SQLITE

Android has built in SQLite database implementation. It is available locally over the device (mobile & tablet) and contain data in text format. It carries light weight data and suitable with many languages. So, it doesn't require any administration or setup procedure of the database. Important Note – The database created is saved in a **directory: data/data/APP_Name/databases/DATABASE_NAME**. (1)

Creating And Updating Database In Android

For creating, updating and other operations you need to create a subclass or SQLiteOpenHelper class. SQLiteOpenHelper is a helper class to manage database creation and version management. It provides two methods onCreate(SQLiteDatabase db), onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion).

The SQLiteOpenHelper is responsible for opening database if exist, creating database if it does not exist and upgrading if required. The SQLiteOpenHelper only require the DATABASE_NAME to create database. After extending SQLiteOpenHelper you will need to implement its methods onCreate, onUpgrade and constructor.

onCreate(SQLiteDatabase sqLiteDatabase) method is called only once throughout the application lifecycle. It will be called whenever there is a first call to getReadableDatabase() or getWritableDatabase() function available in super SQLiteOpenHelper class. So SQLiteOpenHelper class call the onCreate() method after creating database and instantiate SQLiteDatabase object. Database name is passed in constructor call.

Create:

```
/**A helper class to perform database related queries*/


public class SqliteManager extends SQLiteOpenHelper {
public static final String DATABASE_NAME = "abhiandroid.db";
public static final int version = 1;
public SqliteManager(Context context) {
   super(context, DATABASE_NAME, null, version);
  }

@Override
public void onCreate(SQLiteDatabase sqLiteDatabase) {
   String dbQuery = "CREATE TABLE Items (id INTEGER PRIMARY KEY
   AUTOINCREMENT,name TEXT, description TEXT)";
   sqLiteDatabase.execSQL(dbQuery);
  }

@Override
public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {
  }
}
```

onUpgrade(SQLiteDatabase db,int oldVersion, int newVersion) is only called whenever there is a updation in existing version. So to update a version we have to increment the value of version variable passed in the superclass constructor.

In onUpgrade method we can write queries to perform whatever action is required. In most example you will see that existing table(s) are being dropped and again onCreate() method is being called to create tables again. But it's not mandatory to do so and it all depends upon your requirements.

We must change database version if we have added a new row in the database table. If we have requirement that we don't want to lose existing data in the table then we can write alter table query in the onUpgrade(SQLiteDatabase db,int oldVersion, int newVersion) method.

To perform **insert, read, delete, update** operation there are two different ways: (2)

- Write parameterized queries (Recommended)
- Write raw queries

Parameterized Queries: These are those queries which are performed using inbuilt functions to insert, read, delete or update data. These operation related functions are provided in SQLiteDatabase class.

Raw Queries: These are simple sql queries similar to other databases like MySql, Sql Server etc, In this case user will have to write query as text and passed the query string in rawQuery(String sql,String [] selectionArgs) or execSQL(String sql,Object [] bindArgs) method to perform operations.

Important Note: Android documentation don't recommend to use raw queries to perform insert, read, update, delete operations, always use SQLiteDatabase class's insert, query, update, delete functions.

Following is an example of **raw query** to **insert data**:

```
public void insertItem(Item item) {
  String query = "INSERT INTO " + ItemTable.NAME + " VALUES (0,?,?)";
  SQLiteDatabase db = getWritableDatabase();
  db.execSQL(query, new String[]{item.name, item.description});
  db.close();
}
```

While using raw queries we never come to know the result of operation, however with parameterized queries function a value is returned for success or failure of operation.

<u>Insert:</u> To perform insert operation using parameterized query we must call insert function available in SQLiteDatabase class. insert() function has three parameters like public long insert(String tableName,String nullColumnHack,ContentValues values) where tableName is name of table in which data to be inserted.

```
 public long insert(String tableName,String  nullColumnHack,ContentValues values)
```

NullColumnHack (string) may be passed null, it require table column value in case we don't put column name in ContentValues object so a null value must be inserted for that particular column, values are those values that needs to be inserted.

