ANDROID DATA STORAGE

Android Data Storage Options

Android provides several data storage options for you to save persistent application data depends on your specific needs: private/public, small/large datasets :-

Internal Storage : Store private data on the device memory.

External Storage : Store public data on the shared external storage.

Network Storage : Store data on the web with your own network server.

Shared Preferences: *Store private primitive data in key-value pairs.*

SQLite Databases : Store structured data in a private database.

Content Provider : Shared repository globally shared by all apps.

SHARED PREFERENCE

SharedPreference

- *SharedPreference* is an Android lightweight mechanism to store and retrieve <key-value> pairs of primitive data types.
- *SharedPreference* 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.*

```
<map>
<int name="ACCESS_COUNT" value="3" />
</map>
```

Get a Handle to SharedPreference File

You can create a new shared preference file or access an existing one by calling one of two methods:

1. **getPreferences() :** Use this from an Activity if you need to use only one shared preference file for the activity. Because this retrieves a default shared preference file that belongs to the activity, you don't need to supply a name

```
SharedPreferences sharedPref = this.getPreferences(Context.MODE_PRIVATE);
```

2. **getSharedPreferences() :** Use this if you need multiple shared preference files identified by name, which you specify with the first parameter. You can call this from any Context in your app

```
SharedPreferences sharedPref = this.getSharedPreferences(
  getString(R.string.preference_file_key),   Context.MODE_PRIVATE);
```

File Location: /data/data/packageName/shared_prefs/

Read from Shared Preference

- To retrieve values from a shared preferences 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:

```
int  highScore = sharedPref.getInt(“ KEY NAME”, defaultValue);
```

Write to Shared Preference

- To write to a shared preferences file, create a SharedPreferences.Editor by calling edit() on SharedPreferences
- Pass the keys and values you want to write with methods such as putInt() and putString(). Then call commit() to save the changes

```
SharedPreferences.Editor editor = sharedPref.edit();
editor.putInt("KEY NAME", value);
editor.commit();
```

Removing from Shared Preference

- Get a reference to Shared Preferences Editor by calling edit()
- Editor object's method remove(String Key), will delete the object from the file
- This need to be committed using commit()

```
SharedPreferences.Editor editor = sharedPref.edit();
editor.remove("KEY NAME");
editor.commit();
```

FILE DATA STORAGE

Android File Storage (interenal /external)

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.
•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().
External storage is the best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.

•Although apps are installed onto the internal storage by default, you can specify the android:installLocation attribute in your manifest so your app may be installed on external storage.
•To write /Read to the external storage, you must request the WRITE_EXTERNAL_STORAGE or READ_EXTERNAL_STORAGE permission in your manifest file

Android Internal Stroage

❖ **Internal File Location**

When saving a file to internal storage, you can acquire the appropriate directory as a File by calling one of two methods:

- getFilesDir()
- getCacheDir()

❖ **Writing to a File**

The method openFileOutput() should be called to get a FileOutputStream that writes to a file in your internal directory.

```
String fileName="sisoft.txt";
String str="Output to File";
FileOutputStream  fos ;
try{    fos = openFileOutput(filename, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
}   catch (Exception e) {   e.printStackTrace();  }
```

Android Internal Stroage

❖ **Reading File**

• The method openFileInput() should be called to get a FileInputStream that reads to a file in your internal directory.

```
 try
{
FileInputStream fis ;
fis = openFileInput("sisoft.txt");
String temp=""; int c ;
 while( (c = fis.read()) != -1)
{          temp = temp + Character.toString((char)c);        }
Toast.makeText(this, "READ:"+temp, Toast.LENGTH_LONG).show();
}
catch(Exception e) {  Toast.makeText(this, e.getMessage(), Toast.LENGTH_LONG).show();}
```

❖ Delete a file in internal storage by ::> myContext.deleteFile(fileName);

Android external Storage

- Methods and constants provided through Environment class
- Check availability

  String getExternalStorageState()
- Get directory for external storage using

  File getExternalFilesDir (String type)
  - /<external_storage>/Android/data/<package>/files/
- For public files (not deleted on uninstall)

  File getExternalStoragePublicDirectory(String type)

Exmp:

```
File file = new File (getExternalFilesDir (null),"Demo.txt");
OutputStream os = new FileOutputStream (file);
os.write (data);
```

- Delete file by ***file.delete();***

SQLITE DATA STORAGE

Sqlite Database

- Persistent storage of data
- Popular embedded database
- Combines SQL interface with small memory footprint and decent speed
- SQLite implements most of the SQL-92 standard for SQL.
- SQLITE does not implement referential integrity constraints through the foreign key constraint model and not support for outer join.

