Android SQLite Database Tutorial

Android provides several ways to store user and app data. SQLite is one way of storing user data. SQLite is a very light weight database which comes with Android OS. In this tutorial I'll be discussing how to write classes to handle all SQLite operations.

Download Code

In this tutorial I am taking an example of storing user contacts in SQLite database. I am using a table called Contacts to store user contacts. This table contains three columns id (INT), name (TEXT), phone_number(TEXT).

Following is the structure of contacts table.

Table Structure AndroidHive

```
Table Name: Contacts

+------------------+--------------+------+
| Field            | Type         | Key  |
+------------------+--------------+------+
| id               | INT          | PRI  |
| name             | TEXT         |      |
| phone_number     | TEXT         |      |
+------------------+--------------+------+
```

Writing Contact Class

Before you go further you need to write your Contact class with all getter and setter methods to maintain single contact as an object.

Contact.java

```
package com.androidhive.androidsqlite;

public class Contact {

    //private variables
    int _id;
    String _name;
    String _phone_number;

    // Empty constructor
    public Contact(){

    }
    // constructor
    public Contact(int id, String name, String _phone_number){
        this._id = id;
        this._name = name;
        this._phone_number = _phone_number;
    }

    // constructor
    public Contact(String name, String _phone_number){
        this._name = name;
        this._phone_number = _phone_number;
    }
    // getting ID
    public int getID(){
        return this._id;
    }

    // setting id
    public void setID(int id){
        this._id = id;
    }

    // getting name
    public String getName(){
        return this._name;
    }

    // setting name
    public void setName(String name){
        this._name = name;
    }

    // getting phone number
    public String getPhoneNumber(){
        return this._phone_number;
    }

    // setting phone number
    public void setPhoneNumber(String phone_number){
        this._phone_number = phone_number;
```

```
    }
}
```

Writing SQLite Database Handler Class

We need to write our own class to handle all database CRUD(Create, Read, Update and Delete) operations.

1. Create a new project by going to File New Android Project.
2. Once the project is created, create a new class in your project src directory and name it as `DatabaseHandler.java` (Right Click on src/package New Class)
3. Now extend your DatabaseHandler.java class from SQLiteOpenHelper.

```
public class DatabaseHandler extends SQLiteOpenHelper {
```

4. After extending your class from SQLiteOpenHelper you need to override two methods onCreate() and onUpgrage()

`onCreate()` – These is where we need to write create table statements. This is called when database is created.

`onUpgrade()` – This method is called when database is upgraded like modifying the table structure, adding constraints to database etc.,

```
public class DatabaseHandler extends SQLiteOpenHelper {

    // All Static variables
    // Database Version
    private static final int DATABASE_VERSION = 1;

    // Database Name
    private static final String DATABASE_NAME = "contactsManager";

    // Contacts table name
    private static final String TABLE_CONTACTS = "contacts";

    // Contacts Table Columns names
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_PH_NO = "phone_number";

    public DatabaseHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    // Creating Tables
    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "("
                + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
                + KEY_PH_NO + " TEXT" + ")";
        db.execSQL(CREATE_CONTACTS_TABLE);
```

```
    }

    // Upgrading database
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);

        // Create tables again
        onCreate(db);
    }
```

All CRUD Operations (Create, Read, Update and Delete)

Now we need to write methods for handling all database read and write operations. Here we are implementing following methods for our contacts table.

```
// Adding new contact
public void addContact(Contact contact) {}

// Getting single contact
public Contact getContact(int id) {}

// Getting All Contacts
public List<Contact> getAllContacts() {}

// Getting contacts Count
public int getContactsCount() {}
// Updating single contact
public int updateContact(Contact contact) {}

// Deleting single contact
public void deleteContact(Contact contact) {}
```

