SQLite

Note: It is recommended to revise Android Application development exercises uploaded under section- " Android Part 1 " before completing this issue.

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. A **database engine** is the underlying software component that a **database** management system (DBMS) uses to create, read, update, and delete data from a **database**. SQLite is an SQL Database engine, meant for embedded devices. It is a Relational Database Engine, with low memory foot-print.

SQL Vs NoSQL

SQL databases are primarily called RDBMS or Relational Databases. NoSQL databases are primarily called as Non-relational or distributed database

Parameter	SQL	NoSQL
Query Language	Structured query language (SQL)	No declarative query language
Type	SQL databases are table based databases	NoSQL databases can be document based, key-value pairs, graph databases
Schema	SQL databases have a predefined schema	NoSQL databases use dynamic schema for unstructured data.
Ability to scale	SQL databases are vertically scalable	NoSQL databases are horizontally scalable
Examples	Oracle, Postgres, and MS-SQL, SQLite.	MongoDB, Redis , Neo4j, Cassandra, Hbase.
Best suited for	An ideal choice for the complex query intensive environment.	Not good fit complex queries.
Hierarchical data storage	SQL databases are not suitable for hierarchical data storage.	More suitable for the hierarchical data store as it supports key-value pair method.
Best Used for	RDBMS database is the right option for solving ACID (Atomicity, Consistency, Isolation,Durability) problems.	NoSQL is a best used for solving data availability problems
Importance	It should be used when data validity is important	Use when it's more important to have fast data than correct data
ACID vs. BASE Model	ACID(Atomicity, Consistency, Isolation, and Durability) is a standard for RDBMS	Base (Basically Available, Soft state, Eventually Consistent) is a model of many NoSQL systems

A brief comparison of the above strategies for an IoT System can be found here. "MySQL and NoSQL database comparison for IoT application" -Paper available in nalanda.

Android uses SQLite for its on-device database. We have to use SQL statements for creating and updating the database.

https://developer.android.com/reference/android/database/sqlite

All the classes used in the project can be understood in detail by referring to the above site.

Maintaining the database is taken care of by the Android platform.
Some users of SQLite in Android are Contacts, Gallery, `Music`.

Read more here: https://www.sqlite.org/index.html

SQLite in Android

Example Application

- Develop an application to create an SQLite Database and store ambient light (in lux) recorded when a button 'Record' is clicked.
- Show the recorded data when 'Show' button is clicked

Database Structure:

The database structure is in the form of tables. In our database, we will store the light at the time when the 'Record' button is clicked. So in one row of the table, we will store the time in 'hours', 'minutes' and 'seconds' and the corresponding 'light'. 'Sl No' will be there for indexing purpose
We will have one table per day. So every day, when the first time the record button is pressed a new table will be created. Further data will be recorded as and when the record button is clicked in the same the table on that day.

Fig shows the proposed table architecture.

Sl.No	hours	minutes	seconds	light

Each table should have a name. Each column is recognised by a string called Key. Here keys are 'Sl No', 'hours', 'minutes', 'seconds' and 'light'.
There should be a 'PRIMARY KEY', which is used as an index in the table. It should be unique. Here we are using 'Sl No' as the primary key

There are 'schemas' for creating, deleting, inserting, etc into the table. You can refer to any SQLite documentation for these schemas.

Code for the Project-'SQLite Test'.

Following points are to be taken care while creating the project
The 'activity_main.xml' should be a Relative Layout, so that you have the liberty to place the views in different places.

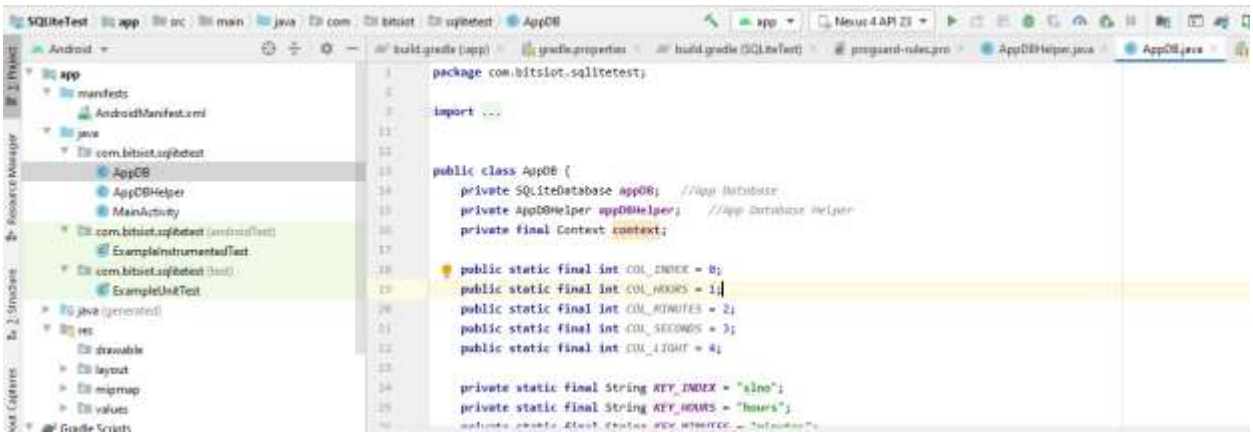
Create a Text View at the top and two buttons 'Record' and 'Show' at the bottom left and right corners. Since the data will grow as time passes, let us encapsulate the text view by a 'ScrollView'.

