

Android Application Development, COMP 10073

Mohawk College, Winter 2021

Android SQLite

<https://www.journaldev.com/9438/android-sqlite-database-example-tutorial>

Android SQLite is a very lightweight database which comes with Android OS. Android SQLite combines a clean SQL interface with a very small memory footprint and decent speed. For Android, SQLite is “baked into” the Android runtime, so every Android application can create its own SQLite databases.

Android SQLite native API is not JDBC (Java API that can access any kind of tabular data), as JDBC might be too much overhead for a memory-limited smartphone. Once a database is created successfully its located in `data/data//databases/` accessible from Android Device Monitor.

SQLite is a typical relational database, containing tables (which consists of rows and columns), indexes etc. We can create our own tables to hold the data accordingly. This structure is referred to as a schema.

SQLiteOpenHelper

<https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper>

A helper class to manage database creation and version management. it handles 2 common problems: When the application runs the first time – At this point, we do not yet have a database. So we will have to create the tables, indexes, starter data, and so on.

When the application is upgraded to a newer schema – Our database will still be on the old schema from the older edition of the app. We will have option to alter the database schema to match the needs of the rest of the app.

You create a subclass implementing `onCreate(SQLiteDatabase)`, `onUpgrade(SQLiteDatabase, int, int)` and optionally `onOpen(SQLiteDatabase)`, and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state.

```
public SQLiteOpenHelper (Context context,
                        String name,
                        SQLiteDatabase.CursorFactory factory,
                        int version)
```

Create a helper object to create, open, and/or manage a database. This method always returns very quickly. The database is not actually created or opened until one of `getWritableDatabase()` or `getReadableDatabase()` is called.

context Context: used for locating paths to the the database. Can be null.

name String: database file name, null for an in-memory database.

factory SQLiteDatabase.CursorFactory: to use for creating cursor objects, or null for the default This value may be null.

version int: number of the database (starting at 1); if the database is older, `onUpgrade(SQLiteDatabase, int, int)` will be used to upgrade the database; if the database is newer, `onDowngrade(SQLiteDatabase, int, int)` will be used to downgrade the database.

MyDbHelper.java

```
public class MyDbHelper extends SQLiteOpenHelper {

    public static final String TAG = "MyDbHelper";

    /** collection of DataBase field names */
    public static final String DATABASE_FILE_NAME = "MyDatabase.db";
    public static final int DATABASE_VERSION = 1;
    public static final String MYTABLE = "mytable";
    public static final String ID = "_id";
    public static final String TITLE = "title";
    public static final String SUBTITLE = "subtitle";

    // This is the SQL Statement that will be executed to create the table and columns
    // "_id" is recommended to be a standard first column and primary key
    private static final String SQL_CREATE =
        "CREATE TABLE " + MYTABLE + " ( " + ID + " INTEGER PRIMARY KEY, " +
        TITLE + " TEXT, " + SUBTITLE + " TEXT) ";

    public MyDbHelper(Context context) {
        super(context, DATABASE_FILE_NAME, null, DATABASE_VERSION);
        Log.d(TAG, "constructor");
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        Log.d(TAG, "onCreate " + SQL_CREATE);
        db.execSQL(SQL_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // This is only called if the DATABASE_VERSION changes
        // Possible actions - delete table (ie DROP TABLE IF EXISTS mytable), then call onCreate
    }
}
```

activity_main.xml

<https://developer.android.com/guide/topics/ui/layout/relative>

RelativeLayout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left or center).

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Database Example"
        android:id="@+id/textView2" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText"
        android:hint="Enter Title"
        android:layout_below="@+id/textView2"
        android:layout_centerHorizontal="true" />

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/editText2"
        android:hint="Enter Subtitle"
        android:layout_below="@+id/editText"
        android:layout_centerHorizontal="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add to Database"
        android:onClick="addData"
        android:id="@+id/button"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show List"
        android:onClick="callList"
        android:id="@+id/button2"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        />

</RelativeLayout>
```



getWritableDatabase

[https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper#getWritableDatabase\(\)](https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper#getWritableDatabase())

The main activity creates a database, adds entries and launches a separate activity for displaying the database items.

