Android SQLite Database

Prashanth B S[1]

[1]Department of Information Science & Engineering, Nitte Meenakshi Institute of Technology,
Yelahanka - 560064, Bengaluru

1 Introduction to SQLite Database

SQLite Databases are regular Relational DBMS designed for lower or mid range devices. SQLite databases are used when one wants a database to leave very low memory imprint and with fewer basic operations only. Some of the features of SQLite databases are listed below,

1. SQLite database along with the packages sums up to the size of 600Kb which is suitable for mobile devices
2. SQLite databases are serverless unlike MySQL or Oracle databases
3. In SQLite datbases, the data is encrypted and stored on to regular files and the read-write operations are directly performed as if one accessing the files which makes the database run very agile.
4. SQLite is ACID complaint, does not support advanced concepts such as Triggers
5. SQLite databases are written in C and is part of android libraries, there is no need to install the package explicitly

1.1 SQLiteOpenHelper Class

The APIs related to SQLite database are managed by an Android component called SQLite Manager. It is the component which provides the functions, methods, & classes for accessing, manipulating and versioning of SQLite databases. The SQLite Manager provides a class to do all the operations on the database called as *SQLiteOpenHelper* Class. To use the features of the SQLiteOpenHelper class, the model class must extend and make the class as inheritor of the features from Super class SQLiteOpenHelper as shown below,

```
public class DBHelper extends SQLiteOpenHelper{}
```

Once the custom class DBHelper extends SQLiteOpenHelper, One must add a Constructor, and override functions such as shown below,

1. DBHelper constructor takes context, Database name, cursor factory, Database version as arguments, and override the functions. The constructor should override the constructor of the super class taking same params as the constructor as arguments as shown below,

```
public DBHelper(@Nullable Context context,@Nullable String name,@Nullable
    SQLiteDatabase.CursorFactory factory,int version) {
        super(context, dbName, null, dbVersion);
    }
```

The context specifies the current class context, The second argument is a Database name which should be defined earlier as String and the CursorFactory specifies whether we are using default cursor object provided by SQLiteOpenHelper class or using custom cursor. By default, this object is set to NULL to tell that we are using default cursor object. The final argument is used for the database version.

2. The second function to be override by extending the SQLiteOpenHelper class is called as onCreate(SQLiteDatabase db) that takes SQLiteDatabase object pointing to the current database. It is called only once when the DB is created for the first time. The override function is as shown below,

```
@Override
public void onCreate(SQLiteDatabase db) {
   // Logic to create Database
}
```

3. Another default override function is onUpgrade(SQLiteDatabase db, int prev_version, int new version), which is used to upgrade database to another version or any preliminary sanitisation function like dropping tables if already exists, etc.

```
@Override
public void onUpgrade(SQLiteDatabase db, int prev_version, int new version) {
   // Logic to upgrade Database & perform preliminary setups
}
```

4. Similar to onUpgrade, the database can be downgraded using onDowngrade function which recieves the same arguments as onUpgrade and syntax is similar as well. It is shown below.

```
@Override
public void onDowngrade(SQLiteDatabase db, int old_version, int new version) {
   // Logic to downgrade Database
}
```

5. Another function that is provided by SQLiteOpenHelper is close(). which is used to close the database upon the completion of operations on database.

1.2 SQLiteDatabase Class

The Android System also provides SQLiteDatabase class that facilitates the process of table creation, and basic CRUD operation on the database. The following are some of the member functions of SQLiteDatabase class that are often used,

1. execSQL function is used to execute raw SQL statements which is called by SQLiteDatabase object

```
SQLiteDatabase db ; //instance of SQliteDatabase object
db.execSQL(String SQL_statement) ; // SQL statement
```

2. getReadableDatabase() and getWritableDatabase() functions are used to open the Database file in Read and Write/ mode respectively. Both are demonstrated here

```
SQLiteDatabase db = = context.getWritableDatabase(); // DB is opened in Write mode
SQLiteDatabase db = = context.getReadableeDatabase(); // DB is opened in Read mode
```

