ANDROID

CH - 3

Database Connectivity & Content Provider

Prepared By :

Ms. Kakadiya Jainam A.

Points Of learn :

- 4.1 Saving Data
- 4.2 Saving Key-value Sets
- 4.3 Saving files
- 4.4 Saving data in SQL Database
- 4.5 Sending Simple data to other Apps
- 4.6 Receiving Simple Data from Other databse

Basic Introduction

- SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.

- SQLite supports all the relational datbase features. In order to access this database , you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

4.1 Saving data

- Most Android apps need to save data, even if only to save information about the app state during onPause() so the user’s progress is not lost.

- Most non-trivial apps also need to save user setting, and some apps must manage larger amounts of information in files and database.
  - Saving key value pairs simple data types
  - Saving arbitrary files in Android's file system
  - Using databases managed by SQLite

4.2 Saving Key value sets

- If you have a relatively small collection of key-values that you’d like to save, you should used the SharedPreferences APIs.
- A SharedPreferences objects points to a file containing key-value pairs and provides simple method to read and write them.
- Each SharedPreferences files is managed by the framework and can be private or shared.

4.2.1 Get a Handle to a SharedPreferences

- You can create a new SharedPreferences files or access an existing one by calling one of two methods :
  - **1) getSharedPreferences() :** Use this method if you need multiple SharedPreferences files identified by name, which you specify with the first parameter. You can call this from any Context in your app.
  - **2) getPreferences() :** Use this from an Activity if you need to use only one SharedPreferences file for the activity.

- For example, the following code is executed inside a Fragment. It access a shared preference file that's identified by the resources string R.String.prefernce_file_key and opens it using the private mode so the file is accessible by only your app.

  Context context=getActivity();

  SharedPreferences sharedPref=context.getSharedPreferences(getString(R.string.preference_file_key),contextMODE_PRIVATE);

  OR

  SharedPreferences sharedPref=getActivity().getPreferences(Context.MODE_PRIVATE);

4.2.2 Write to Shared Preferences

- To write to a shared Preferences file, create a SharedPreferences. Editor by calling edit() on your SharedPreferences.

- Pass the key and value you want to write with methods such as putInt() and putString(). Then call commit() to save the changes.

- **For example :**

```
SharedPreferences
sharedpf=getActivity().getPreferences(Context
.MODE_PRIVATE);

SharedPreferences.Editor editor=sharedPref.
edit();

editor.putInt(getString(R.string.saved_high_sc
ore),newHeightScore);

editor.commit();
```

4.2.3 Read to Shared Preferences

- To retrieve values from a shared preference file, call methods such as getInt() and getString(), providing the key for the value you want and optionally a default value to return if the key isn't present.
- **For example:**

SharedPreferences sharedPref= getActivity().
getPrefereneces(Context.MODE_PRIVATE);

int default value=getResources().getInteger
(R.String.saved_high_score_default);

sharedPref.getInt(getString(R.String.saved_high
_score),defaultValue);

4.3 Saving Files

- Android uses a file system that's similar to disk-based file systems on other platforms.
- A file object is suited to reading or writing large amount of data in start-to-finish order without skipping around.
- For example, it's good image files or anything exchanged over a network.

4.3.1 Choose Internal or External Storage

- All Android devices have two file storage area: “Internal” and “External” storage.

- Some devices divided the permanent storage space into “internal” and “external” partitions, so even without a removable storage medium, there are always two storage space and the API behaviour is the same whether the external storage is removable or not.

- The following lists summarize the facts about each storage space.

- **Internal storage :**
  - It’s always available.
  - Files saved here are accessible by only your app by default.
  - When the user uninstalls your app, the system removes all your app’s files from internal storage.
  - Internal storage is best when you want to be sure that neither the user nor other apps can access your files.

- **External storage :**
  - It’s not always available, because the user can mount the external storage as USB storage and in some cases remove it from the device.
  - It’s world-readable, so files saved here may be read outside of your control.
  - When the user uninstalls your app, the system removes your app’s files from here only if you save them in the directory from getExternalFilesDir().

