqlite3 myNewDatabase**
- You may directly reach the Emulator's data folder and operate on existing databases.
- Assume an emulator is running.
- We will use **adb shell** to tap in the emulator's internal memory

```
E:\Android> adb shell
```

```
# sqlite3 /data/data/matos.sql1/databases/myfriendsDB
```

```
sqlite3 /data/data/matos.sql1/databases/myfriendsDB
```

```
SQLite version 3.5.9
```

```
Enter ".help" for instructions
```

# Summary of SQLITE3 commands

```
sqlite3> .help
```

<code>.bail ON OFF</code>	Stop after hitting an error. Default OFF
<code>.databases</code>	List names and files of attached databases
<code>.dump ?TABLE? ...</code>	Dump the database in an SQL text format
<code>.echo ON OFF</code>	Turn command echo on or off
<code>.exit</code>	Exit this program
<code>.explain ON OFF</code>	Turn output mode suitable for EXPLAIN on or off.
<code>.header(s) ON OFF</code>	Turn display of headers on or off
<code>.help</code>	Show this message
<code>.import FILE TABLE</code>	Import data from FILE into TABLE
<code>.indices TABLE</code>	Show names of all indices on TABLE
<code>.load FILE ?ENTRY?</code>	Load an extension library

# Summary of SQLITE3 commands(Cont.)

<code>.mode MODE ?TABLE?</code>	Set output mode where MODE is one of:
<code>csv</code>	Comma-separated values
<code>column</code>	Left-aligned columns. (See <code>.width</code> )
<code>html</code>	HTML <code>&lt;table&gt;</code> code
<code>insert</code>	SQL insert statements for TABLE
<code>line</code>	One value per line
<code>list</code>	Values delimited by <code>.separator</code> string
<code>tabs</code>	Tab-separated values
<code>tcl</code>	TCL list elements
 <code>.nullvalue STRING</code>	 Print STRING in place of NULL values
<code>.output FILENAME</code>	Send output to FILENAME
<code>.output stdout</code>	Send output to the screen
<code>.prompt MAIN CONTINUE</code>	Replace the standard prompts

# Summary of SQLITE3 commands(Cont.)

<code>.quit</code>	Exit this program
<code>.read FILENAME</code>	Execute SQL in FILENAME
<code>.schema ?TABLE?</code>	Show the CREATE statements
<code>.separator STRING</code>	Change separator used by output mode and <code>.import</code>
<code>.show</code>	Show the current values for various settings
<code>.tables ?PATTERN?</code>	List names of tables matching a LIKE pattern
<code>.timeout MS</code>	Try opening locked tables for MS milliseconds
<code>.width NUM NUM ...</code>	Set column widths for "column" mode



# Example

```
sqlite> .tables
.tables
android_metadata tblAMIGO

sqlite> select * from tblAMIGO;

1|AAAXXX|555
2|BBBXXX|777
3|Maria|999
4|Maria|000
5|Maria|001

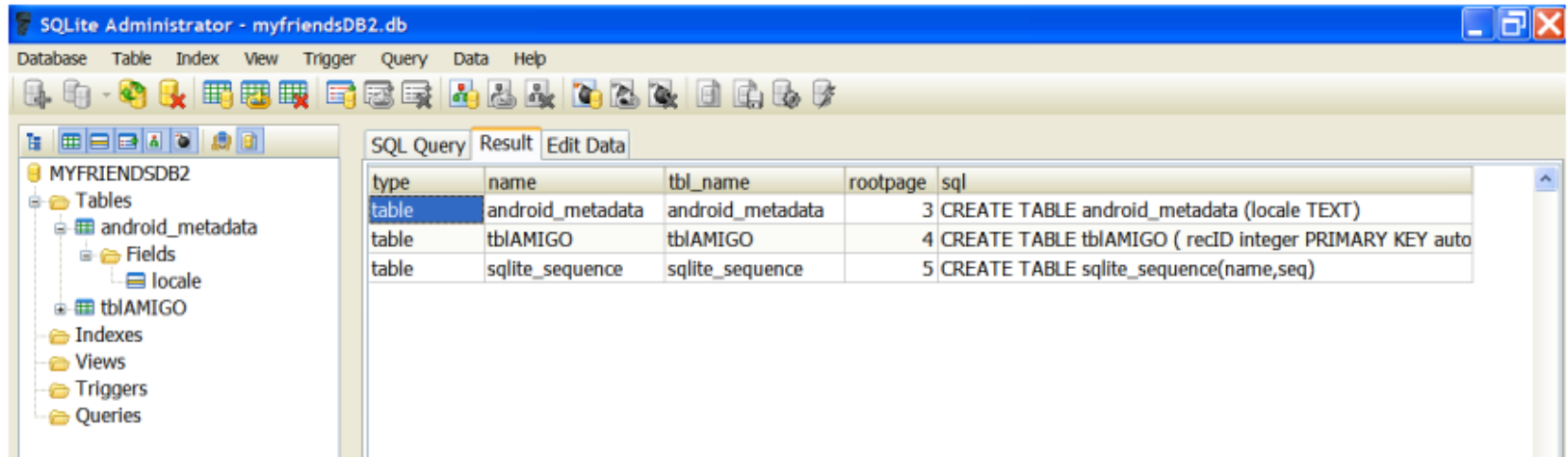
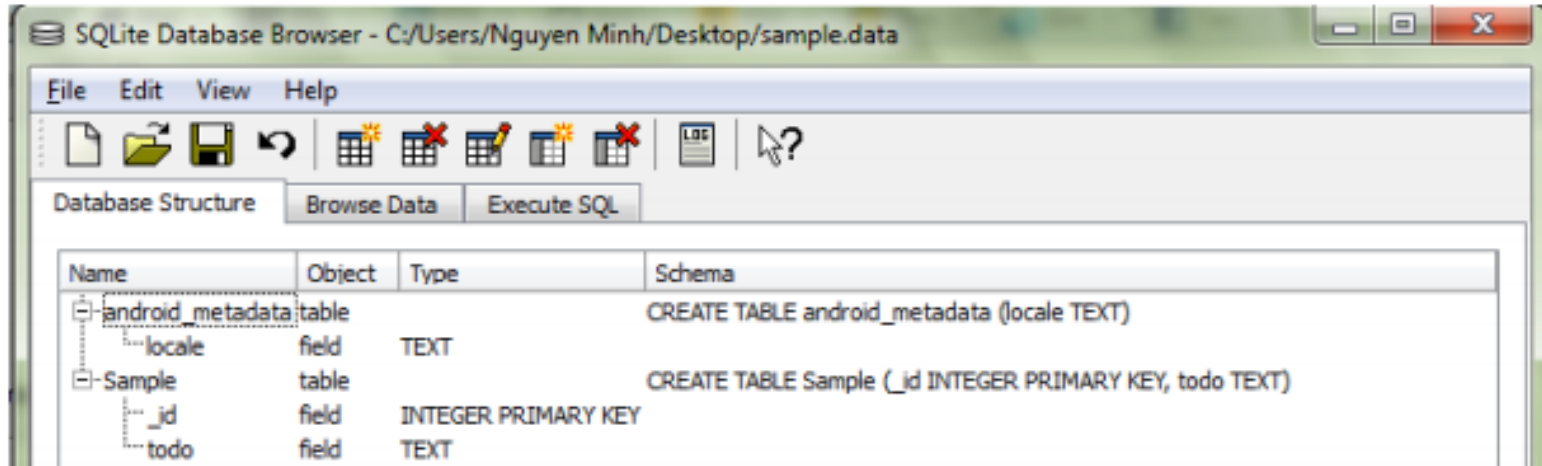
sqlite> .exit
#
```