3. To insert data into database, data is actually inserted in the form of (Key,value) pair which is prepared by the ContentValue object. The default values can also be made null with the help of nullColumnHack argument. The API is as shown below,

```
long insert(String tablename, String nullColumnHack, ContentValues values);
```

4. To delete, we use delete API, which takes three arguments, they are tablename, where clause and arguments for the where arguments. It is as shown below,

```
int delete(String table, String whereClause, String[] whereArgs);
```

5. To update, we use update API, which takes three arguments, they are tablename, where clause and arguments for the where arguments. It is as shown below,

```
int update(String table, ContentValues values, String whereClause, String[]
    whereArgs);
```

6. to execute a query we use query function which returns a cursor object. A cursor object points to the first row output of the SQL statement that got executed and its iteratable. The query functions uses arguments such as tablename, column name, selection and its arguments, groupby, having and orderby arguments respectively depending upon the user choices. Not all the arguments are usable at the same time and they can be set to null if its not being used.

```
Cursor cur = query(String table, String[] columns, String selection, String[]
    selectionArgs, String groupBy, String having, String orderBy, String limit);
```

7. rawQuery function is also used to execute the SQL query that returns a cursor object to the ResultSet.

```
Cursor cur = rawQuery(String sql, String[] selectionArgs);
```

2 Exercise

Create a Login and Registration App using SQLite database which provides basic CRUD operations such as,

- Insert a new user
- Delete an existing user upon inputting the name
- Update the existing user
- Display all the users in the form of a String

The following are the steps followed to create the Application.

1. Create 4 different Activities for the operations Register(insert), Delete, Update and Display operation respectively. This is done by right clicking on project and selecting new->Activity->Empty Activity. The following files are created upon creating 4 activities.
    - activity_registration.xml & Registration.java for Registration page
    - activity_display_page.xml & DisplayPage.java for Display Activity

- activity_updatepage.xml & UpdatePage.java for Update Activity
- activity_deletepage.xml & DeletePage.java for Delete Activity

2. Prepare a Separate class encapsulating all the functions on the database. Call this class as DBHelper Class. Do the following steps,

   (a) Make the DBHelper Class extend the SQLiteOpenHelper Class

   (b) Define the variables such as tablename, database name, version.

   (c) Add the constructor matching the SQLiteOpenHelper Class and add the user defined database name and version

   (d) Override the methods of SQLiteOpenHelper such as onCreate() and onUpgrade() which are mandatory

   After completing all the above steps, the code looks as follows,

```
public class DBHelper extends SQLiteOpenHelper{
private static final String dbName="studentdb"; // Database Name
private static final String tbName="student" ; // Table Name
private static final int dbVersion = 1 ; // Version - 1 indicating first version

// Constructor matching the SQLiteOpenHelper class
public DBHelper(@Nullable Context context,@Nullable String name,@Nullable
    SQLiteDatabase.CursorFactory factory,int version) {
        super(context, dbName, null, dbVersion);
    }
//  onCreate()
@Override
public void onCreate(SQLiteDatabase db) {
   // Logic to create Database
}
// onUpgrade()
@Override
public void onUpgrade(SQLiteDatabase db, int prev_version, int new version) {
   // Logic to upgrade Database & perform preliminary setups
}
}
```

3. Create the table in the onCreate function using execSQL function as follows,

```
@Override
public void onCreate(SQLiteDatabase db) {
       db.execSQL("CREATE TABLE "+tbName+"(uname VARCHAR(10),passw
           VARCHAR(10))"+";");
       //SQL - CREATE TABLE tbName(uname VARCHAR(10),passw VARCHAR(10));
}
```

4. Implement on onUpgrade function. If the table already exists, then we will drop the table and then we recreate the table again by calling onCreate function as follows,

```
 @Override
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
       db.execSQL("DROP TABLE IF EXISTS "+tbName);
       onCreate(db);
}
```

5. Implement insert function as addUser function that takes name and usn as arguments. We perform the following steps

   (a) Create a SQLiteDatabase and open it in Writeable mode using getWritableDatabase()

   (b) Prepare the Data for insert as (key,value) pair. The key would be the column name and value will be the value for the name. This is done by instantiating the ContentValues.

