CRUD operation using SQLite database connection

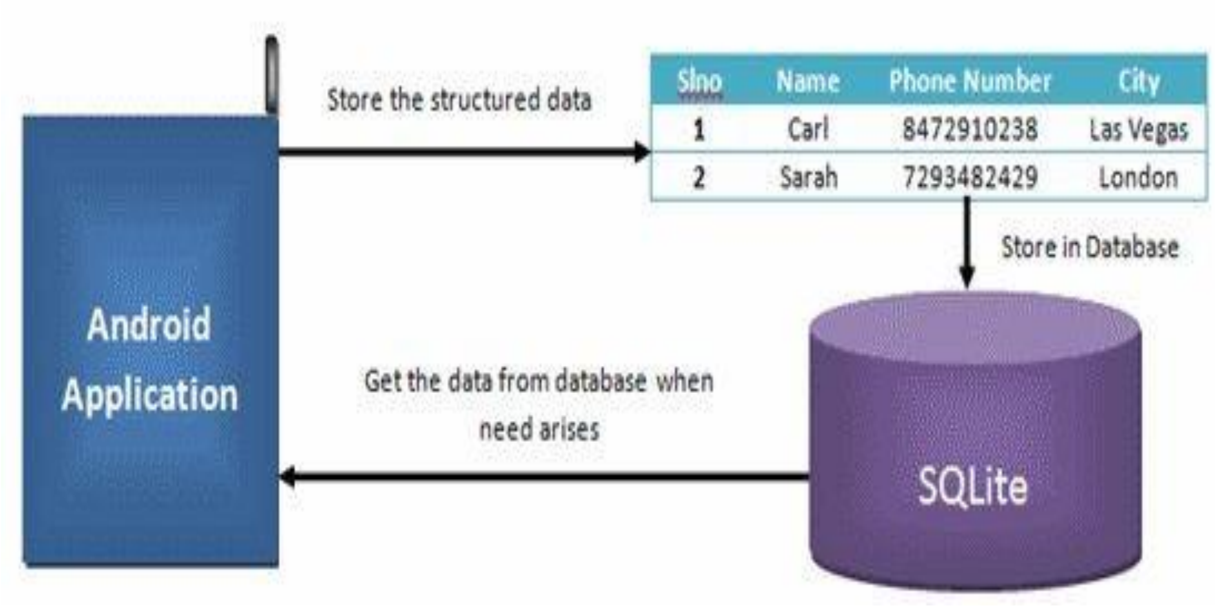
What is SQLite Database ?

- A database is very useful for any large or small system, unless your system deals only with simple data, without using a bank to store information.
- SQLite Database is an open-source database provided in Android which is used to store data inside the user's device in the form of a Text file.
- We can perform so many operations on this data such as adding new data, updating, reading, and deleting this data.
- SQLite is an offline database that is locally stored in the user's device and we do not have to create any connection to connect to this database.