```
// Create a global instance of our SQL Helper class
MyDbHelper mydbhelp = new MyDbHelper(this);

// Get an instance of the database using our helper class
SQLiteDatabase db = mydbhelp.getWritableDatabase();
```

The method `getWritableDatabase` creates and/or open a database that will be used for reading and writing. The first time this is called, the database will be opened and `onCreate(SQLiteDatabase)`, `onUpgrade(SQLiteDatabase, int, int)` and/or `onOpen(SQLiteDatabase)` will be called.

Once opened successfully, the database is cached, so you can call this method every time you need to write to the database. (Make sure to call `close()` when you no longer need the database.) Errors such as bad permissions or a full disk may cause this method to fail, but future attempts may succeed if the problem is fixed.

ContentValues

<https://developer.android.com/reference/android/content/ContentValues>

The `ContentValues` class is used to store a set of values that the `ContentResolver` can process. It provides a number of `get` and `put` methods along with some additional facilities like `search`, and `hash`.

```
// ContentValues provides a helper function to add our values
ContentValues values = new ContentValues();

// Put values using column name and value
values.put(MyDbHelper.TITLE, title.getText().toString());
values.put(MyDbHelper.SUBTITLE, subtitle.getText().toString());
```

SQL As Understood By SQLite

https://sqlite.org/lang_insert.html

The insert command creates one or more new rows in an existing table. Each of the named columns of the new row is populated with the results of evaluating the corresponding VALUES expression. Table columns that do not appear in the column list are populated with the default column value (specified as part of the CREATE TABLE statement), or with NULL if no default value is specified.

```
// The db.insert command will do a SQL insert on our table,  
// return the new row ID  
long newrowID = db.insert(MyDbHelper.MYTABLE, null, values);
```

Sample operations

Assumes string constants are defined in MyDbHelper class

Insert Records in an Android SQLite database

```
public void insert(String name, String desc) {  
    ContentValues contentValues = new ContentValues();  
    contentValues.put(MyDbHelper.TITLE, name);  
    contentValues.put(MyDbHelper.SUBTITLE, desc);  
    db.insert(MyDbHelper.MYTABLE, null, contentValues);  
}
```

Updates Records in an Android SQLite database

```
public int update(long _id, String name, String desc) {  
    ContentValues contentValues = new ContentValues();  
    contentValues.put(MyDbHelper.TITLE, name);  
    contentValues.put(MyDbHelper.SUBTITLE, desc);  
    int i = db.update(MyDbHelper.MYTABLE,  
        contentValues, mydbhelp._ID + " = " + _id, null);  
    return i;  
}
```

Deletes Records in an Android SQLite database

```
public void delete(long _id) {  
    db.delete(MyDbHelper.MYTABLE,  
        MyDbHelper.ID + " = " + _id, null);  
}
```

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    public static final String TAG = "==MainActivity==";
    // Create a global instance of our SQL Helper class
    MyDbHelper mydbhelp = new MyDbHelper(this);

    /**
     * onCreate (default)
     * @param savedInstanceState - (default)
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(TAG, "onCreate");
    }
    /**
     * onClick handler, adds a data base row consisting of two fields
     * @param view - button view (unused)
     */
    public void addData(View view) {

        // Get an instance of the database using our helper class
        SQLiteDatabase db = mydbhelp.getWritableDatabase();

        EditText title = (EditText) findViewById(R.id.editText);
        EditText subtitle = (EditText) findViewById(R.id.editText2);

        // A ContentValues class provides an easy helper function to add our values
        ContentValues values = new ContentValues();

        // Similar to using a bundle - put values using column name and value
        values.put(MyDbHelper.TITLE, title.getText().toString());
        values.put(MyDbHelper.SUBTITLE, subtitle.getText().toString());

        // The db.insert command will do a SQL insert on our table, return the new row ID
        long newrowID = db.insert(MyDbHelper.MYTABLE, null, values);
        Log.d(TAG, "New ID " + newrowID);

        // Clear out fields for next entry
        title.setText("");
        subtitle.setText("");
    }

    /**
     * onClick handler, launches our List Activity
     * @param view - button view (unused)
     */
    public void callList(View view) {
        Intent intent = new Intent(this, ListActivity.class);
        startActivity(intent);
        Log.d(TAG, "callList");
    }
}
```