   (c) Call the SQLiteDatabase.insert function passing tablename, nullcolumnHack and content values. The return value is in the long type, store it to a long variable

   (d) close the database and return the long value. The steps are shown below,

```
public long adduser(String name, String pass){
        SQLiteDatabase sqLiteDatabase = this.getWritableDatabase(); //5(a)
        ContentValues cv = new ContentValues() ;// 5(b)
        cv.put("uname", name);
        cv.put("passw ",pass) ;
        long result = sqLiteDatabase.insert(tbName, null,cv); // 5(c)
        sqLiteDatabase.close(); // 5(d)
        return result ;
}
```

6. Implement the update function on the similar lines, the steps are,

   (a) Create a SQLiteDatabase and open it in Writeable mode using getWritableDatabase()

   (b) Use the execSQL to which pass the update SQL Statement

   (c) Close the database

```
public void update(String name, String pass){
        SQLiteDatabase sqLiteDatabase = this.getWritableDatabase(); //6(a)
        sqLiteDatabase.execSQL("UPDATE "+tbName+" SET passw='"+pass+"'"+" WHERE
            uname='"+name+"';");//6(b)
        // UPDATE tbName SET passw='pass' WHERE uname='name' ; SQL statement
        sqLiteDatabase.close(); // 6(c)
    }
```

7. Implement the delete function that takes the name as the input argument and follow these steps,

   (a) Create a SQLiteDatabase and open it in Writeable mode using getWritableDatabase()

   (b) Use the execSQL to which pass the DELETE SQL Statement using name as matching argument

   (c) Close the database

```
public void delete(String name){
        SQLiteDatabase sqLiteDatabase = this.getWritableDatabase(); // 7(a)
        sqLiteDatabase.execSQL("DELETE FROM "+tbName+" WHERE uname='" +name+"';" );
            // 7(b)
        // SQL statement : DELETE FROM tbName WHERE uname='name';
        sqLiteDatabase.close(); // 7(c)
    }
```

8. Implement the Display function by following the steps below,

(a) Create a SQLiteDatabase and open it in Readable mode using getReadableDatabase()

(b) Execute the SELECT statement using SQLiteDatabase.rawQuery() function and store the resultset into Cursor Object

(c) iterate through the Cursor Object using Cursor.moveToNext() to parse the resultset

(d) use CursorObject.getString(int Columnnumber) to retrieve the relevant fields from the table for all rows. Columns can be referred by using index that starts from 0.

The steps are shown below,

```
public String display(Context ctx){
        SQLiteDatabase sqLiteDatabase = this.getReadableDatabase(); //8(a)
        Cursor cursor = sqLiteDatabase.rawQuery("SELECT * FROM "+tbName, null); //8(b)
        String finalres = " " ;
        while(cursor.moveToNext()){ //8(c)
            finalres += cursor.getString(0)+":"+cursor.getString(1); // 8(d)
        }
        return finalres;
}
```

9. Design the Landing Page by tweaking the activity_main.xml which is shown in figure 1

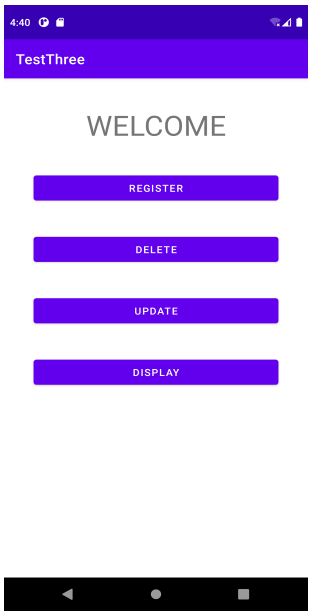
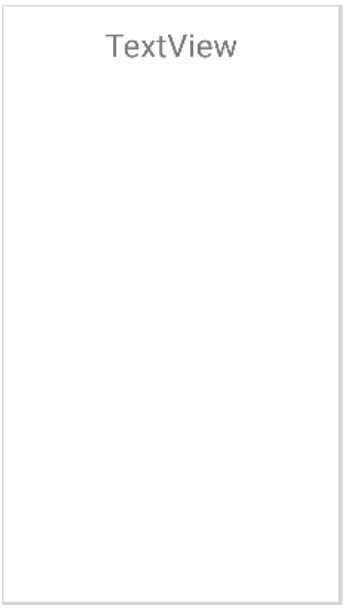