Inserting new Record

The `addContact()` method accepts Contact object as parameter. We need to build ContentValues parameters using Contact object. Once we inserted data in database we need to close the database connection.

```
  addContact()
    // Adding new contact
public void addContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName()); // Contact Name
    values.put(KEY_PH_NO, contact.getPhoneNumber()); // Contact Phone Number

    // Inserting Row
    db.insert(TABLE_CONTACTS, null, values);
    db.close(); // Closing database connection
}
```

Reading Row(s)

The following method `getContact()` will read single contact row. It accepts id as parameter and will return the matched row from the database.

```
  getContact()
    // Getting single contact
public Contact getContact(int id) {
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.query(TABLE_CONTACTS, new String[] { KEY_ID,
            KEY_NAME, KEY_PH_NO }, KEY_ID + "=?",
            new String[] { String.valueOf(id) }, null, null, null, null);
    if (cursor != null)
        cursor.moveToFirst();

    Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),
            cursor.getString(1), cursor.getString(2));
    // return contact
    return contact;
}
```

`getAllContacts()` will return all contacts from database in array list format of Contact class type. You need to write a for loop to go through each contact.

```
  getAllContacts()
    // Getting All Contacts
public List<Contact> getAllContacts() {
    List<Contact> contactList = new ArrayList<Contact>();
    // Select All Query
    String selectQuery = "SELECT  * FROM " + TABLE_CONTACTS;

    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            Contact contact = new Contact();
            contact.setID(Integer.parseInt(cursor.getString(0)));
            contact.setName(cursor.getString(1));
            contact.setPhoneNumber(cursor.getString(2));
            // Adding contact to list
            contactList.add(contact);
        } while (cursor.moveToNext());
    }

    // return contact list
    return contactList;
}
```

`getContactsCount()` will return total number of contacts in SQLite database.

```
  getContactsCount()
// Getting contacts Count
    public int getContactsCount() {
        String countQuery = "SELECT  * FROM " + TABLE_CONTACTS;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(countQuery, null);
        cursor.close();

        // return count
        return cursor.getCount();
    }
```

Updating Record

`updateContact()` will update single contact in database. This method accepts Contact class object as parameter.

```
  updateContact()
    // Updating single contact
public int updateContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NAME, contact.getName());
    values.put(KEY_PH_NO, contact.getPhoneNumber());

    // updating row
    return db.update(TABLE_CONTACTS, values, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
}
```

Deleting Record

`deleteContact()` will delete single contact from database.

```
  deleteContact()
    // Deleting single contact
public void deleteContact(Contact contact) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete(TABLE_CONTACTS, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
    db.close();
}
```

Complete DatabaseHandler.java Code:

