LESSON – 9

SQLITE DATABASE

Agenda

- Introduction to SQLite Database
- SQLiteOpenHelper
- Content values
- Cursor
- CRUD Queries
- Hands on Example-1
- Hands on Example-2
- SQLite Browser

INTRODUCTION

- SQLite is a lightweight, embedded relational database management system that is included as part of the Android framework and provides a mechanism for implementing organized persistent data storage for Android applications.
- In addition to the SQLite database, the Android framework also includes a range of Java classes that may be used to create and manage SQLite based databases and tables.

Introduction

- SQLite is an open source storage database that manages all database related queries.
- Maintain structures data.
  - Server less and self contained.
  - Full documentation is available at http://www.sqlite.org
- Android provides its own API to deal with database connectivity. This API is provided in android.database and android.database.sqlite packages.
- In apps, we interact with SQLite database using the SQLiteOpenHelper class and the SQLiteDatabase class.
- For the full syntax of all SQLite comments see https://sqlite.org/lang.html

Steps to follow

1. Create data model (Classes like Employee, Customer)
2. Subclass SQLiteOpenHelper
   a. Create constants for tables(Database Version, Database name, table name, columns names etc.,)
   b. onCreate()—create SQLiteDatabase with tables
   c. onUpgrade(), and optional methods(eg: onDownGrade(),onOpen(), etc.,)
   d. Implement query(), insert(), delete(), update(), count()
3. In MainActivity, create instance of SQLiteOpenHelper
4. Call methods of SQLiteOpenHelper to work with database

Example : SQLiteCRUDDemo Folder

SQLiteOpenHelper

- Using SQLiteOpenHelper
  - The main functionality of the class is to open the database if it exists, create if it does not, and upgrade the version as required.
  - It provides a constructor to construct a helper class by subclassing this class and overriding the methods named onCreate() an onUpgrade().

Syntax:

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

where,

- context – represent the context to create or open the database
- name – represents the name of the database
- factory – represents the factory class used for creating the cursor object ,its an optional parameter passed as null.
- version – represent the number of the database.(3 – latest version)

SQLiteOpenHelper Example

Step 1 :

Create your DataBaseHelper class which extends from SQLiteOpenHelper with the necessary parameters.

```
class DatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, 1){

}
```

- Some of the commonly used methods of this class are as follows:
- onCreate (SQLiteDatabase db)
  - The method is invoked when the database is created for the first time. This is where the table is created and populated. The database name is passed as an argument.
- onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)
  - The method is invoked when the database needs to be upgraded. The implementation should use this method to drop tables, add tables, or perform any other operations such as the need to upgrade to the new schema version.

Note : Refer SQLite version from

https://developer.android.com/reference/android/database/sqlite/package-summary.html

Declare fields as instance or static

Step 2 : Declare the necessary fields as static to get direct access without instance using companion object in Kotlin. Kotlin does not support static keyword like Java, instead you can use companion object. Inside companion object fields and methods are works like static.

```
companion object {
        val DATABASE_NAME = "emp.db"
        val TABLE_NAME = "emp_table"
        val COL_1 = "ID"
        val COL_2 = "NAME"
        val COL_3 = "DESIG"
        val COL_4 = "DEPT"
    }
```

Employee Table

Column	Data Type
Id	Number
Name	Varchar
Desig	Varchar
Dept	Varchar

Step 3 : Override onCreate() and onUpgrade() inside DatabaseHelper

```
class DatabaseHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, 1) {
    /**
     * Our onCreate() method.
     * Called when the database is created for the first time. This is  where the creation of tables and the
      *  initial population of the tables should happen.     */
    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL("CREATE TABLE $TABLE_NAME (ID INTEGER PRIMARY KEY " +
                "AUTOINCREMENT,NAME TEXT,DESIG TEXT,DEPT TEXT)")
    }
    /**
     * Let's create Our onUpgrade method
     * Called when the database needs to be upgraded. The implementation should
     * use this method to drop tables, add tables, or do anything else it needs
     * to upgrade to the new schema version.
     */
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME)
        onCreate(db)
    }
```

Notable Methods of the SQLiteOpenHelper Class

- The database is not actually created or opened until one of getWritableDatabase() or getReadableDatabase() is called.
- getWritableDatabase() – Opens or creates a database for reading and writing. Returns a reference to the database in the form of a SQLiteDatabase object.
- getReadableDatabase() – Creates or opens a database for reading only. Returns a reference to the database in the form of a SQLiteDatabase object.
- close() – Closes the database.

Content values

- ContentValues is a convenience class that allows key/value pairs to be declared consisting of table column identifiers and the values to be stored in each column.
- This class is of particular use when inserting or updating entries in a database table.
- Example

```
var values=ContentValues();
values.put("id",Integer.parseInt(et1.getText().toString()));
values.put("name",et2.getText().toString());
values.put("desig",et3.getText().toString());
values.put("dept", et4.getText().toString());
```

Cursor Class

- Cursor : A class provided specifically to provide access to the results of a database query. Key methods of this class are as follows:
  - close() – Releases all resources used by the cursor and closes it.
  - getCount() – Returns the number of rows contained within the result set.
  - moveToFirst() – Moves to the first row within the result set.
  - moveToLast() – Moves to the last row in the result set.
    moveToNext() – Moves to the next row in the result set.
    moveToPosition(int position): Moves the cursor to the specified position
  - get<type>() – Returns the value of the specified <type> contained at the specified column index of the row at the current cursor position (variations consist of getString(), getInt(), getShort(), getFloat() and getDouble()).