Cursor

<https://developer.android.com/reference/android/database/Cursor>

This interface provides random read-write access to the result set returned by a database query. A Cursor represents the entire result set of the query. Once the query is fetched a call to `cursor.moveToFirst()` is made. Calling `moveToFirst()` does two things:

- It allows us to test whether the query returned an empty set (by testing the return value)
- It moves the cursor to the first result (when the set is not empty)

The following code is used to fetch all records:

```
public Cursor fetch() {
    String[] columns = new String[] { MyDbHelper.ID,
                                      MyDbHelper.TITLE, MyDbHelper.SUBTITLE };
    Cursor cursor = db.query(MyDbHelper.MYTABLE, columns,
                            null, null, null, null, null);
    if (cursor != null) {
        cursor.moveToFirst();
    }
    return cursor;
}
```

Another way to use a Cursor is to wrap it in a CursorAdapter. Just as ArrayAdapter adapts arrays, CursorAdapter adapts Cursor objects, making their data available to an AdapterView like a ListView.

activity_list.xml

For this demonstration we will use an activity that uses a simple TextView for output. A ListView is a better alternative, see the homework.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/outputtext"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true" />
</RelativeLayout>
```

ListActivity.java

```
public class ListActivity extends AppCompatActivity {
    public static final String TAG = "ListActivity";
    // Create a global instance of our SQL Helper class
    MyDbHelper mydbhelp = new MyDbHelper(this);

    /**
     * onCreate - get an instance of our database, use a cursor to display the values
     * @param savedInstanceState (default)
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list);
        Log.d(TAG, "onCreate");

        // Get an instance of the database using our helper class
        SQLiteDatabase db = mydbhelp.getReadableDatabase();

        // A projection defines what fields we want to retrieve.
        String[] projection = { MyDbHelper.ID, MyDbHelper.TITLE, MyDbHelper.SUBTITLE};

        // db.query will retrieve the data and return a Cursor to access it
        Cursor mycursor = db.query(MyDbHelper.MYTABLE, projection, null,
            null, null, null, null);

        String results = "";

        if (mycursor != null) {
            // Loop through our returned results from the start
            while (mycursor.moveToNext()) {
                Log.d(TAG, "found DB item");

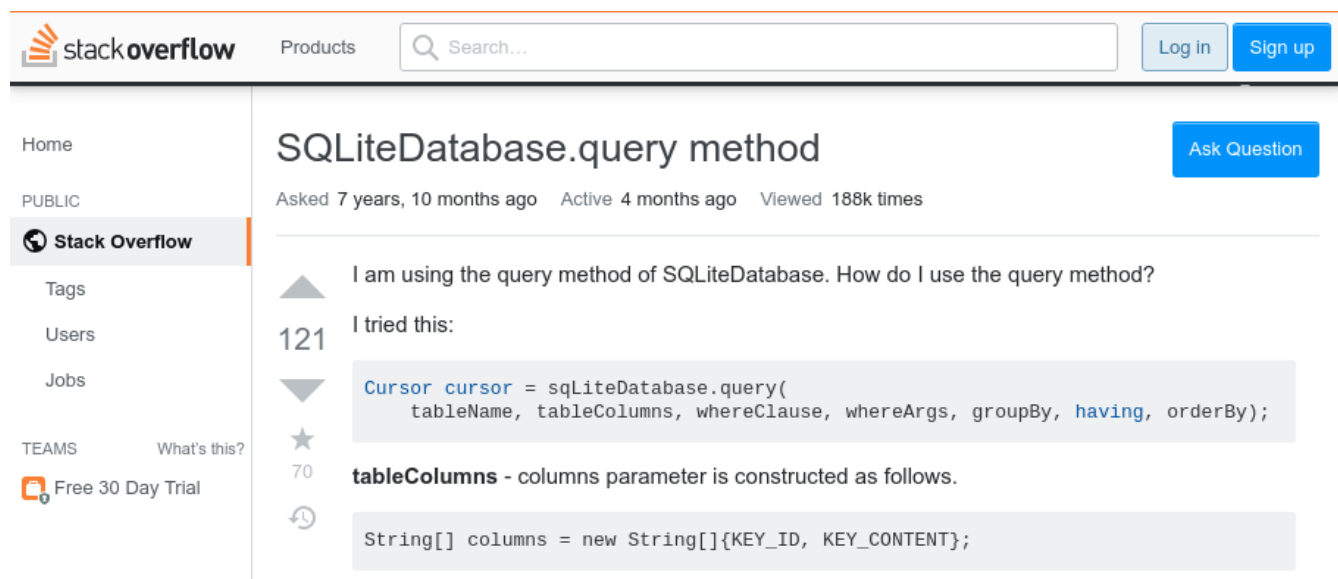
                String title = mycursor.getString(
                    mycursor.getColumnIndex(MyDbHelper.TITLE));
                String subtitle = mycursor.getString(
                    mycursor.getColumnIndex(MyDbHelper.SUBTITLE));
                long itemID = mycursor.getLong(
                    mycursor.getColumnIndex(MyDbHelper.ID));