SQLite database

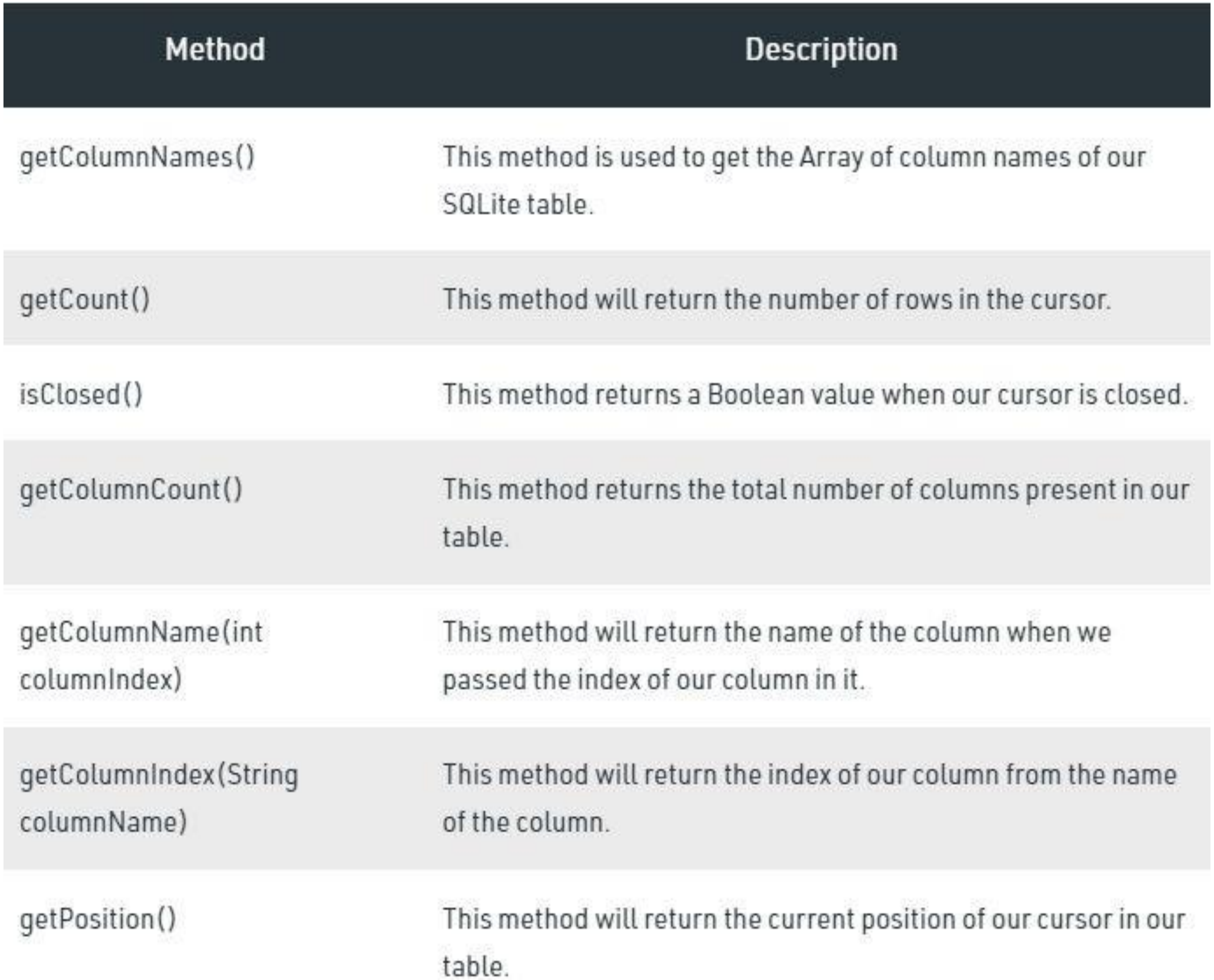
- The Android uses the SQLite database that is open-source and widely used in popular applications.
- Other examples of who uses the SQLite are Mozilla Firefox and iPhone.
- Database that you create in an Android application is only available for the same application, unless you use a content provider.
- Once you have created the database, it is stored in the directory **"/ data / data / {package name} / databases / database {name}"**.
- The recommended method to create a new SQLite database is to create a subclass of SQLiteOpenHelper and override the onCreate () method, in which you can run a SQLite command to create tables in the database.

How Data is Being Stored in the SQLite Database?


Android Application
Store the structured data
Store in Database
Get the data from database when need arises
SQLite

Slno	Name	Phone Number	City
1	Carl	8472910238	Las Vegas
2	Sarah	7293482429	London

Methods in SQLite Database



Method	Description
getColumnNames()	This method is used to get the Array of column names of our SQLite table.
getCount()	This method will return the number of rows in the cursor.
isClosed()	This method returns a Boolean value when our cursor is closed.
getColumnCount()	This method returns the total number of columns present in our table.
getColumnName(int columnIndex)	This method will return the name of the column when we passed the index of our column in it.
getColumnIndex(String columnName)	This method will return the index of our column from the name of the column.
getPosition()	This method will return the current position of our cursor in our table.