4.3.2 Obtain Permissions for External Storage

- To write to the external storage, you must request the WRITE_EXTERNAL_STORAGE permission in your manifest file :

```
<manifest...>
    <uses-permission
    android:name=“android.permission.WRITE_
    EXTERNAL_STORAGE” />
</manifest>
```

4.3.3 Save Files on Internal Storage

- When saving a files to internal storage, you can acquire the appropriate directory as a File by calling one of **two** methods:
  - **1) geFilesDir() :** Returns a File representing an internal directory for your app.
  - **2) getCachDir() :** Returns a File representing an internal directory for your app’s temporary cache files. If the system begins running low on storage, it may delete your cache files without warning.

- **For example, here's how to write some text to a file :**

```
String data=edata.getText.toString();
FileOutputStream fos;
try { fos=openFileOutput(filename, Context.
MODE_PRIVATE);
fos.write(data.getBytes());
fos.close();
Toast.makeText(MainActivity.this, “data written
done”, Toast.LENGTH_LONG).show();
}catch(Exception e) {
Toast.makeText(MainActivity.this, e.toString(),
Toast.LENGTH_LONG).show();
```

4.3.3 Save Files on External Storage

- Because the external storage may be unavailable such as when the user has mounted the storage to a PC or has removed the SD card that provides the external storage you should always verify that the volume is available before accessing it.

- Although the external storage is modify by the user and other apps, there are two categories of files you might save here :

```
Public isExternalStorageWritable() {
Sting state=Environment.getExternalStorageState();
If(Envoronment.MEDIA_MOUNTED.equals(state))
{
Return true;
}
Returns false;
```

- **Public files :**Files should be freely available by the user and other apps and to the user.
  - **For ex,** photos captured by your app or other downloaded files.
- **Private files :** Files that rightfully belong to your app and should be declared when the user uninstalls your app.
  - **For ex,** additional resources downloaded by your app or temporary media files.

4.3.4 Delete File

- You should always delete files that you no longer need. The most straightforward way to delete a file is to have the opened file reference call delete() on itself.

  - **myFile.delete();**

- If the file is saved on internal storage, you can also ask the Context to locate and delete a file by calling deleteFile():

  - **myContext.deleteFile(fileName);**

4.4 Saving Data in SQL Databases

- Saving data to a database is ideal for repeating data, such as contact information. This class assumes that you are familiar with SQL database in general and helps you get started with SQLite databases on Android.

- Define a Schema and Contract : One of the main principles of SQL databases is the schema is the schema: a formal declaration of how the database is organized. The schema is reflected in the SQL statements that you use to create your database.

4.4.1 Create a Database Using a SQL Helper

- Once you have define how your database looks, you should implement methods that create and maintain the database and table. Here are some typical statements that create and delete a table:

- Private static final String TEXT_TYPE= “TEXT”;
- Private static final String COMMA_SEP= “,”;
- Private static final String SQL CREATE ENTRIES = “CREATE TABLE”+Bentry.TABLE_NAME+ “(“ +Bentry._ID+ “INTEGER PRIMARY KEY,” + Bentry.COLUMN_NAME_TITLE + TEXT_TYPE”)”;

4.4.1 Create a Database Using a SQL Helper

- SQL_DELETE_ENTRIES = “DROP TABLE IF EXISTS” + Bentry.TABLE_NAME;

- Just like files that you save on the device’s internal storage. Android stores your database in private disk space that’s associated application. Your data is secure, because by default this area is not accessible to other applications.

- A useful set of APIs is available in the SQLiteOpenHelper class.

- To use SQLiteOpenHelper, create a subclass that overrides the onCreate(), onUpgrade() and onOpen() callback methods. You may want to implement onDowngrade(), but it’s not required.

4.4.2 Put Information into a Database

- Insert data into the database by parsing a contentValue object to the insert() method :

```
SQLiteDatabase db=mDbHelper.
getWritetableDatabase();
contentValues values=new ContenttValues();
long newRowid;
newRowid=db.insert(Bentry.TABLE_NAME,Bentr
y.COLUMN_NAME_NULLABLE,values);
```

4.4.3 Read Information from a Database

- To read from a database, use the query() method, passing it your selection criteria and desired columns.