Figure 1: Landing Page

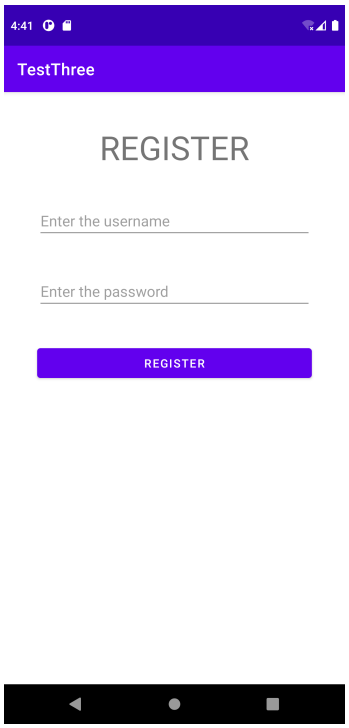
10. Design the pages for activity_display_page.xml, activity_registration.xml as shown in the figure 2a, 2b respectively.

11. Design the pages for activity_deletepage.xml & activity_updatepage.xml as shown in the figure 3a & 3b respectively.

12. Modify the MainActivity.java and instantiate the respective buttons. Add onClickListerner to the buttons and navigate from MainActivity to the respective action activities(DeletePage, Registration, UpdatePage, DisplayPage) using Anonymous onClick definition respectively. Sample Code for navigation to Registration Page is shown below,

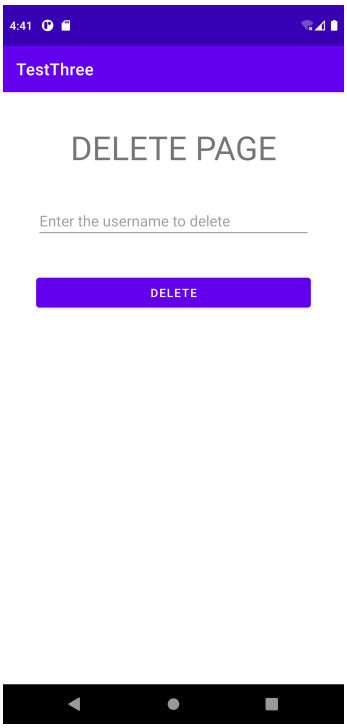


(a) Display Page

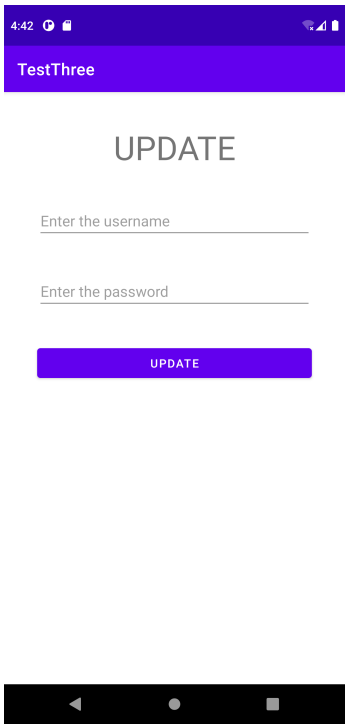


(b) Registration Page

Figure 2: Display & Registration Page Design



(a) Delete Page



(b) Update Page

Figure 3: Delete & Update Page Design

```
Button register, update, delete, display;
 @Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    register = findViewById(R.id.register);
   // equate the other buttons for update, delete and display here
   // setting onClickListener Anonymously
   register.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
   // Using Intent to Navigate to Registration.java and starting it upon register
       button click
            Intent it = new Intent(MainActivity.this, Registration.class);
            startActivity(it);
    }
});
// Similarly define Anonymous OnClickListerner for update, delete and display
    buttons as well here
}
```