Step by Step Implementation of SQLite DB

- Step 1: Create a New Project
- Step 2: Working with the activity_main.xml file
- Navigate to the **app > res > layout > activity_main.xml** and add the required code to that file.
- Step 3: Creating a new **Java class for performing SQLite operations/**Create a database using an SQL helper
- Navigate to the **app > java > your app's package name (where your MainActivity.java is present)> Right-click on it > New > Java class** and name it and add the required code to it
- Step 4: Working with the **MainActivity.java** file

SQLiteOpenHelper class

- A helper class to manage database creation and version management.
- You can 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.

Constructors of SQLiteOpenHelper class

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.

Parameters

context

Context- Context to use for locating paths to the the database This value may be null.

name

String- name of the database file, or null for an in-memory database This value may be null.
SQLiteDatabase.CursorFactory
Factory- 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

Constructors of SQLiteOpenHelper class

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

- Create a helper object to create, open, and/or manage a database. The database is not actually created or opened until one of **getWritableDatabase()** or **getReadableDatabase()** is called.
- Accepts input param: a concrete instance of DatabaseErrorHandler to be used to handle corruption when sqlite reports database corruption.

Note : SQLiteDatabase.CursorFactory: to use for creating cursor objects, or null for the default This value may be null. The reason of passing null is you want the standard SQLiteCursor behaviour. If you want to implement a specialized Cursor you can get it by by extending the Cursor class(this is for doing additional operations on the query results). And in these cases, you can use the CursorFactory class to return an instance of your Cursor implementation.

Methods of SQLiteOpenHelper class

public abstract void onCreate(SQLiteDatabase db)

- called only once when database is created for the first time.

public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)

- called when database needs to be upgraded.

public synchronized void close ()

- closes the database object.

public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)

- called when database needs to be downgraded.

Methods of SQLiteOpenHelper class

- **public void close ()**
  Close any open database object.

- **public String getDatabaseName ()**

  Return the name of the SQLite database being opened, as given

  to the constructor.

Public methods of SQLiteOpenHelper Class

- **public SQLiteDatabase getReadableDatabase ()**
  - Create and/or open a database.
  - This will be the same object returned by **getWritableDatabase()** unless some problem, such as a full disk, requires the database to be opened read-only.
  - In that case, a read-only database object will be returned. If the problem is
    fixed, a future call to **getWritableDatabase()** may succeed, in which case the read-only database object will be closed and the read/write object will be returned in the future.

SQLiteDatabase

A database object valid until **getWritableDatabase()** or **close()** is called.

SQLiteException

if the database cannot be opened

SQLiteDatabase class

SQLiteDatabase has methods to create, delete, execute SQL commands, and perform other common database management tasks.

void execSQL(String sql)

executes the sql query not select query.

long insert(String table, String nullColumnHack, ContentValues values)

inserts a record on the database. The table specifies the table name, nullColumnHack doesn't allow completely null values. If second argument is null, android will store null values if values are empty. The third argument specifies the values to be stored

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

updates a row.

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

returns a cursor over the resultset.

Cursor

- Cursors are **what contain the result set of a query made against a database in Android**.
- The Cursor class has an API that allows an app to read (in a type-safe manner) the columns that were returned from the query as well as iterate over the rows of the result set.
- A Cursor represents the result of a query and basically points to one row of the query result.
- This way Android can buffer the query results efficiently; as it does not have to load all data into memory.
- To get the number of elements of the resulting query use the getCount() method.
- Cursor also provides the getColumnIndexOrThrow(String) method which allows to get the column index for a column name of the table.
- A Cursor needs to be closed with the close() method call.
- A query returns a Cursor object.

Cursor

Use the Cursor interface as a data collection.

It is similar to a Cursor in PL/SQL in the way that it holds one or more rows returned by some queries with its pointer.

The following methods are available in the Cursor interface which iterate through the Cursor, setting the Cursor pointer to the desired position:

- **moveToFirst()**
- **moveToLast()**
- **moveToNext()**
- **moveToPrevious()**
- **moveToPosition(position)**

‘

```
Cursor cursor = db.query(TABLE_EMPLOYEE, new String[]
      {KEY_ID,KEY_NAME,KEY_AGE,KEY_CITY},  null, null, null, null, null);
```

table String: The table name to compile the query against.

columns String: A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.

selection String: A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.

selectionArgs String: You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.