DatabaseHandler.java

```java
package com.androidhive.androidsqlite;

import java.util.ArrayList;
import java.util.List;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHandler extends SQLiteOpenHelper {

    // All Static variables
    // Database Version
    private static final int DATABASE_VERSION = 1;

    // Database Name
    private static final String DATABASE_NAME = "contactsManager";

    // Contacts table name
    private static final String TABLE_CONTACTS = "contacts";

    // Contacts Table Columns names
    private static final String KEY_ID = "id";
    private static final String KEY_NAME = "name";
    private static final String KEY_PH_NO = "phone_number";

    public DatabaseHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    // Creating Tables
    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTACTS + "("
                + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
                + KEY_PH_NO + " TEXT" + ")";
        db.execSQL(CREATE_CONTACTS_TABLE);
    }

    // Upgrading database
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);

        // Create tables again
        onCreate(db);
    }

    /**
     * All CRUD(Create, Read, Update, Delete) Operations
     */

    // Adding new contact
    void addContact(Contact contact) {
        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();
        values.put(KEY_NAME, contact.getName()); // Contact Name
```

```
        values.put(KEY_PH_NO, contact.getPhoneNumber()); // Contact Phone

        // Inserting Row
        db.insert(TABLE_CONTACTS, null, values);
        db.close(); // Closing database connection
    }

    // Getting single contact
    Contact getContact(int id) {
        SQLiteDatabase db = this.getReadableDatabase();

        Cursor cursor = db.query(TABLE_CONTACTS, new String[] { KEY_ID,
                KEY_NAME, KEY_PH_NO }, KEY_ID + "=?",
                new String[] { String.valueOf(id) }, null, null, null, null);
        if (cursor != null)
            cursor.moveToFirst();

        Contact contact = new Contact(Integer.parseInt(cursor.getString(0)),
                cursor.getString(1), cursor.getString(2));
        // return contact
        return contact;
    }

    // Getting All Contacts
    public List<Contact> getAllContacts() {
        List<Contact> contactList = new ArrayList<Contact>();
        // Select All Query
        String selectQuery = "SELECT  * FROM " + TABLE_CONTACTS;

        SQLiteDatabase db = this.getWritableDatabase();
        Cursor cursor = db.rawQuery(selectQuery, null);

        // looping through all rows and adding to list
        if (cursor.moveToFirst()) {
            do {
                Contact contact = new Contact();
                contact.setID(Integer.parseInt(cursor.getString(0)));
                contact.setName(cursor.getString(1));
                contact.setPhoneNumber(cursor.getString(2));
                // Adding contact to list
                contactList.add(contact);
            } while (cursor.moveToNext());
        }

        // return contact list
        return contactList;
    }

    // Updating single contact
    public int updateContact(Contact contact) {
        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();
        values.put(KEY_NAME, contact.getName());
        values.put(KEY_PH_NO, contact.getPhoneNumber());

        // updating row
        return db.update(TABLE_CONTACTS, values, KEY_ID + " = ?",
                new String[] { String.valueOf(contact.getID()) });
    }

    // Deleting single contact
    public void deleteContact(Contact contact) {
```

```
        SQLiteDatabase db = this.getWritableDatabase();
        db.delete(TABLE_CONTACTS, KEY_ID + " = ?",
                new String[] { String.valueOf(contact.getID()) });
        db.close();
    }

    // Getting contacts Count
    public int getContactsCount() {
        String countQuery = "SELECT  * FROM " + TABLE_CONTACTS;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(countQuery, null);
        cursor.close();

        // return count
        return cursor.getCount();
    }

}
```

Usage:

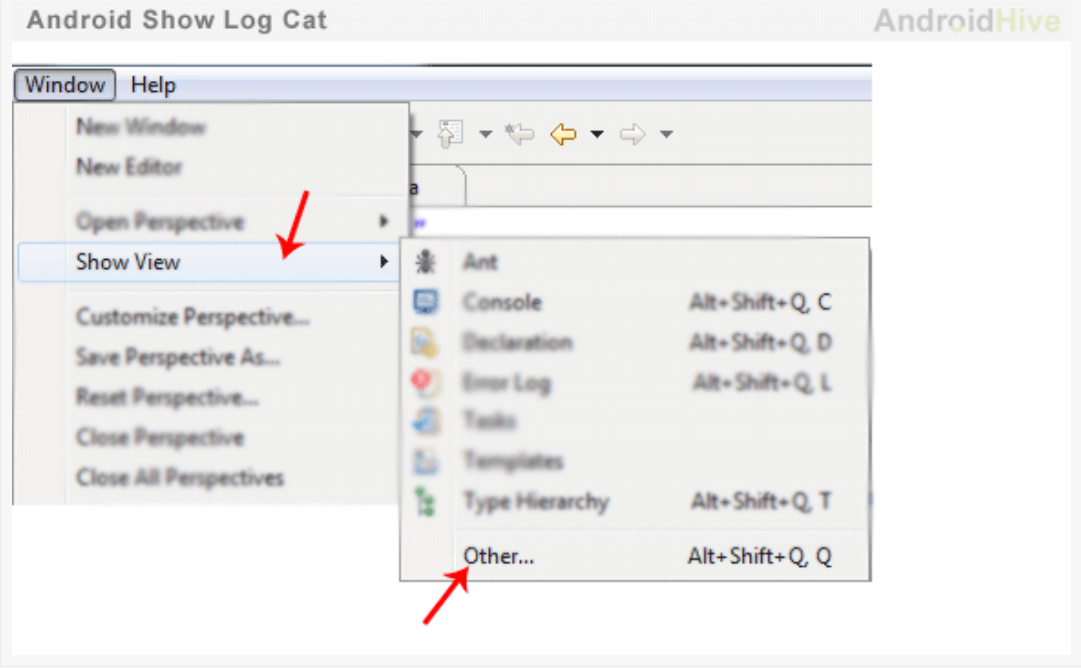
AndroidSQLiteTutorialActivity

```
package com.androidhive.androidsqlite;

import java.util.List;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class AndroidSQLiteTutorialActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        DatabaseHandler db = new DatabaseHandler(this);

        /**
         * CRUD Operations
         * */
        // Inserting Contacts
        Log.d("Insert: ", "Inserting ..");
        db.addContact(new Contact("Ravi", "9100000000"));
        db.addContact(new Contact("Srinivas", "9199999999"));
        db.addContact(new Contact("Tommy", "9522222222"));
        db.addContact(new Contact("Karthik", "9533333333"));

        // Reading all contacts
        Log.d("Reading: ", "Reading all contacts..");
        List<Contact> contacts = db.getAllContacts();

        for (Contact cn : contacts) {
            String log = "Id: "+cn.getID()+" ,Name: " + cn.getName() + " ,Phone: " + c
                // Writing Contacts to log
        Log.d("Name: ", log);
    }
    }
```