- Integrated in Android runtime
- Native API not JDBC
- Data accessible to a single application (the owner)
- Tools: http://sqlitebrowser.org/

SQLite: Data Types

- **NULL** – null value
- **INTEGER** - signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value
- **REAL** - a floating point value, 8-byte IEEE floating point number.
- **TEXT** - text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
- **BLOB**. The value is a blob of data, stored exactly as it was input.

Android SQLite Support Classes

- SqliteOpenHelper
- SqliteDatabase
- ContentValues
- Cursors

SQLiteOpenHelper

- SQLiteOpenHelper is a helper class to manage database creation and version management
- To use SQLiteOpenHelper in Android, a Java class should be creating extending SQLiteHelper
- It has defined constructor to create the database. When you are extending it , you have to create a constructor class which defines and create database named in it
  - **public SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory factory, int version)**
  - **public SQLiteOpenHelper (Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)**
- This class has callback methods onCreate, onUpgrade, onDownGrade, onOpen etc, these needs to be overridden
- It has methods getReadableDatabase() and getWritableDatabase(). These return handle to SQLiteDatabase

SQLiteDatabase

- SQLiteDatabase is mainly for CRUD and queries operations.
- It has methods to create, delete, execute SQL commands, and perform other common database management tasks

- Methods in sqliteDatabase :
  - execSql(“string”): for performing any type for CRUD operations.mainly for create operation.
  - insert(“tablename”, “nullColumnHack”, contentvalue)
  - update(“tablename”,”contentvalue”,”whereargscondition”,”conditionarguments”)
  - delete (“tablename”, ”whereargscondition”, ”conditionarguments”)
  - rawQuery(“string”): for select query operation in database
  - query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)
  - close()

ContentValues

- The Class ContentValues allows to define key/values
- The *key* represents the table column identifier and the *value* represents the content for the table record in this column.
- ContentValues are used for inserts and updates of database entries.

CURSOR

Cursors are used to gain (sequential & 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 operators, among them:

- **Positional awareness operators**: (isFirst(), isLast(), isBeforeFirst(), isAfterLast()),
- **Record Navigation** : (moveToFirst(), moveToLast(), moveToNext(), moveToPrevious(), move(n))
- **Field extraction**: (getInt, getString, getFloat, getBlob, getDate, etc.)
- **Schema inspection**: (getColumnName, getColumnNames, getColumnIndex, getColumnCount, getCount)

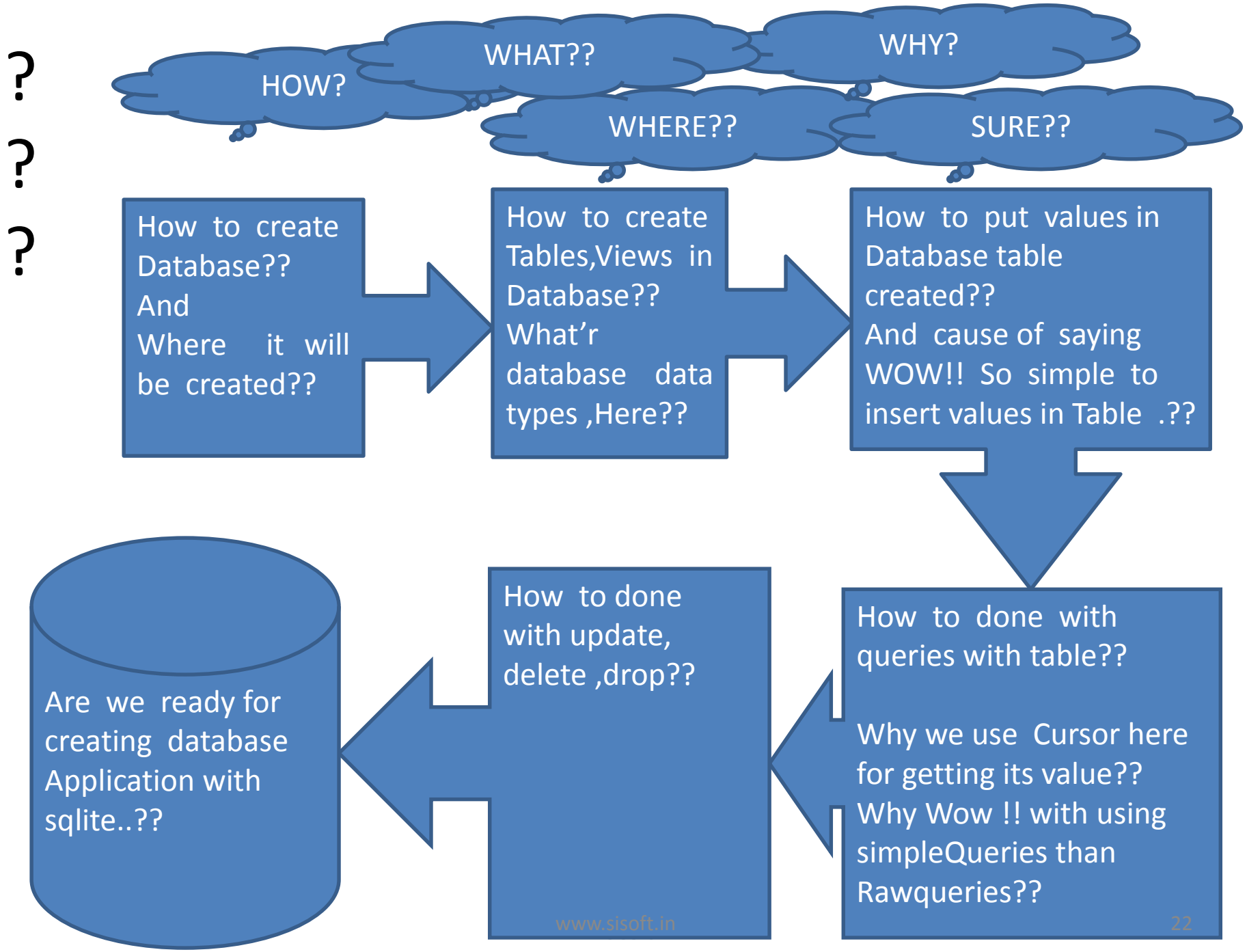
❖The getColumnIndex method is called to determine the position of chosen columns in the current row.