Code for the Database – AppDBHelper.java

'SQLiteOpenHelper' class is provided to take care of creation, deletion, upgrade of SQLite database.
So, create a Java class named 'AppDBHelper', which should be derived from SQLiteOpenHelper.
You need to put the code for creation of your database inside the 'onCreate' method of this class.
We have to give a name to the database. Here it is 'light.db'. Assign this to the variable DB_NAME.
Each table we will give name like 'light_<date>'. So put the common string to the variable DB_TABLE_NAME.
Create variables to store the keys in the table such as KEY_SLNO, KEY_HOURS, KEY_MINUTES, KEY_SECONDS and KEY_LIGHT.
The constructor for this class is AppDBHelper and it takes the db name. You may pass the db name through this constructor rather than defining it.
Inside the 'onCreate' method of AppDBHelper, create the table if it doesn't exist. Corresponding schema is here. Since we are creating tables per date basis, we need to know the date of today. This can be achieved by creating a 'Calendar' instance and getting today's date from it.

With today's date, create the table name and create the table for today if it doesn't exist with the the help of the schema.
Two extra functions 'createTable' and 'deleteTable' are created so that they can be used to explicitly create or delete tables whenever required.
In the method 'onUpgrade', we need to put the code for recreating the table again.

Code for the Database – AppDB



Now, we need to create another class 'AppDB' which will instantiate AppDBHelper inside it. From the other classes, this AppDB will be the front end to access the Database. Hence we need to have methods for inserting, deleting and querying.
So create the class 'AppDB'. Define the variables containing the DB name, DB table name and the Keys as done in AppDBHelper.
Implement the constructor. Inside the constructor, instantiate AppDBHelper instance.
Add a method 'open' for opening the database. Since we are going to update the database, we need to create a writable database. There are scenarios, where we take and external database and use it for reading only. In such cases, we may open the database in read-only mode. These are accomplished by 'getWritableDatabase' and 'getReadableDatabase' method of the AppDBHelper instances. The SQLiteOpenHelper take care of these methods.
In 'close' function, we will close the DB.

'insert' method again takes the help of AppDBHelper's ''insert' method. We have to pass the DB table name, and the entry to be inserted to this method. The entry to be inserted should be of type 'ContentValues'. So we will create an object of ContentValue called 'newEntry'. Then add the fields to this 'newEntry'. In our case, when the 'Record' button is pressed, we have to note the 'hour', 'minute', 'second' and the 'light' at that time. In fact these are the keys. So we will add these three values against the corresponding keys to 'newEntry' and pass it to the 'insert' function. The table name will be formed from the data passed.
There are two more methods, we have implemented here : getAllRows and get Specific Rows based on the key passed. In case of get All Rows, we have to call the 'query' method of the AppDBHelper instance. The format is shown. In case of getting a specific row , we have to call the 'query' method of

the AppDBHelper instance with the specific key being mentioned in the format shown. Note that the 2nd parameter is the array of String containing the keys, which will result in returning these values corresponding to the keys passed. The third parameter is for matching keys passed. For all rows it is 'null' so that it will return all rows.
The query method returns an Android 'Cursor'. We have to use that Cursor and extract the information present in it and use them.

Java Code for MainActivity

In the MainActivity.java,

- Instantiate the views.
- Instantiate the DB.
- Use Calendar Instance for getting Date and Time.
- Use SensorManager for getting environment Light.
- Implement the onClickListeners for the Buttons.

Here is how SensorManager is used inside the MainActivity.
In order to use Sensor Manager, the MainActivity needs to implement the interface 'SensorEventListner'.
For that it needs to have two methods implemented: onSensorChanged and onAccuracyChanged.
Inside onCreate, we will create an instance of SensorManager.
Then we will get the Ambient Light Sensor instance.
We will register the sensor event listener for the sensor instance created inside onResume and deregister it inside the onPause method. It is to make sure that the sensor is active only when the application is in the foreground. You can do otherwise also by registering the listener in onCreate. Then it will be recording the sensor output even when the application is in the background.
We will have an integer variable 'light' (ideally it should be floating point) and will update this variable inside 'onSensorChanged' method. SensorEvent.values[0] will provide this value.
Similarly here is how the Calendar is used to get the current date and time.
We will create an instance of Calendar and we will get Date and Time from this. Calendar instance has to be created each time we want to record. Hence it should be kept inside the onClickListener of the Record button.
Inside the onClickListner of Record button, we will get the date and time from the calendar event and will take the value of the 'light' variable and call 'insert' method of the AppDB instance.
Inside the onClickListner of Show button, we will query all rows of the AppDB, extract values from the cursor, for a string having each row in one line and display the values in the TextView.

Assignment III:

You can use, the code provided for this exercise, and the previous exercise (Android-Part 1) for reference.

No of times the record button is pressed should be stored in the light 'column'.

Every time a button press is detected, the light column data along with the system time should be sent to the firebase cloud.

When the show button is pressed, the data recorded until then should be shown on the display.

You may use this link as a reference

https://www.c-sharpcorner.com/article/real-time-data-store-in-firebase-using-android-studio/

Deadline- June 15th 2020.