ContentValues is a key-pair based object which accepts all primitive type values so whenever data is being put in ContentValues object it should be put again table column name as key and data as value. insert function will return a long value i.e number of inserted row if successfully inserted, – 1 otherwise.

```
//Item is a class representing any item with id, name and description.
public void addItem(Item item) {
  SQLiteDatabase db = getWritableDatabase();
  ContentValues contentValues = new ContentValues();
  contentValues.put("name",item.name);
   // name - column
   contentValues.put("description",item.description);
   // description is column in items table, item.description has value for description
  db.insert("Items", null, contentValues);//Items is table name
   db.close();
}
```

<u>Update:</u> Update function is quite similar to insert but it requires two additional parameters, it doesn't required nullColumnHack. It has total four parameters two are similar to insert function that is tableName and contentValues. Another two are whereClause(String) and whereArgs(String[]). Update function is available in SQLiteDatabase class it looks as follows:

```
public int update(String tableName,ContentValues contentValues,String whereClause,Strin
g[] whereArgs)
```

Here whereClause is tell the database where to update data in table, it's recommended to pass ?s (questions) along with column name in whereClause String. Similarly whereArgs array will contain values for those columns whose against ?s has been put in whereClause. Update function will return number of rows affected if success, 0 otherwise.

```java
//Item is a class representing any item with id, name and description
public void updateItem(Item item) {
  SQLiteDatabase db = getWritableDatabase();
  ContentValues contentValues = new ContentValues();
  contentValues.put("id", item.id);
  contentValues.put("name", item.name);
  contentValues.put("description", item.description);
  String whereClause = "id=?";
  String whereArgs[] = {item.id.toString()};
  db.update("Items", contentValues, whereClause, whereArgs);
}
```

Delete: Similar to insert and update, delete function is available in SQLiteDatabase class, So delete is very similar to update function apart from ContentValues object as it's not required in delete. public int delete(String tableName,String whereClause,String [] whereArgs) function has three parameters these are totally similar to update function's parameters and are used in same way as in update function.

```java
public void deleteItem(Item item) {
SQLiteDatabase db = getWritableDatabase();
String whereClause = "id=?";
String whereArgs[] = {item.id.toString()};
db.delete("Items", whereClause, whereArgs);
}
```

Here whereClause is optional, passing null will delete all rows in table. delete function will return number of affected row if whereClause passed otherwise will return 0.

Important Note: If you want to remove all rows and require count of deleted ones also then pass 1 as whereClause.

Select: Reading from a database table is bit different from other functions like insert, update and delete. **SQLiteDatabase class provides query() method to read data from table**. query() method is **overloaded with different set of parameters**. It **returns Cursor object** so Cursor is a **result-set with queried data**, it provides different functions really helpful while reading data.

public Cursor query (String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

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

public Cursor query (boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)

Most of the parameters are optional except from table and distinct any of other parameters can be passed as null. if distinct is passed as true Cursor data set will not have any duplicate row.

```java
public ArrayList<Item> readAllItems() {
ArrayList<Item> items = new ArrayList<>();
SQLiteDatabase db = getReadableDatabase();
//see above point 2 function
Cursor cursor = db.query("Items"
, null// columns - null will give all
, null// selection
, null// selection arguments
, null// groupBy
, null// having
, null// no need or order by for now;
if (cursor != null) {
  while (cursor.moveToNext()) {
  // move the cursor to next row if there is any to read it's data
           Item item = readItem(cursor);
           items.add(item);
     }
  }
return items;
}

private Item readItem(Cursor cursor) {
  Item item = new Item();
  item.id = cursor.getInt(cursor.getColumnIndex(ItemTable.COL_ID));
  item.name = cursor.getString(cursor.getColumnIndex(ItemTable.COL_NAME));
  item.description = cursor.getString(cursor.getColumnIndex(ItemTable.COL_DESCRIPTION))
;
  return item;
}
```

In theory we have already created an SQLite database. So we want to use this database file with the Android project.
Instead of creating a new database, how can the application gain access to this database and use it as its database? (3)

NOTE: Before trying this code, please find this line in the below code:

```
private static String DB_NAME ="YourDbName"; // Database name
```

DB_NAME here is the name of your database. It is assumed that you have a copy of the database in the assets folder, so for example if your database name is ordersDB, then the value of DB_NAME will be ordersDB,

```
private static String DB_NAME ="ordersDB";
```

Keep the database in assets folder

(for creating asset folder File->New->Folders->Asset Folder you can place your pre-created db in there) **NEW -> FOLDER -> ASSETS FOLDER**

and then follow the below:

DataHelper class: *(3) -> 7 years ago*

New version (4) -> 5+ years ago

First, to use a database, in general, in android, you should extend the SQLiteOpenHelper class. This class is the one responsible for creating your database (and upgrading) when needed from a sql script you provide in your implementation.

