Example 4:

Develop an application that makes use of SQLite Database.

SQLite Database.

In Android, there are several ways to store persistent data. **SQLite** is one way of storing app data. It is very lightweight database that comes with Android OS. In Android, integrating SQLite is a tedious task as it needs writing lot of boilerplate code to store simple data. Consider SQLite when your app needs to store simple data objects. Alternatively you can consider Room Persistence Library for better APIs and easier integration.

In this example we are going to learn basics of SQLite database CRUD operations, Creating Database, **Creating Tables, Creating Records, Reading Records, Updating Records and Deleting Records.**

Before that we need to know about some basic things:

What is SQLite?

SQLite is an SQL Database. I am assuming here that you are familiar with SQL databases. So in SQL database, we store data in tables. The tables are the structure of storing data consisting of rows and columns. We are not going in depth of what is an SQL database and how to work in SQL database. If you are going through this post, then you must know the Basics of SQL.

What is CRUD?

As the heading tells you here, we are going to learn the CRUD operation in SQLite Database. But what is CRUD? CRUD is nothing but an abbreviation for the basic operations that we perform in any database. And the operations are

- Create
- Read
- Update
- Delete

SQLiteOpenHelper

Android has features available to handle changing database schemas, which mostly depend on using the SQLiteOpenHelper class. SQLiteOpenHelper **is designed to get rid of two very common problems.**

1. When the application runs the first time – At this point, we do not yet have a database. So we will have to create the tables, indexes, starter data, and so on.

2. When the application is upgraded to a newer schema – Our database will still be on the old schema from the older edition of the app. We will have option to alter the database schema to match the needs of the rest of the app.

SQLiteOpenHelper wraps up these logic to create and upgrade a database as per our specifications. For that we'll need to create a custom subclass of SQLiteOpenHelper implementing at least the following three methods.

- **Constructor:** This takes the Context (e.g., an Activity), the name of the database, an optional cursor factory (we'll discuss this later), and an integer representing the version of the database schema you are using (typically starting from 1 and increment later).

```
public DatabaseHelper(Context context) {
super(context, DB_NAME, null, DB_VERSION);
}
```

- **onCreate(SQLiteDatabase db) :** It's called when there is no database and the app needs one. It passes us a SQLiteDatabase object, pointing to a newly-created database, that we can populate with tables and initial data.

- **onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) :** It's called when the schema version we need does not match the schema version of the database, It passes us a SQLiteDatabase object and the old and new version numbers. Hence we can figure out the best way to convert the database from the old schema to the new one.

We define a DBManager class to perform all database CRUD(Create, Read, Update and Delete) operations.

Opening and Closing Android SQLite Database Connection

Before performing any database operations like insert, update, delete records in a table, first open the database connection by calling **getWritableDatabase()** method as shown below:

```
public DBManager open() throws SQLException {
    dbHelper = new DatabaseHelper(context);
    database = dbHelper.getWritableDatabase();
    return this;
}
```

The dbHelper is an instance of the subclass of SQLiteOpenHelper. To close a database connection the following method is invoked.

```
public void close() {
    dbHelper.close();
  }
```

Inserting new Record into Android SQLite database table

The following code snippet shows how to insert a new record in the android SQLite database.

```
public void insert(String name, String desc) {
    ContentValues contentValue = new ContentValues();
    contentValue.put(DatabaseHelper.SUBJECT, name);
    contentValue.put(DatabaseHelper.DESC, desc);
    database.insert(DatabaseHelper.TABLE_NAME, null, contentValue);
  }
```

Content Values creates an empty set of values using the given initial size. We'll discuss the other instance values when we jump into the coding part.

Updating Record in Android SQLite database table

The following snippet shows how to update a single record.

```
public int update(long _id, String name, String desc) {
    ContentValues contentValues = new ContentValues();
    contentValues.put(DatabaseHelper.SUBJECT, name);
    contentValues.put(DatabaseHelper.DESC, desc);
    int i = database.update(DatabaseHelper.TABLE_NAME, contentValues,
DatabaseHelper._ID + " = " + _id, null);
    return i;
  }
```

Android SQLite – Deleting a Record

We just need to pass the id of the record to be deleted as shown below.

```
public void delete(long _id) {
    database.delete(DatabaseHelper.TABLE_NAME, DatabaseHelper._ID + "=" + _id, null);
  }
```

Android SQLite Cursor

A Cursor represents the entire result set of the query. Once the query is fetched a call to **cursor.moveToFirst()** is made. Calling moveToFirst() does two things:

- It allows us to test whether the query returned an empty set (by testing the return value)
- It moves the cursor to the first result (when the set is not empty)

The following code is used to fetch all records:

```
public Cursor fetch() {
    String[] columns = new String[] { DatabaseHelper._ID, DatabaseHelper.SUBJECT,
DatabaseHelper.DESC };
    Cursor cursor = database.query(DatabaseHelper.TABLE_NAME, columns, null, null,
null, null, null);
    if (cursor != null) {
       cursor.moveToFirst();
    }
    return cursor;
  }
```