                // We could add our results to an array, or process them here if we want
                results += itemID + " " + title + " " + subtitle + "\n"; // XXX hackish
            }

            // Close the cursor when we're done
            mycursor.close();
        }
        if (results == "") {
            results = "!! no data !!";
        }
        // Show our results
        TextView output = (TextView) findViewById(R.id.outputtext);
        output.setText(results); // XXX TODO fix this
    }
}
```


The DB Query

<https://stackoverflow.com/questions/10600670/sqlitedatabase-query-method>



The screenshot shows the Stack Overflow interface. The header includes the Stack Overflow logo, a search bar, and 'Log in' and 'Sign up' buttons. The left sidebar contains navigation links: Home, PUBLIC, Stack Overflow (selected), Tags, Users, Jobs, TEAMS, What's this?, and a 'Free 30 Day Trial' badge. The main content area displays a question titled 'SQLiteDatabase.query method' with a blue 'Ask Question' button. Below the title, it says 'Asked 7 years, 10 months ago', 'Active 4 months ago', and 'Viewed 188k times'. The question text is 'I am using the query method of SQLiteDatabase. How do I use the query method?'. It has 121 votes and 70 answers. The first answer shows a code snippet:

```
Cursor cursor = SQLiteDatabase.query(
    tableName, tableColumns, whereClause, whereArgs, groupBy, having, orderBy);
```

 and a comment: 'tableColumns - columns parameter is constructed as follows.' followed by another code snippet:

```
String[] columns = new String[]{KEY_ID, KEY_CONTENT};
```

tableColumns

- `null` for all columns as in `SELECT * FROM ...`
- `new String[] { "column1", "column2", ... }` for specific columns as in `SELECT column1, column2 FROM ...` - also put complex expressions here:
 - `new String[] = { "(SELECT max(column1) FROM table1) AS max" }` would give you a column named max holding the max value of column1

whereClause

- the part you put after `WHERE` without that keyword, e.g. `"column1 > 5"`
- should include `?` for things that are dynamic, e.g. `"column1=?"` -> see `whereArgs`

whereArgs

- specify the content that fills each `?` in whereClause in the order they appear

the others

- just like `whereClause` the statement after the keyword or `null` if you don't use it.

Full Example Query

The following Android code:

```
String[] tableColumns = new String[] {
    "column1",
    "(SELECT max(column1) FROM table2) AS max"};
String whereClause = "column1 = ? OR column1 = ?";
String[] whereArgs = new String[] {
    "value1",
    "value2"};
String orderBy = "column1";
Cursor c = SQLiteDatabase.query("table1", tableColumns,
    whereClause, whereArgs, null, null, orderBy);

// since we have a named column we can do
int idx = c.getColumnIndex("max");
```

Is equivalent to the following raw query:

```
String queryString =
    "SELECT column1, (SELECT max(column1) FROM table1) AS max FROM table1 " +
    "WHERE column1 = ? OR column1 = ? ORDER BY column1";
SQLiteDatabase.rawQuery(queryString, whereArgs);
```