CRUD Queries

- SQLiteDatabase : The class has methods to create, delete, execute SQL commands, and perform other common database management tasks. Some of the commonly used methods of this class are as follows:

Method	Description
execSQL(String sql): Unit	Executes the SQL query, not a select query. Example: db is an instance of SQLiteDatabase db.execSQL("CREATE TABLE $TABLE_NAME (ID INTEGER PRIMARY KEY " + "AUTOINCREMENT,NAME TEXT,DESIG TEXT,DEPT TEXT)") db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME)

CRUD Queries

Method	Description
insert(String table, String nullColumnHack, ContentValues values): Long	**Inserts a record on the database. The table specifies the table name, nullColumnHack doesn't allow completely null values for the row. Useful to add nullalble values by specifying column names. The third argument specifies the values to be stored.** **Example:** **val db = this.*writableDatabase*** **val contentValues = ContentValues()** **contentValues.put(COL_2, name)** **contentValues.put(COL_3, surname)** **contentValues.put(COL_4, marks)** **db.insert(TABLE_NAME, null, contentValues)**

CRUD Queries

Method	Description
update(String table, ContentValues values, String whereClause, String[] whereArgs): Int	Updates a row. Example: **val** db = **this**.*writableDatabase* **val** contentValues = ContentValues() contentValues.put(**COL_1**, id) contentValues.put(**COL_2**, name) contentValues.put(**COL_3**, des) contentValues.put(**COL_4**, dpt) db.update(**TABLE_NAME**, contentValues, **"ID = ?"**, *arrayOf*(id))
Delete(String table, String whereClause, String[] whereArgs): Int	Delete a row val db = this.writableDatabase db.delete(TABLE_NAME,"ID = ?", arrayOf(id))

CRUD Queries

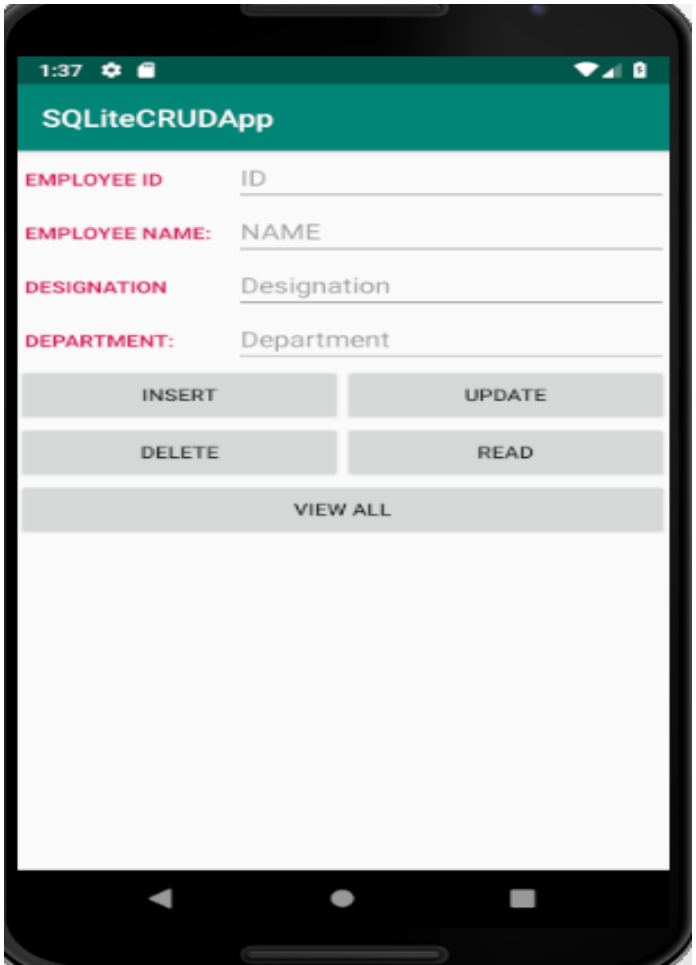
Method	Description
rawQuery() **rawQuery(String sql, String[] selectionArgs) : Cursor**	The rawQuery method is same as execSQL, used to execute an SQL query on the database, but a result is expected from the rawQuery method. The returned result is a Cursor object, which is a wrapper object of the returned rows. Here res is Cursor object Example: val db = this.writableDatabase val res = db.rawQuery("SELECT * FROM " + TABLE_NAME, null) val q = "SELECT * FROM " + TABLE_NAME + " WHERE id =" + eid val k = db.rawQuery(q, null)

CRUD Queries

Method	Description
query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy): Cursor	Returns a cursor over the resultset. SQLiteDatabase db = helper.getReadableDatabase(); String table = "table2"; String[] columns = {"column1", "column3"}; String selection = "column3 =?"; String[] selectionArgs = {"apple"}; String groupBy = null; String having = null; String orderBy = "column3 DESC"; String limit = "10"; val cursor = db.query(table, columns, selection, selectionArgs, groupBy, having, orderBy, limit);