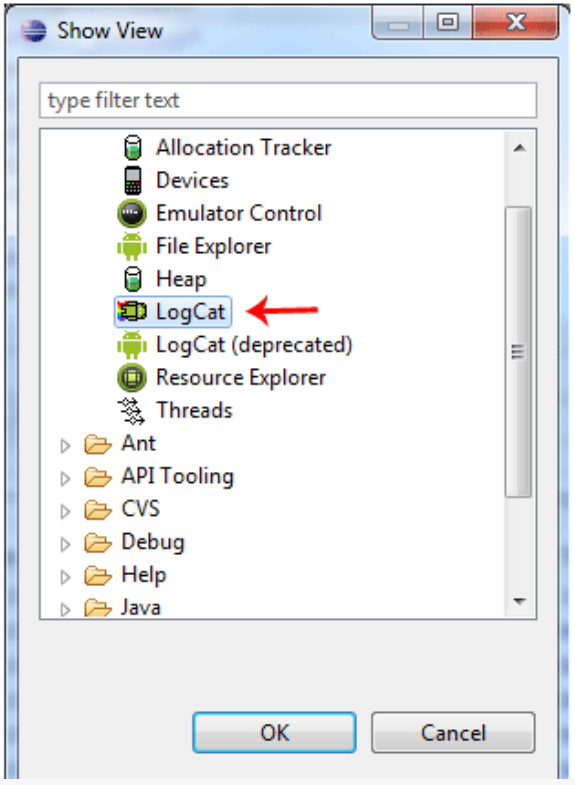
```
}
```

Android Log Cat Report:

I am writing output to Log report. You can see your log report by going to Windows Show View Other.. Android Log Cat.



SQLite Output Log AndroidHive



SQLite Output Log AndroidHive

LogCat

ccepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.

	PID	Application	Tag	Text
...	371	com.androidhive...	Insert:	Inserting ..
...	371	com.androidhive...	Reading:	Reading all contacts..
...	371	com.androidhive...	Name:	Id: 1 ,Name: Ravi ,Phone: 910000000
...	371	com.androidhive...	Name:	Id: 2 ,Name: Srinivas ,Phone: 91999
...	371	com.androidhive...	Name:	Id: 3 ,Name: Tommy ,Phone: 95222222
...	371	com.androidhive...	Name:	Id: 4 ,Name: Karthik ,Phone: 953333

Add New Comment

Login

Type your comment here.