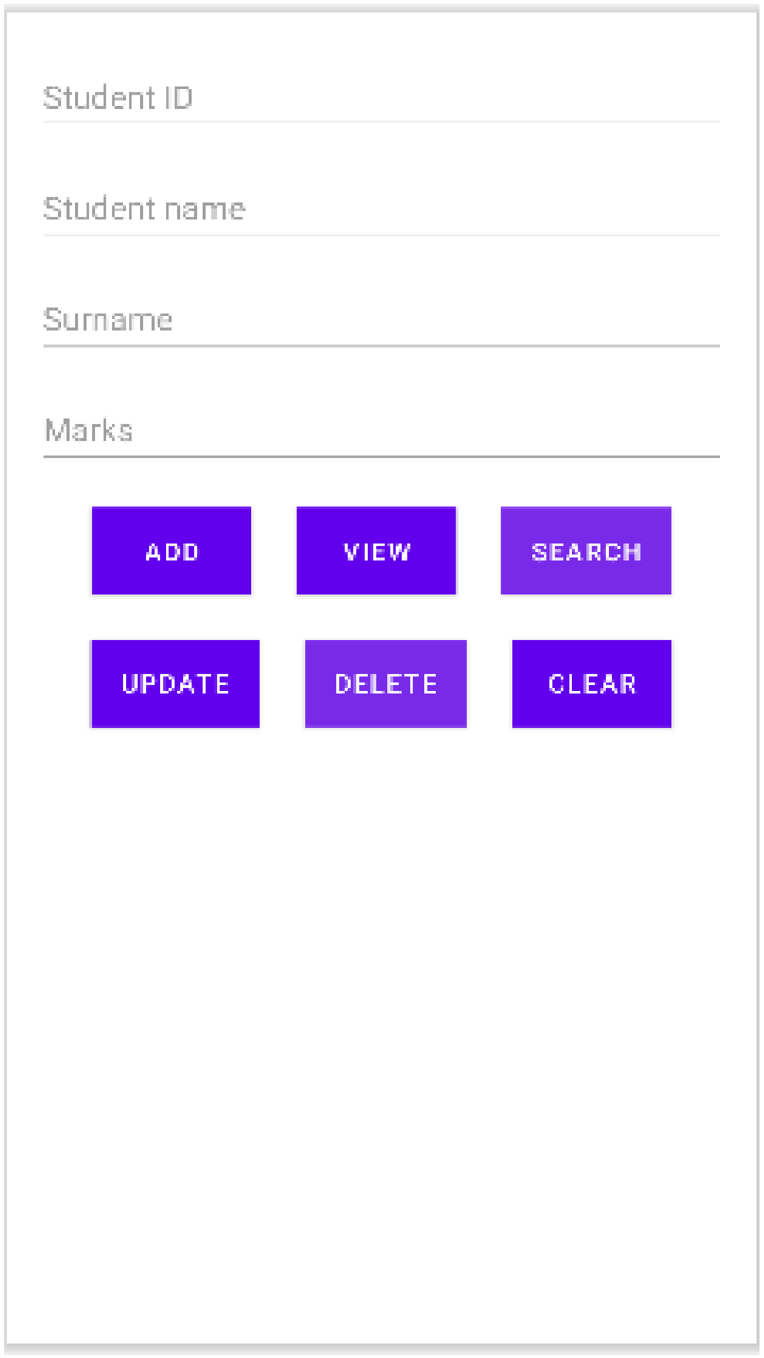
groupBy String: A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.

having String: A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.

orderBy String: How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing
null will use the default sort order, which may be unordered.

limit String: Limits the number of rows returned by the query, formatted as LIMIT clause. Passing null denotes no LIMIT clause.