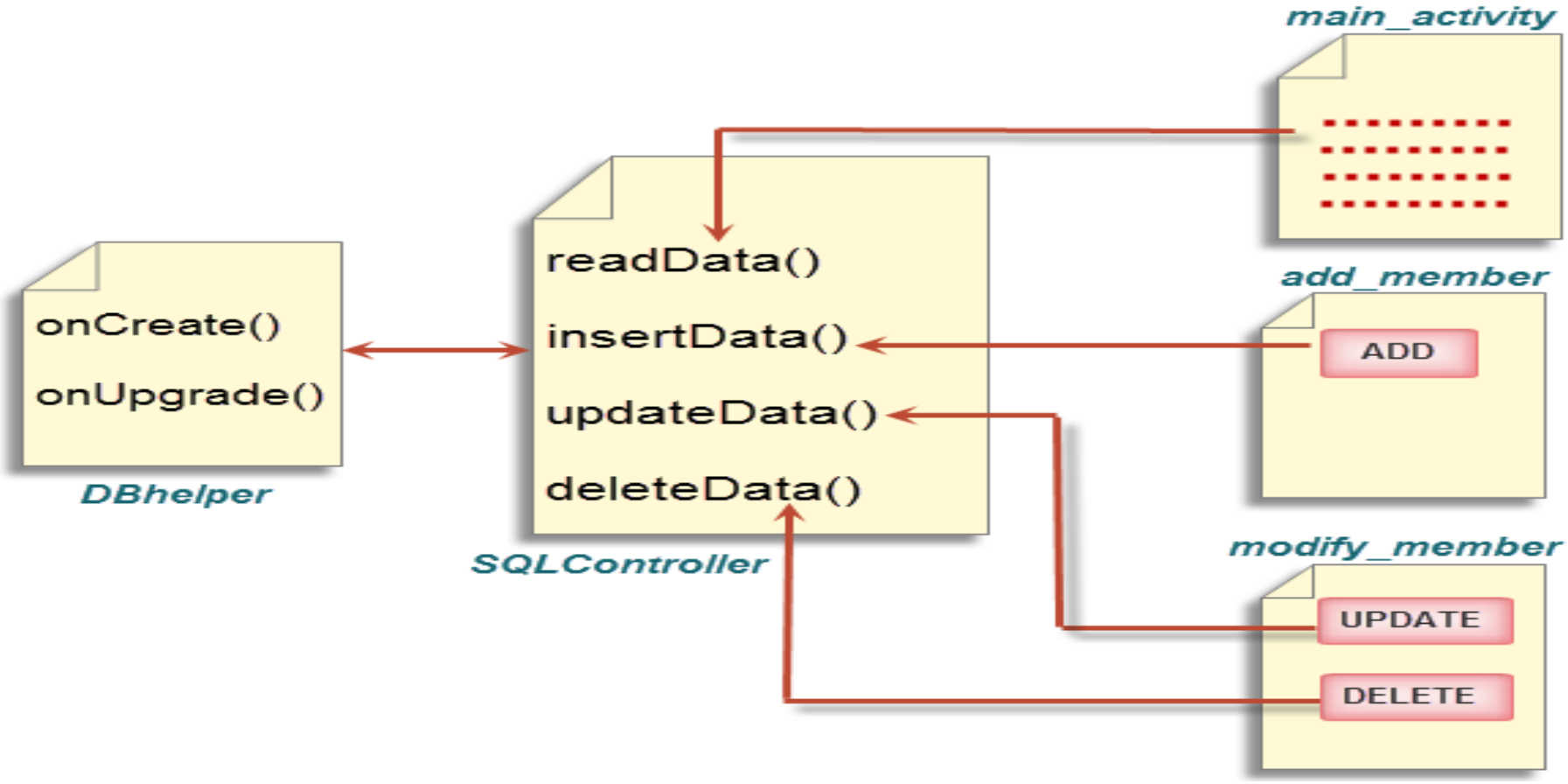
Hands on example - 1

- Work with simple Employee database with single table to perform CRUD operation using SQLiteOpenHelper and SQLiteDatabase. Eg : SQLiteCRUDDemo



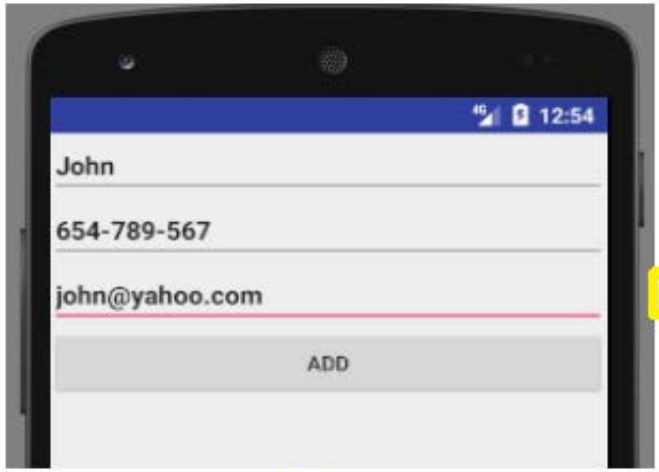
Hands on Example - 2

Here a database called contact with a table called test to hold contactid, name, mobile and email and perform the following operation. Refer : Lesson9/CustomerSQLite