❖The getters: getInt, getString commands are used for field extraction.

❖The moveToNext command forces the cursor to displace from its before-first position to the first available row.

❖The loop is executed until the cursor cannot be advanced any further.

❖Cursors provide **READ_ONLY access to records.**


?
?
?
HOW?
WHAT??
WHY?
WHERE??
SURE??
How to create Database?? And Where it will be created??
How to create Tables,Views in Database?? What'r database data types ,Here??
How to put values in Database table created?? And cause of saying WOW!! So simple to insert values in Table .??
How to done with queries with table??
Why we use Cursor here for getting its value??
Why Wow !! with using simpleQueries than Rawqueries??
How to done with update, delete ,drop??
Are we ready for creating database Application with sqlite..??

Creating and Using Database

➢ Using Subclass of SQLiteOpenHelper :: >

A helper class to manage database creation and version management.

When you are extending it , you have create a construstor class which defines and create database named in it.

Also , you have to override onCreate() and OnUpgrade() methods of it.

➢ *openorCreateDatabase() or openDatabase()* ::>

Syntax for openDatabase() ::

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

Where flags = to control database access mode (OPEN_READWRITE /OPEN_READONLY/ CREATE_IF_NECESSARY)

Syntax for openorCreateDatabase() ::

Database filename

Modes: MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE and

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

Creating Table, View for Application

db.execSQL("Sqlite create Table Statement/ View Statement");

Examp:

1) for creating Table:----- >

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

or you can defined table as;

➢ ***db.execSQL("CREATE TABLE " + MemberTable + "(" + colID + " INTEGER PRIMARY KEY AUTOINCREMENT," + colName + " TEXT, " + colImg + " BLOB); ");***

❖The field **recIDis 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,…).***

❖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.***

Creating Table, View for Application

Example: -- for creating Views ----- >

```
db.execSQL("CREATE VIEW " + viewMember + " AS  SELECT "
        + MemberTable + "." + colID + " AS  _id," + " “
       +  MemberTable + "."+ colName + ","  + " "
       +MemberTable + "."+ colGender + "," + " "
       + " FROM " + MemberTable+ "");
```

- Creating view is particularly useful for security purpose of database. when you have to hide some of columns of created table.

Inserting values in Table

1> db.execSQL("sql insertion statement");

db.execSQL(insert into MemberTable values ('Ankit', 'Male'););

OR

2> db.**insert(String table,**String nullCol,**ContentValues** values);

➢***ContentValues conv= new ContentValues();***
conv.put("name", "ABC");
conv.put("Gender", "Male");
db.insert("MemberTable", null, conv);

- Put() method of contentvalues' object is used for mapping for column and for entering value.
- string null col is used for default value of that column where no value is entered.
- The insert method returns the id of row just inserted or -1 if there was an error during insertion

Update Table

- ***db.execSQL("update table statement");*** ***OR***

- ***db. update(String tab,ContentValues val,String whereClaus, String[] whereArgs)***

Exam:

- **db.execSQL("update MemberTable set name = (name || 'XXX') where phone >= '001' ");**

Or it can be written as;

- ***String [] whereArgs= {"2", "7"};***
  ContentValues updValues= new ContentValues();
  updValues.put("name", "Maria");
  db.update("tblAMIGO", updValues, "recID> ? and recID< ?", whereArgs);

Delete and Drop command for Table

- **db.execSQL("Delete or drop sqlite command"); OR**
- **db. delete(String table, String whereClause, String[] whereArgs) ;**

Exmp:

```
String [] whereArgs= {"2", "7"};
db.delete("tblAMIGO","recID> ? and recID< ?",whereArgs);
```

Query with Table

1.Retrieval queries are SQL-select statements.