Create an android application to insert, update ,select and delete records from SQLite Database.



activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:padding="16dp"
   android:gravity="center_horizontal"
   tools:context=".MainActivity">
   <!--    edit text for student ID    -->
   <EditText
       android:id="@+id/et_id"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginVertical="8dp"
       android:hint="Student ID" />
   <!--    edit text for student name    -->
   <EditText
       android:id="@+id/et_name"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginVertical="8dp"
       android:hint="Student name" />
   <!--    edit text for surname    -->
   <EditText
       android:id="@+id/et_surname"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginVertical="8dp"
       android:hint="Surname" />
   <!--    edit text for player's runs    -->
   <EditText
       android:id="@+id/et_marks"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginVertical="8dp"
       android:inputType="number"
       android:hint="Marks" />
```

```xml
<!--    container for add and clear buttons    -->
   <LinearLayout
       android:id="@+id/btns_add_clear"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:orientation="horizontal"
       android:gravity="center">
       <!--    add button  -->
       <Button
           android:id="@+id/btn_add"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_margin="12dp"
           android:background="#4CAF50"
           android:textColor="#FFFFFF"
           android:text="Add" />
       <!--    view button  -->
       <Button
           android:id="@+id/btn_view"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_margin="12dp"
           android:background="#4CAF50"
           android:textColor="#FFFFFF"
           android:text="View"/>
       <!--    search button  -->
       <Button
           android:id="@+id/btn_search"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_margin="12dp"
           android:background="#CCF44336"
           android:textColor="#FFFFFF"
           android:text="Search" />
   </LinearLayout>
```

```xml
<!--    container for update delete and cancel buttons    -->
   <LinearLayout
       android:id="@+id/btns_update_cancel"
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:visibility="visible"
       android:orientation="horizontal"
       android:gravity="center">
       <!--    update button  -->
       <Button
           android:id="@+id/btn_update"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_margin="12dp"
           android:background="#2196F3"
           android:textColor="#FFFFFF"
           android:text="Update" />
       <!--    delete button  -->
       <Button
           android:id="@+id/btn_delete"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_margin="12dp"
           android:background="#CCF44336"
           android:textColor="#FFFFFF"
           android:text="Delete" />
       <!--    clear button  -->
       <Button
           android:id="@+id/btn_clear"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_margin="12dp"
           android:background="#4CAF50"
           android:text="Clear" />
   </LinearLayout>
</LinearLayout>
```

DatabaseHelper.java

```java
package com.example.myapplicationcrud;

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

import androidx.annotation.Nullable;

public class DatabaseHelper extends SQLiteOpenHelper {
   private SQLiteDatabase sqLiteDatabase;
   public static final String DATABASE_NAME="Student.db";
   public static final String TABLE_NAME="Student";
   public static final String COL_2="NAME";
   public static final String COL_3="SURNAME";
   public DatabaseHelper(@Nullable Context context) {
       super(context, DATABASE_NAME, null, 1);
   }

   @Override
   public void onCreate(SQLiteDatabase sqLiteDatabase) {
sqLiteDatabase.execSQL("create table "+TABLE_NAME+"(ID Integer Primary key Autoincrement,NAME 
Text,SURNAME Text,MARKS integer)");
   }

   @Override
   public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
       sqLiteDatabase.execSQL("drop table if exists "+TABLE_NAME);
       onCreate(sqLiteDatabase);
   }
public boolean insertData(String name,String surname,String marks){
        sqLiteDatabase=getWritableDatabase();
        ContentValues contentValues=new ContentValues();
        contentValues.put(COL_2,name);
        contentValues.put(COL_3,surname);
        contentValues.put("MARKS",marks);
        long result=sqLiteDatabase.insert(TABLE_NAME,null,contentValues);
        if(result==-1)
            return false;
        else
            return true;
    }
    public Cursor getAllStudents(){
        sqLiteDatabase=getReadableDatabase();
        String query="select * from "+TABLE_NAME;
        return sqLiteDatabase.rawQuery(query,null);
    }
    public Cursor getStudentById(Integer id){
        sqLiteDatabase=getReadableDatabase();
        String query="select * from " +TABLE_NAME + " where ID = " + id;
        return sqLiteDatabase.rawQuery(query,null);
    }
    public void updateData(String id, String name, String surname, String marks){
        sqLiteDatabase=getWritableDatabase();
        ContentValues contentValues=new ContentValues();
        contentValues.put("ID", id);
        contentValues.put(COL_2, name);
        contentValues.put(COL_3, surname);
        contentValues.put("MARKS", marks);
        sqLiteDatabase.update(TABLE_NAME,contentValues,"ID=? ",new String[]{id});
    }
    public Integer deleteStudent(String id){
        sqLiteDatabase=getWritableDatabase();
       return sqLiteDatabase.delete(TABLE_NAME,"ID=? ",new String[]{id});
    }
}
```

MainActivity.java