Showing 20 of 233 comments

Sort by popular now

A guest

Hello
Good example :) like that all the db handling is in one single file.
Some question tho about closing for db and cursor.

1 - For e.g. "addContact" you open a db for writing so you close it in the end [db.close();].
But for "updateContact" you don't, why?
Is the db closed "automatically" when it is "return:ed" [return db.update(...);] or something?

2a - When should the cursor be closed?
Why is it only closed in "getContactsCount" but not in the other methods that reads from the db?
2b - and why could a closed cursor be return:ed [return cursor.getCount();]?

3 - why are the db.update(...); return:ed in "updateContact" but db.insert(...); is not in "addContact"? What do the return do here? Isn't the "result" type the same, just data "added" in both?

Thanks in advance

9 months ago 6 Likes Like Reply

Jonathan Bruce

Thanks for this tutorial, it's helped a lot, but I'd also like to know these answers. Logcat reports
close() was never explicitly called on database ...
android.database.sqlite.DatabaseObjectNotClosedException: Application did not close the cursor or database object that was opened here

Which I believe is related. Thanks.

7 months ago in reply to A guest 1 Like Like Reply

Ravi Tamada

I forgot to close database connection in one function. Check the code in Sqlite hanlder class. You need to use .close() method.

7 months ago in reply to Jonathan Bruce 3 Likes Like Reply

Vlad Turbuleasa

to close the DB we must get a int variable and store the row affected by the db.update() , am i right? Thanks for the tutorial!

4 months ago in reply to Ravi Tamada Like Reply

Kyle Beckman43

Great examples, they helped a TON, but I was also curious about these same issue.

8 months ago in reply to A guest Like Reply

Guest

Why is "get all contacts" writing data, instead of reading?

12 months ago 5 Likes Like Reply

Ravi Tamada

Ya it is wrong . It should be readable.

11 months ago in reply to Guest 3 Likes Like Reply

Eric Itzhak

I think you should change the code and fix the stuff that's wrong, for future copy-pasters.

7 months ago in reply to Ravi Tamada 5 Likes Like Reply

Guest

I think he shouldn't !:))

4 months ago in reply to Eric Itzhak 1 Like Like Reply

Karan

Great tutorial! Clear cut.
Although, has the code been updated with the correct version?

5 months ago in reply to Ravi Tamada 1 Like Like Reply

Guest

It's because the SQLite DB isn't storing the objects directly, just their information. So when you return the data, you have to reconstruct the objects before you return them. I think...

10 months ago in reply to Guest Like Reply

A guest

how to do a upgrade??

6 months ago 4 Likes Like Reply

guest

change the database version

3 months ago in reply to A guest 2 Likes Like Reply

Roger

Thanks for helpful tutorial! may this will be also helpful http://www.enterra-inc.com/tec...

6 months ago 3 Likes Like Reply

Momin Ayesha

what are the controls i have to put it on the emulator

7 months ago 1 Like Like Reply

newbie

sometimes I see people return value without 'this'

```
public long getId() {
return id;
}
```

from http://www.vogella.de/articles...

what is the difference between them, if we add 'this' and if we didn't add 'this'

10 months ago 1 Like Like Reply

David Crow

See here: http://en.wikipedia.org/wiki/T...

3 days ago in reply to newbie Like Reply

Ravi Tamada

'this' - if you this key work you are pointing a variable which is private to that class only. If you have same name for class local variable and incoming parameters then if you want use the variable of class then you need to use 'this' key word

10 months ago in reply to newbie Like Reply

Raniqueeen

Hai,Ravi...
Could please tell me how to make a customized listview in android..
iam newer to android..
thanks..

10 months ago 1 Like Like Reply

Ravi Tamada

I already wrote tutorial
http://www.androidhive.info/20...

10 months ago in reply to Raniqueeen Like Reply

Subscribe by email RSS

Load more comments

02799882