13. Modify the Registration.java page with the following steps,

    (a) Define the Equivalent JAVA objects in Registration.java. Copy the dbName and dbVersion parameters from DBHelper class. Define an object of DBHelper class.

    (b) Equate the JAVA objects with XML objects using findViewByID function

    (c) Add an anonymous Event Listener to register button and override onClick function.

    (d) Inside the onClick function create the DBHelper Object by calling the default constructor and passing the context, dbName, cursorfactory and dbversion as arguments

    (e) Pass on the name and password from EditText to the addUser function called by using DBHelper object which returns -1 on error.

    (f) if it returns other value than -1, then it indicates the operation is successful and use Intent to navigate back to the main activity.

    The code snippet for the above steps are shown below indicating the step number

```
EditText uname; // 13(a)
EditText upass ;
DBHelper dbHelper;
Button register;
private static final String dbName="studentdb";
private static final String tbName="student" ;
private static final int dbVersion = 1 ;

 @Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_registration);
   uname = findViewById(R.id.rname) ; //13(b)
   upass = findViewById(R.id.rpass) ;

   register = findViewById(R.id.r_register) ;
```

```
register.setOnClickListener(new View.OnClickListener() { // 13(c)
    @Override
    public void onClick(View v) {
        dbHelper = new DBHelper(Registration.this, dbName, null, dbVersion) ;
            //13(d)
        long val = dbHelper.adduser(uname.getText().toString(),
            upass.getText().toString()) ; // 13(e)
        if(val == -1)
             Toast.makeText(Registration.this, "Error in adding user",
                 Toast.LENGTH_SHORT).show();
        else{
             Toast.makeText(Registration.this, "USER REGISTERED",
                 Toast.LENGTH_SHORT).show(); //13(f)
             Intent it = new Intent(Registration.this, MainActivity.class);
             startActivity(it);
             }
         }
     });
 }
```

14. On the similar grounds, Implement the UpdatePage.java,The code inside the onClick function is as shown below,

```
update.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
               dbHelper = new DBHelper(Updatepage.this, dbName, null, dbVersion) ;
               dbHelper.update(uname.getText().toString(),
                   upass.getText().toString());
               Intent it = new Intent(Updatepage.this, MainActivity.class);
               startActivity(it);
           }
});
```

15. The delete function defined in DBHelper class takes in name as input and we have to pass that as an argument for the delete function.Modify tehe DeletePage.java. Follow the template for Registration page and the onClick function code for the Delete button is as shown below,

```
delete.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
               dbHelper = new DBHelper(DeletePage.this, dbName,null, dbVersion) ;
               dbHelper.delete(duname.getText().toString());
               Intent it = new Intent(DeletePage.this, MainActivity.class);
               startActivity(it);
           }
});
```

16. Finally call the DBHelper.display() to retreive the username and update that in the TextView by modifying DisplayPage.java class. The code snippet demonstrate the same.

```
TextView text ;
DBHelper dbHelper;
```

(a) After Inserting Two Records

(b) After Deleting one Record

Figure 4: Results

```
private static final String dbName="studentdb";
private static final int dbVersion = 1 ;
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_display_page);
        text = findViewById(R.id.textView) ;
        dbHelper = new DBHelper(DisplayPage.this, dbName, null, dbVersion);
        String out= dbHelper.display(DisplayPage.this);
        text.setText(out);
    }
```

3 Results

The Result obtained after inserting couple of users(Sanket & Rajesh) into the SQLiteDatabase is depicted in output snapshot shown in figure 4a, the second snapshot shows after deleting the first user Sanket how database looks like in figure 4b.