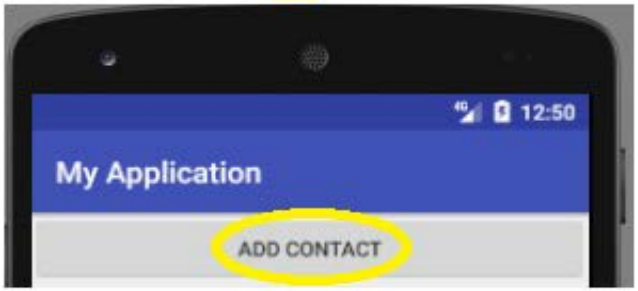


Adding Contacts

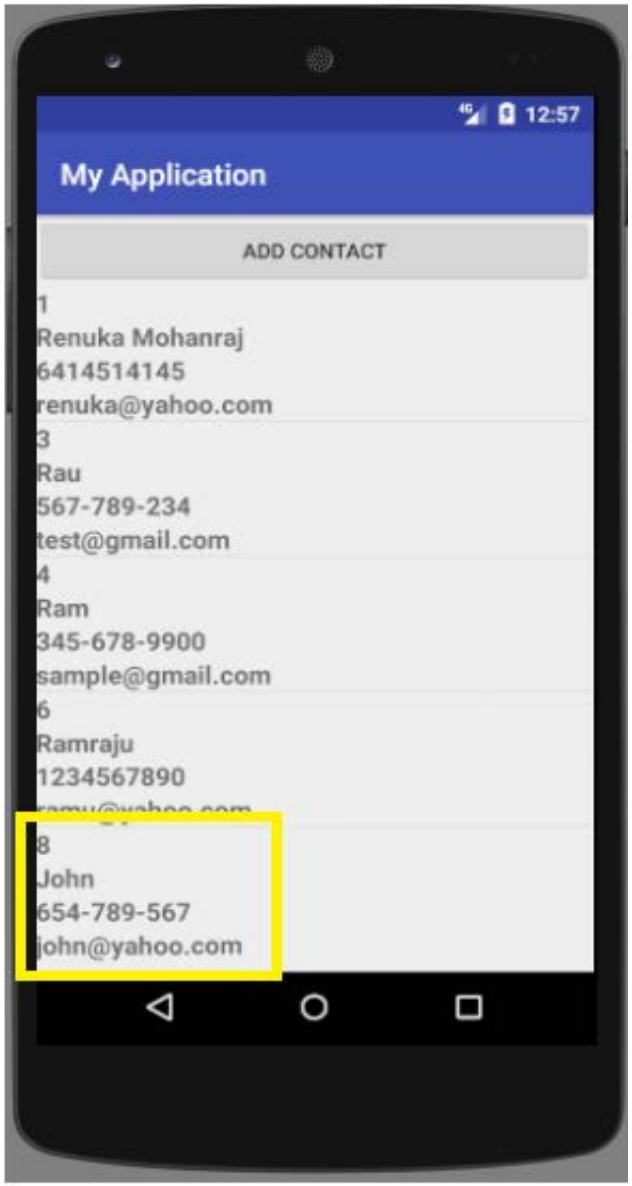
Add_member.java



MainActivity.java

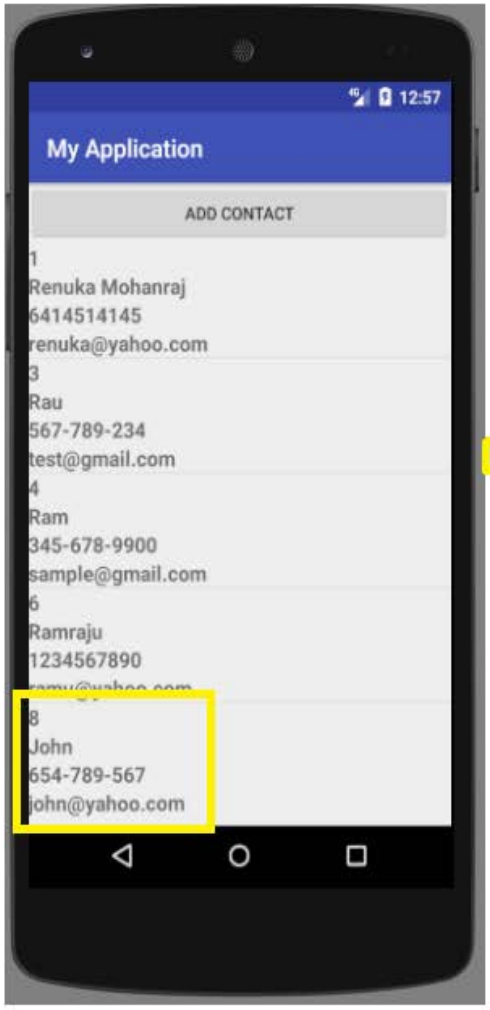


MainActivity.java

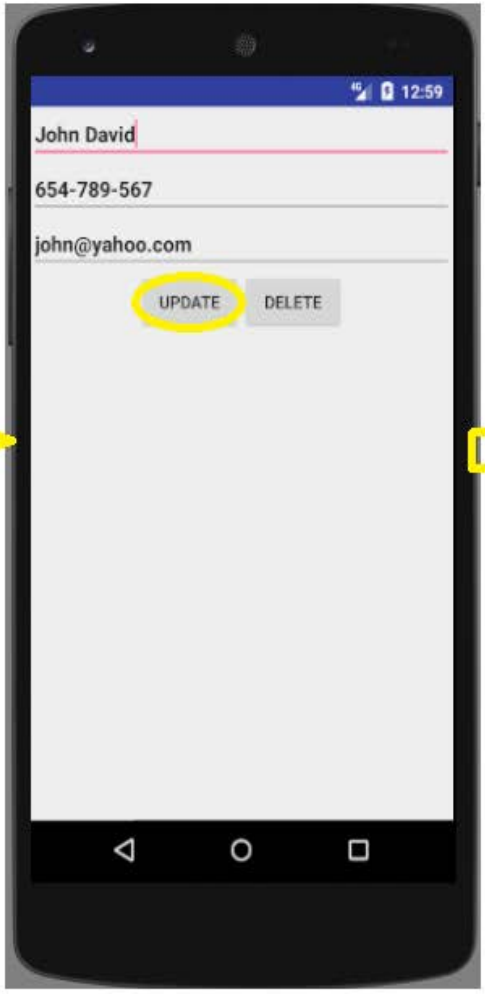


Updating Contacts

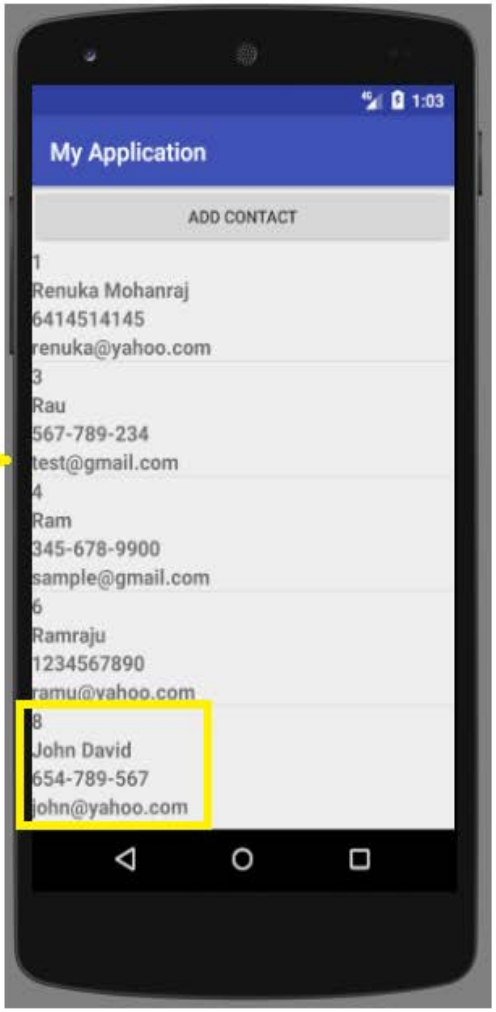
Once you click on entry from the Listview, then you will get Modify_member activity. You can go for Update, changes will be reflect in the MainActivity ListView.