- The method combines elements of insert() and update(), except the column list defines the data you want to fetch, rather than the data to insert.

- The results of the query are returned to you in a Cursor object.

SQLiteDatabase db= mDHelper.
getReadableDatabase();

String[] projection= {Bentry._ID,
Bentry.COLUMN_NAME_TITLE,
Bentry.COLUMN_NAME_UPDATE};

String
sortOrder=Bentry.COLUMN_NAME_UPDATE +
“DESC”;

Cursor C=db.query(Bentry.TABLE_NAME,
projection, selection, null, sortOrder);

4.4.4 Delete Information from a Database

- To delete rows from a table, you need to provide selection criteria that identify the rows. The database API provides a mechanism for creating selection criteria that protects against SQL injection.

- The mechanism divides the selection specification into a selection clause and selection arguments.

- String selection=Bentry.COLUMN_NAME_ ENTRY_ID+ “LIKE ?”; //define where part of query
  bd.delete(table_name, selection, selectionArgs);

4.4.5 Update Information in a Database

- When you need to modify a subset of your database values, use the update() method.

- Updating the table combines the content values syntax of insert() with the where syntax of delete().

- SQLiteDatabse db=mDbHelper. getReadableDatabase();
  ContentValues values=new ContentValues();
  values.put(Bentry.COLUMN_NAME_TITLE, title);
  String selection=Bentry.COLUMN_NAME_ ENTRY_ID+ “LIKE ?”;

```
String[] selectionArgs={String.valueOf(rowId)};
int
count=db.update(BReadDbHelper.Bentry.TABLE
_NAME, values, selection, selectionArgs);
```

4.5 SENDING SIMPLE DATA TO OTHER APPS

- When you construct intent, you must specify the action you want the intent to “trigger”. Android defines several actions, including ACTION_SENT which, as you can probably guess, indicates that the intent is sending data from one activity to another.

- To send data to another activity, all you need to do is specify the data and its type. Sending and receiving data between applications with intents is most commonly used for social sharing of content.

- Here is the code to implement this type of sharing

```
Intent sendIntent=new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, ”This is my text to send”);
sentIntent.setType(“text/plain”);
startActivity(sendIntent);
```

4.5.1 Send Multiple Pieces of Content

- To share multiple pieces of content, use the ACTION_SEND_MULTIPLE action together with a list of URIs pointing to the content.
- For example, if you share mixture of image types, it should be “image/*” to match an activity that handles any type of images.

- **Here’s an Example :**

```
ArrayList<Uri> imageUris=new ArrayList<Uri>();
imageUris.add(imageUri1);// add your image here
imageUris.add(imageUri2);
Intent shareIntent=new Intent();
shareIntent.setAction(Intent.ACTION_SENT_MUL
TIPLE);
shareIntent.puParcelArrayListExtra(Intent.EXTRA
_STREAM,imageUris);
shareIntent.setType(“image/*”);
startActivity(Intent.createChooser(shareIntent,
“Share image to….”));
```

4.6 RECEIVING SIMPLE DATA FROM OTHER APPS

- Just as your application can send data to other application, so too can it easily receive data from applications.

- For example, a social networking application would likely be interested in receiving text content, like an interesting web URL, from another app.

- The Google+ Android application accepts both text and single or multiple images.

4.6.1 Update Your Manifest

- Intent filters inform the system what intents an application component is willing to accept. You create intent filters in order to be able to receive intents with this action.

- You define an intent filter in your manifest, using the <intent-filter> element. For example, if your application handles receiving text content, a single image of any type, or multiple images of any type, your manifest would look like :

```
<activity android:name=“.ui.MyActivity”>
   <intent-filter>
      <action android:name=“android.intent.
        Action.SEND” />
      <category android:name=“android.intent.
        catagory.DEFAULT” />
      <data android:mimeType=“image/*”>
   </intent-filter><intent-filter>
      <action android:name=“android.intent.
        Action.SEND” />
      <category android:name=“android.intent.
        catagory.DEFAULT” />
      <data android:mimeType=“text/plain”>
   </intent-filter> </activity>
```