So the trick is, you need to override the behavior of the SQLiteOpenHelper to copy your database file from the assets folder instead of create your database.

In this blog post (5 -> 2013), i explain in details the process of overriding this behavior. but here is the final code. Use the Repository class as you would use SQLiteOpenHelper normally.

Then I have found a more recent guide (6) 2+years ago.

Well, it's pretty easy, just put your database.db in assets folder and you can use Android SQLiteAssetHelper (https://github.com/jgilfelt/android-sqlite-asset-helper) to read and write the database, this library makes the process pretty easy and straightforward.

Import the library

```
compile 'com.readystatesoftware.sqliteasset:sqliteassethelper:+'
```

and then

```
public class MyDatabase extends SQLiteAssetHelper {

private static final String DATABASE_NAME = "northwind.db";
private static final int DATABASE_VERSION = 1;

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

}
```

That's all you need to do to access the database.

Example: https://github.com/jgilfelt/android-sqlite-asset-helper/tree/master/samples/database-v1/src/main/java/com/example

Useful guide:

Preparing the SQLite database file (7)

To create a SQLite DB I recommend using DB Browser for SQLite. Download and install. Create a database as a file, save it. (http://sqlitebrowser.org/)

Open your database and add a new table called "android_metadata", you can execute the following SQL statement to do it: (8)

```
CREATE TABLE "android_metadata" ("locale" TEXT DEFAULT 'en_US')
```

Now insert a single row with the text 'en_US' in the "android_metadata" table:

```
INSERT INTO "android_metadata" VALUES ('en_US')
```

Then, it is necessary to rename the primary id field of your tables to "_id" so Android will know where to bind the id field of your tables

```
}
```

After created the database put it in (app -> new assets folder)
https://www.youtube.com/watch?v=NzK5hHdblqk

Create a class with a simple button that when clicked reads and shows data:

```
//code from http://www.codebind.com/android-tutorials-and-examples/android-
sqlite-tutorial-example/
public void viewAll() {
    btnviewAll.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Cursor res = myDb.getAllData();
                    if(res.getCount() == 0) {
                        // show message
                        showMessage("Error","Nothing found");
                        return;
                    }

                    StringBuffer buffer = new StringBuffer();
                    while (res.moveToNext()) {
                        buffer.append("Id :"+ res.getString(0)+"\n");
                        buffer.append("Name :"+ res.getString(1)+"\n");
                        buffer.append("Room :"+ res.getString(2)+"\n");
                    }

                    // Show all data
                    showMessage("Data",buffer.toString());
                }
            }
    );
}


public void showMessage(String title,String Message){
    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setCancelable(true);
    builder.setTitle(title);
    builder.setMessage(Message);
    builder.show();
}
```

And on Database Helper (class on gitlab from http://www.codebind.com/android-tutorials-and-examples/android-sqlite-tutorial-example/(9))

```
public Cursor getAllData() {
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor res = db.rawQuery("select * from "+TABLE_NAME,null);
    return res;
}
```

using a row query to get data

1. https://abhiandroid.com/database/sqlite
2. https://abhiandroid.com/database/operation-sqlite.html
3. https://stackoverflow.com/questions/9109438/how-to-use-an-existing-database-with-an-android-application
4. https://stackoverflow.com/questions/16555315/how-to-use-existing-database-in-android
5. http://blog.kdehairy.com/using-a-preloaded-sqlite-database-with-sqliteopenhelper/
6. https://stackoverflow.com/questions/37135000/how-to-insert-existing-database-in-android-project
7. http://blog.harrix.org/article/6784
8. https://blog.reigndesign.com/blog/using-your-own-sqlite-database-in-android-applications/
9. http://www.codebind.com/android-tutorials-and-examples/android-sqlite-tutorial-example/