# Move, copy database

- In order to move a copy of the database in and out of the Emulator's storage space and either receive or send the file into/from the local computer's file system you may use the commands:
  - `adb pull <full_path_to_database>`
  - `adb push <full_path_to_database>`
- You may also use the Eclipse's DDMS Perspective to push/pull files in/out the emulator's file system.
- Once the database is in your computer's disk you may manipulate the database using a 'user-friendly' tool such as:
  - SQLite Manager (Firefox adds-on)
  - SQLite Administrator (<http://sqliteadmin.orbmu2k.de>)



# GUI Tool



# Copy database in project

- Copy database file to /assets directory
- Create class **DatabaseHelper** extend **SQLiteOpenHelper** in package "android.database.sqlite".
- createDataBase() to copy database file.
- EX:

```
public class DataBaseHelper extends SQLiteOpenHelper{  
    private static String DB_PATH =  
        "/data/data/YOUR_PACKAGE/databases/";  
    //tên file cơ sở dữ liệu  
    private static String DB_NAME = "myDBName";  
    private SQLiteDatabase myDataBase;  
    private final Context myContext;  
    public DataBaseHelper(Context context) {  
        super(context, DB_NAME, null, 1);  
        this.myContext = context;  
    }  
}
```

# Copy database in project(cont.)

```
public void createDataBase()
throws IOException{
    boolean dbExist =
    checkDataBase();
    if(dbExist){
        return;
    }else{
        this.getReadableDatabase();
        try {
            copyDataBase();
        } catch (IOException e) {
            throw new Error("Error copying
            database");
        }
    }
}
```

```
private boolean checkDataBase(){
    SQLiteDatabase checkDB = null;
    try{
        String myPath = DB_PATH +
        DB_NAME;
        checkDB =
        SQLiteDatabase.openDatabase(myPat
        h, null,
        SQLiteDatabase.OPEN_READONLY);
    }catch(SQLiteException e){
        //database doesn't exist yet.
    }
    if(checkDB != null){
        checkDB.close();
    }
    return checkDB != null ? true : false;
}
```

# Copy database in project(cont.)

```
private void copyDataBase()
throws IOException{
    InputStream myInput =
    myContext.getAssets().open(DB_N
    AME);
    String outFileName = DB_PATH +
    DB_NAME;
    OutputStream myOutput = new
    FileOutputStream(outFileName);
    byte[] buffer = new byte[1024];
    int length;
    while ((length =
    myInput.read(buffer))>0){
        myOutput.write(buffer, 0, length);
    }
    myOutput.flush();
    myOutput.close();
    myInput.close();
}
```

```
public void openDataBase() throws
SQLException{
    String myPath = DB_PATH +
    DB_NAME;
    myDataBase =
    SQLiteDatabase.openDatabase(myPath, null,
    SQLiteDatabase.OPEN_READONLY);
}
@Override
public synchronized void close() {
    if(myDataBase != null)
        myDataBase.close();
    super.close();
}
@Override
public void onCreate(SQLiteDatabase db) {
}
@Override
public void onUpgrade(SQLiteDatabase db,
int oldVersion, int newVersion)
```

# References

- Book: Android in Action
- The Busy Coder's Guide to Android Development by Mark L. Murphy
- Slide of Victor Matos
- <http://developer.android.com/>