```java
package com.example.myapplicationcrud;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
   DatabaseHelper databaseHelper;
EditText Id,Name, Surname, marks;
Button Add, btnView, search,btnUpdate, btnDelete, btnClear;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       databaseHelper=new DatabaseHelper(this);
       Id=findViewById(R.id.et_id);
       Name=findViewById(R.id.et_name);
       Surname=findViewById(R.id.et_surname);
       marks=findViewById(R.id.et_marks);
       Add=findViewById(R.id.btn_add);
       btnView=findViewById(R.id.btn_view);
       search=findViewById(R.id.btn_search);
       btnUpdate=findViewById(R.id.btn_update);
       btnDelete=findViewById(R.id.btn_delete);
       btnClear=findViewById(R.id.btn_clear);
       Add.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {

if(databaseHelper.insertData(Name.getText().toString(),Surname.getText().toString(),marks.getText().toString()))
                   Toast.makeText(MainActivity.this, "Data inserted successfully..", Toast.LENGTH_SHORT).show();
               else
                   Toast.makeText(MainActivity.this, "Failed to insert data.", Toast.LENGTH_SHORT).show();

           }
       });
```

```
btnView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Cursor result=databaseHelper.getAllStudents();
                if(result.getCount()==0){
                    Toast.makeText(MainActivity.this, "No entry exists.",
Toast.LENGTH_SHORT).show();
                    return;
                }
                StringBuffer buffer=new StringBuffer();
                while(result.moveToNext()){
                    buffer.append("ID "+result.getString(0)+"\n");
                    buffer.append("Name "+result.getString(1)+"\n");
                    buffer.append("Surname "+result.getString(2)+"\n");
                    buffer.append("Marks "+result.getString(3)+"\n");
                }
                AlertDialog.Builder builder=new
AlertDialog.Builder(MainActivity.this);
                builder.setCancelable(true);
                builder.setMessage(buffer).toString();
                builder.setTitle("Student Data");
                builder.show();
            }
        });
```

```
 search.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Cursor
result=databaseHelper.getStudentById(Integer.parseInt(Id.getText().toString()));
                if(result.getCount()==0){
                    Toast.makeText(MainActivity.this, "No entry exists.",
Toast.LENGTH_SHORT).show();
                    return;
                }
                StringBuffer buffer=new StringBuffer();
                while(result.moveToNext()){
                    buffer.append("ID "+result.getString(0)+"\n");
                    buffer.append("Name "+result.getString(1)+"\n");
                    buffer.append("Surname "+result.getString(2)+"\n");
                    buffer.append("Marks "+result.getString(3)+"\n");
                }
                AlertDialog.Builder builder=new
AlertDialog.Builder(MainActivity.this);
                builder.setCancelable(true);
                builder.setMessage(buffer).toString();
                builder.setTitle("Student " +Id.getText().toString()+ "Data");
                builder.show();
                result.moveToFirst();
                Name.setText(result.getString(1));
                Surname.setText(result.getString(2));
                marks.setText("" + result.getString(3));
            }
        });
```

```java
btnUpdate.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) {

databaseHelper.updateData(Id.getText().toString(),Name.getText().toStrin
g(),Surname.getText().toString(),marks.getText().toString());
               Toast.makeText(MainActivity.this, "Student
"+Id.getText().toString()+" Updated.", Toast.LENGTH_SHORT).show();
           }
       });
btnDelete.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
      if(databaseHelper.deleteStudent(Id.getText().toString())>0)
          Toast.makeText(MainActivity.this, "Student "+
Id.getText().toString()+ " Deleted.", Toast.LENGTH_SHORT).show();;
   }
});
btnClear.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View view) {
       Id.getText().clear();
       Name.getText().clear();
       Surname.getText().clear();
       marks.getText().clear();
   }
});
   }
}
```