MainActivity.java



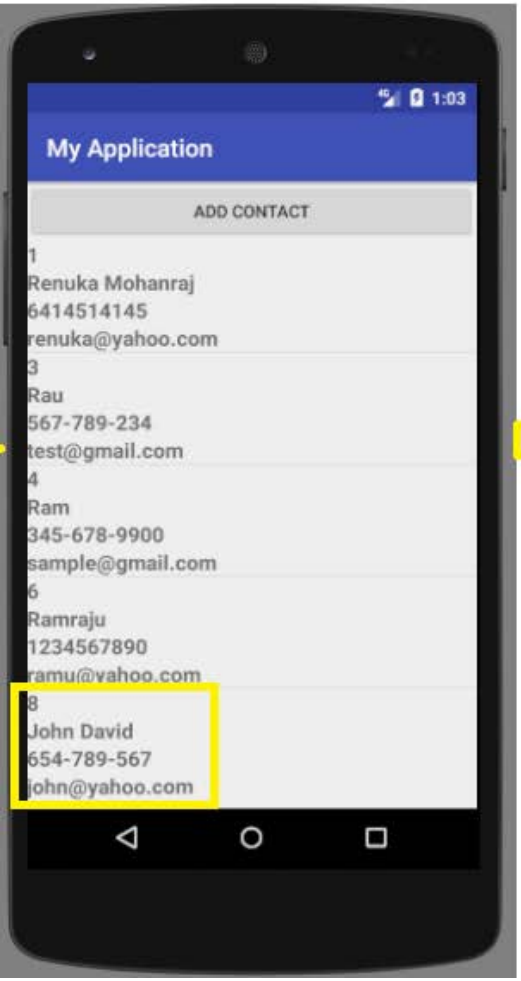
Modify_member.java



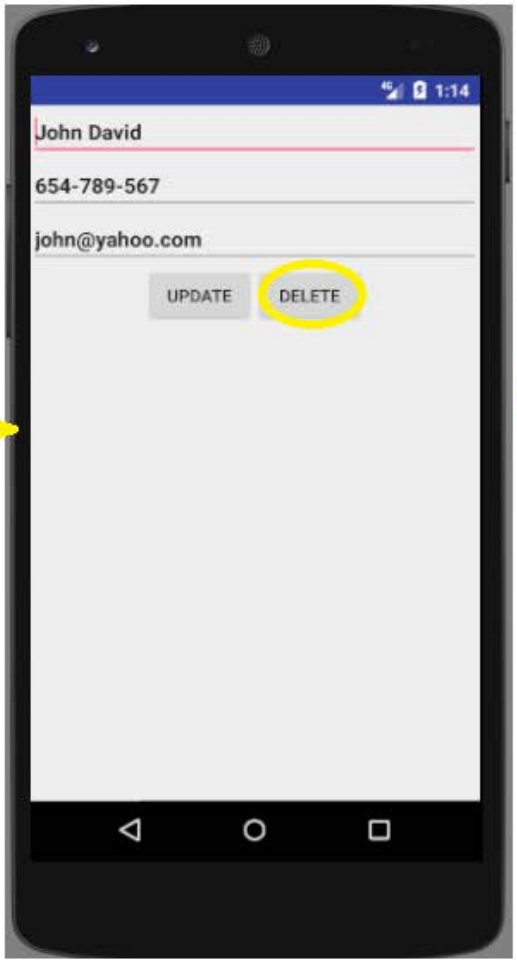
MainActivity.java

Deleting Contacts

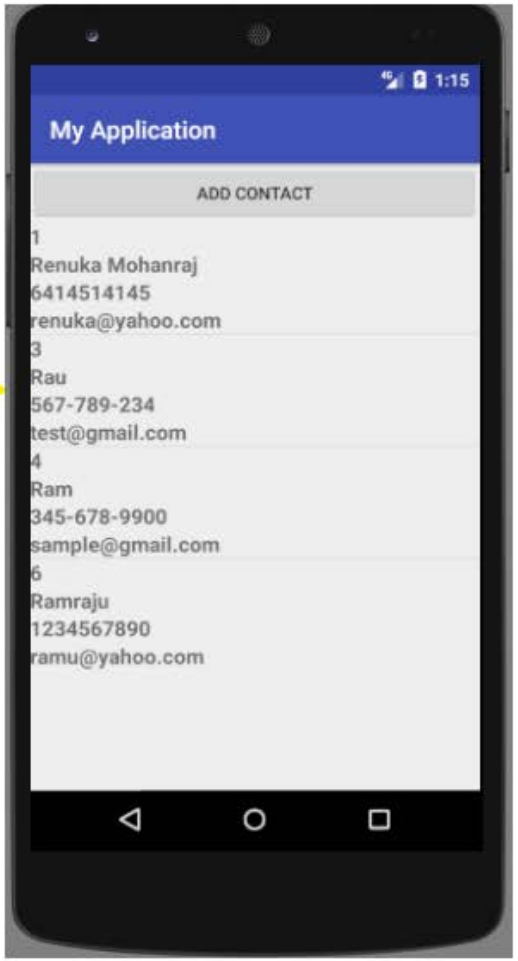
Once you click on entry from the Listview, then you will get Modify_member activity. You can go for Delete, changes will be reflect in the MainActivity ListView.



MainActivity.java



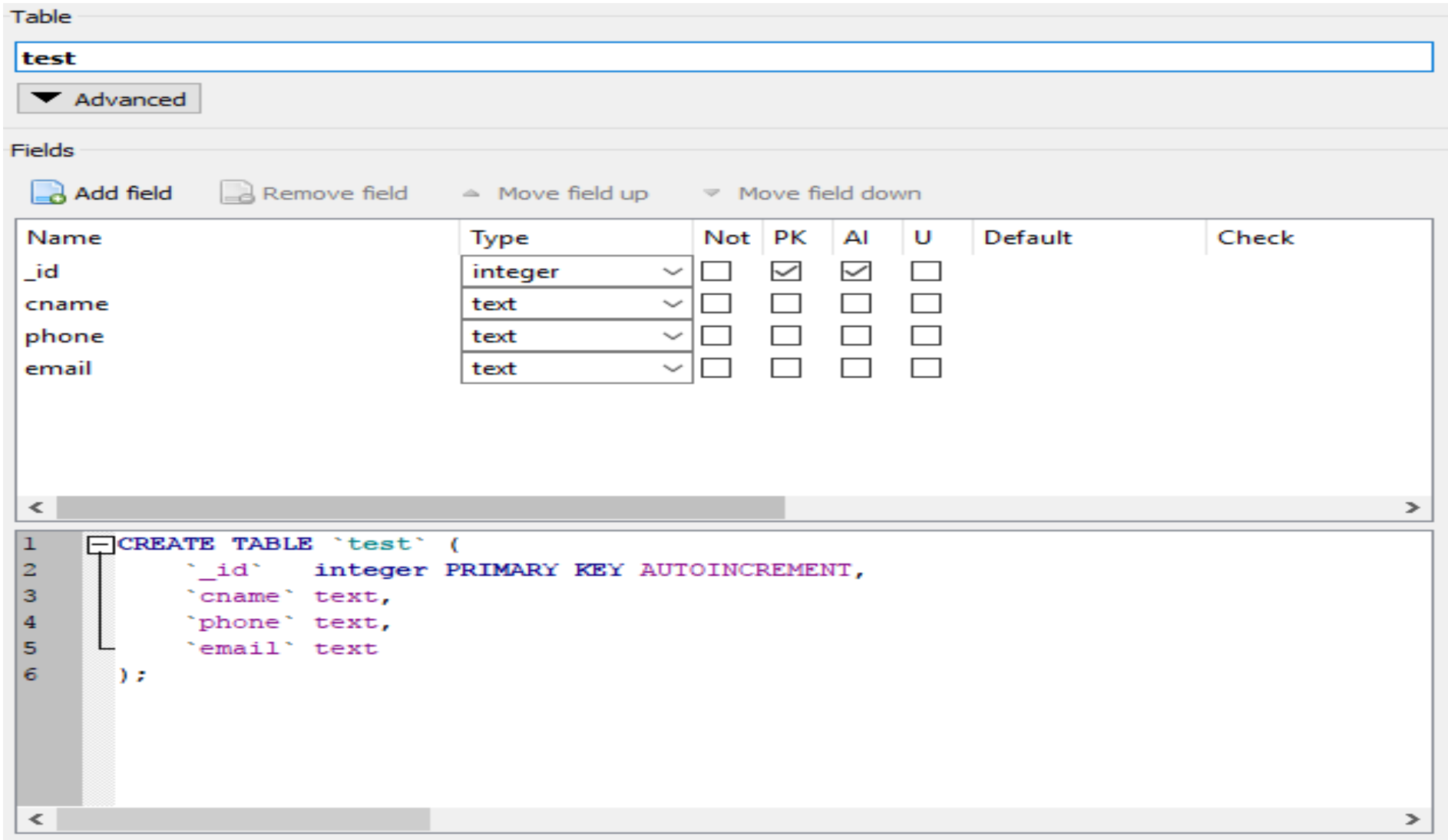
Modify_member.java



MainActivity.java

Database and table structure from SQLite Browser

Database name : Contacts, Table name : test, below is the table structure



SQL Operation	Activity Layout File	Java File
Select	activity_main.xml view_member_entry.xml	MainActivity.kt ListView Component Layout
Insert	add_member.xml	Add_member.kt
Update and Delete	modify_member.xml	Modify_member.kt
Database and Table Creation		DBHelper.kt
CRUD operations methods		SQLController.kt