2.Answers produced by retrieval queries are always held in an output table.

3.In order to process the *resulting rows, the user should provide a cursor device. Cursors allow a row-by-row accessof the records returned by the retrieval queries.*

SQLiteDatabase offers two main methods for phrasing SQL-select statements:

- ***rawQuery***
- ***query***

Both return a database cursor.

1.Raw queries 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).*

2.Simple queries are compact parametized select-like statements that operate on a single table (for developers who prefer not to use SQL).

Raw Queries:: >

```
String mySQL= "select count(*) as Total "
+ " from tblAmigo"
+ " where recID> ?"
+ " and name = ?";
String[] args= {"1", "BBB"};
Cursor c1 = db.rawQuery(mySQL, args);
```

Simple Queries :: >

```
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

CONTENT PROVIDER

Content Provider

- A *SQLite* database is private to the application which creates it.
- If you want to share data with other applications you can use a *content provider*.
- To create your own ContentProvider you have to define a class which extends android.content.ContentProvider
- The ContentProvider must implement six methods : query(), insert(), update(), delete(), getType() and onCreate().

Content Providers: Methods

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

Content Provider in Manifest XML

- Like Activity and Service components, a subclass of ContentProvider must be defined in the manifest file for its application, using the <provider> element.
- Authority (android:authorities): Symbolic names that identify the entire provider within the system.
- Provider class name (android:name): The class that implements ContentProvider

```
<provider
      android:authorities="in.sisoft.android.todos.contentprovider"
       android:name=".contentprovider.MyTodoContentProvider" >
</provider>
```

Content Resolver

•The Content Resolver is the single, global instance in your application that provides access to your (and other applications') content providers
•it accepts requests from clients, and resolves these requests by directing them to the content provider with a distinct authority
•To do this, the Content Resolver stores a mapping from authorities to Content Providers.
•The Content Resolver includes the CRUD (create, read, update, delete) methods corresponding to the abstract methods (insert, delete, query, update) in the Content Provider class
•**insert(Uri url, ContentValues values)**
Inserts a row into a table at the given URL.
•**query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)**
Query the given URI, returning a Cursor over the result set.
•**update(Uri uri, ContentValues values, String where, String[]selectionArgs)**
Update row(s) in a content URI.
•**delete(Uri url, String where, String[] selectionArgs)**
Deletes row(s) specified by a content URI.

Content Resolver

Android itself includes content providers that manage data such as audio, video, images, and personal contact information

Content provider	Intended data
Browser	Browser Bookmarks, Browser History etc
CallLog	Missed call, call details
Contacts	Contact details
MediaStore	Media files (aud/ved/img etc)
Settings	Device setting & preference

Detailed List:

http://developer.android.com/reference/android/provider/package-summary.html

A simple example for using callLog Provider

```
public class CallLogExampleActivity extends Activity {

    TextView textView;
    String typeString;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        textView = (TextView)findViewById(R.id.text_view);

        ContentResolver contentResolver = getContentResolver();
        // URI to call-log data store
        Uri callLogUri = CallLog.Calls.CONTENT_URI;

        // query for number, duration, and type (incoming/outgoing/missed)
        String[] columnsOfInterest = {CallLog.Calls.NUMBER,
                    CallLog.Calls.DURATION, CallLog.Calls.TYPE};
        // query the call log, for three columns of interest, sorting by duration descending
        Cursor cursor = contentResolver.query(callLogUri, columnsOfInterest,
                    null, null, CallLog.Calls.DURATION + " DESC");

        if (cursor.moveToFirst()) { // if anything was retrieved
            do {    // access number, duration, type; change output format based on type
                String number = cursor.getString(cursor.getColumnIndex(CallLog.Calls.NUMBER));
                int duration = cursor.getInt(cursor.getColumnIndex(CallLog.Calls.DURATION));
                int type = cursor.getInt(cursor.getColumnIndex(CallLog.Calls.TYPE));
                if (type == CallLog.Calls.INCOMING_TYPE)
                    textView.setText(textView.getText() +
                            "\nCall from " + number + " for " + duration + " seconds.");
                else if (type == CallLog.Calls.OUTGOING_TYPE)
                    textView.setText(textView.getText() +
                            "\nCall to " + number + " for " + duration + " seconds.");
                else
                    textView.setText(textView.getText() +
                            "\nMissed call from: " + number + ".");
            }
            while (cursor.moveToNext());
        }
    }
}
```