Simple Cursor Adapter

- An easy adapter to map columns from a cursor to TextViews or ImageViews defined in an XML file. You can specify which columns you want, which views you want to display the columns, and the XML file that defines the appearance of these views.
- Example:

```
        lv = (ListView) findViewById(R.id.lv1);
        val cursor = dbcon.readData();
// Columns
var from =
arrayOf<String>{DBHelper.KEY_ROWID,DBHelper.NAME,DBHelper.PHONE,DBHelper.EMAIL};

// Matching ids from view for the specified columns
var to = intArrayOf { R.id.cid,R.id.vname, R.id.vmobile,R.id.vemail };

val adapter = SimpleCursorAdapter(
        this@MainActivity, R.layout.view_member_entry, cursor, from, to,1)
  /*Notifies the attached observers that the underlying data has been
        changed and any View reflecting the data set should refresh itself.  */
    adapter.notifyDataSetChanged();
    lv.setAdapter(adapter);
```

How to view the table using SQLite browser

Step 1 : Install SQLite browser according to your OS using https://sqlitebrowser.org/dl/.

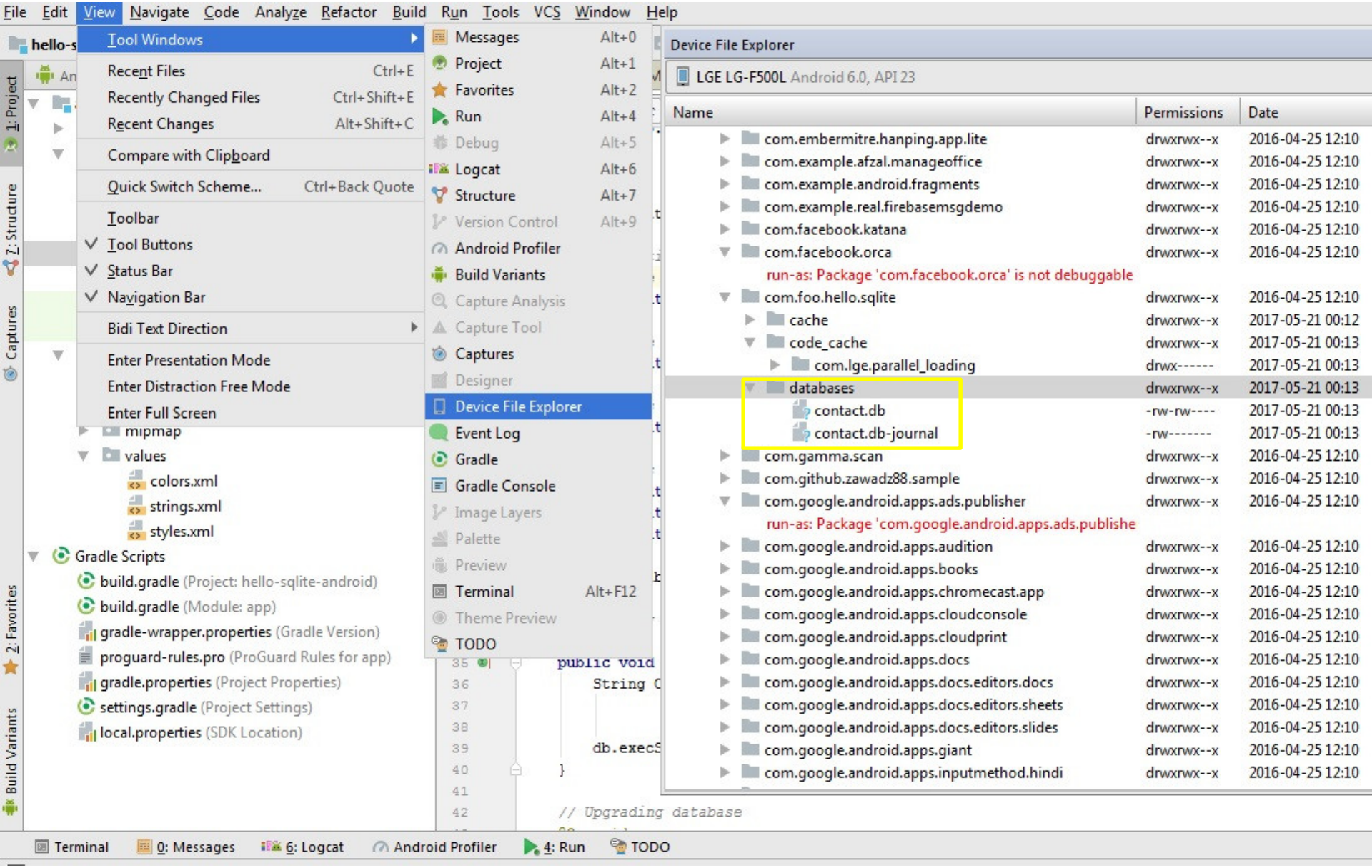
Step 2 : View > Tool Windows > Device File Explorer

Step 3: Expand /data/data/[package-name]\databases

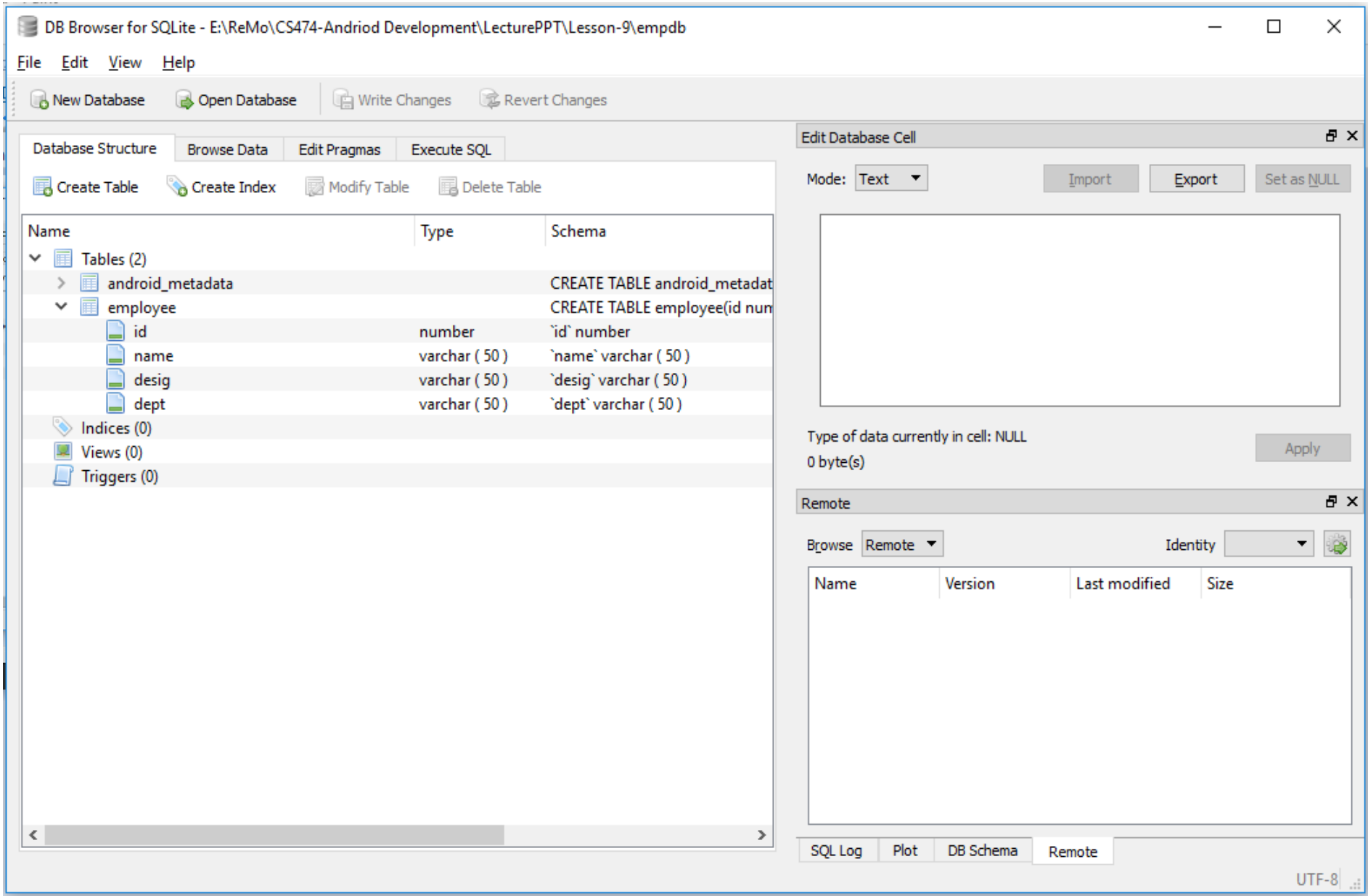
Step 4: Right click on your database and save as on your pc.

Refer the next slide screen shot for visual idea.

Device Explorer



Step 5: Open your saved database file using SQLite Browser.



Firebase

- Firebase lets you build more powerful, secure and scalable apps, using world-class infrastructure.
- Store and sync data with NoSQL cloud database. Data is synced across all clients in real-time and remains available when your app goes offline.
- Data is stored as JSON and synchronized in real-time to every connected client.
- When you build cross-platform apps with iOS, Android, and JavaScript SDKs, all your clients share one Realtime Database instance and automatically receive updates with the newest data.

Firebase Applications

- Cloud Firestore
  - Store and sync data between users and devices
- ML Kit
  - Bring powerful machine learning features to your mobile app
- Cloud Functions
  - Create functions that are triggered by Firebase products, such as changes to data in the Realtime Database, new user sign-ups via Auth etc.,
- Authentication
  - Manage your users in a simple and secure way. Firebase Auth offers multiple methods to authenticate, including email and password, third-party providers like Google or Facebook, and using your existing account system directly.

Firebase Applications

- Hosting
  - Simplify your web hosting with tools made specifically for modern web apps.
- Cloud Storage
  - Store and share user-generated content like images, audio, and video with powerful, simple, and cost-effective object storage built for Google scale.
- Real time database
  - Realtime Database is Firebase's original database. It's an efficient, low-latency solution for mobile apps that require synced states across clients in realtime.