Lab 8.3 – Database

Overview:

In this lab you will create a database that the EventService will use to persist the event objects on the device.

Step	Description
1.	Create a new class that extends SQLiteOpenHelper which will be used to create and upgrade the database.
a.	Create a new packaged called “com.garagze.database”. Create a new class in package “com.garagze.database” called “GaragzeDbHelper” and make it a subclass of “SQLiteOpenHelper”. You will receive errors because of missing constructors. Generate the constructors from the super class.

b.	Write the database create method. This method will run the DDL to create a new database.
	
	Declare the “create” statement as a string constant. Add this code to the “GaragzeDbHelper.java” file.
	
	`private static final String DATABASE_CREATE =`
	`"create table events (" +`
	`"_id integer primary key autoincrement, " +`
	`"id text, " +`
	`"date date, " +`
	`"title text, " +`
	`"street text, " +`
	`"city text, " +`
	`"state text, " +`
	`"zip text, " +`
	`"latitude double, " +`
	`"longitude double, " +`
	`"description text, " +`
	`"rating double, " +`
	`"distance double " +`
	`");";`
	
	Now create the method which will run the DDL create statement.
	
	In “GaragzeDbHelper.java” and the following method:
	
	`@Override`
	`public void onCreate(SQLiteDatabase database) {`
	`Log.v("GaragzeDBHelper", "Create Table using " + DATABASE_CREATE);`
	`database.execSQL(DATABASE_CREATE);`
	`}`
	
	This method uses the string you’ve already created and writes to the log so you can verify that the DDL runs. This method is called automatically by Android when it detects that the database must be created.

c.	Write the database upgrade method. This method is run when Android detects that the database version has changed. This method will alter the database. It may do so by dropping the old database and creating

a new run or by executing alter statements to upgrade the database in place. Altering the database is more complex since it must be determined which version the existing database is.

We will chose a simple strategy. When the database version changes, drop the table if it exists and create a new table.

In "GaragzeDbHelper.java" and the following code:

```
private static final String DATABASE_DROP =
   "drop table if exists events";

@Override
public void onUpgrade
   (SQLiteDatabase database, int oldVersion, int newVersion) {
   Log.v("GaragzeDBHelper",
      "Drop Table, Upgrading from " +oldVersion + " to " + newVersion);
   database.execSQL(DATABASE_DROP);
   onCreate(database);
}
```

2. Create a new DbAdapter class to provide access to the database. This class does not extend any existing Android class but its use is part of the recommended best practices for working with databases in Android. This is the class that will be used by your application code to provide access to the database. You shouldn't access the database classes directly.

   Create a new class in package "com.garagze.database" called "GaragzeDbAdapter"

3. Provide DbAdapter with an implementation of the DbHelper

   a. Add the following code to "GaragzeDbAdapter.java". This code uses "composition" to provide an implementation of "GaragzeDbHelp" inside the adapter. The constructor creates the instance of "GaragzeDbHelper".

```
private static final int DATABASE_VERSION = 1;
private static final String DATABASE_NAME = "garagze.db";

private GaragzeDbHelper dbHelper;

public GaragzeDbAdapter(Context context) {
    dbHelper = new GaragzeDbHelper(
         context, DATABASE_NAME, null, DATABASE_VERSION);
}
```

   Note: Notice that the constructor requires a context object.

   b. In "GaragzeDbAdapter.java" Create methods for opening and closing the database. These are just delegates to the real method in the helper class.

```
public SQLiteDatabase db;

public GaragzeDbAdapter open() {
      db = dbHelper.getWritableDatabase();
      return this;
}

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

	}
4.	Instantiate the "GaragzeDbAdapter" class. Because this class requires a context object when it is created, we must find a place in our code that has access to the context and will be executed when the application starts up. Fortunately, we already have the ideal place for this – the application object.

a.	In application object "GaragzeApplication.java" in package "com.garagze", create a new DbAdapter and provide an access method for it.
	
	First, add a static method to hold a reference to the adapter
	
	`private static GaragzeDbAdapter dbAdapter;.`
	
	Next, create the adapter in the "onCreate" method (after getting the application context) and open the database.
	
	`GaragzeApplication.dbAdapter = new GaragzeDbAdapter(context);`
	`GaragzeApplication.dbAdapter.open();`
	
	Then add a static method to return a reference to the adapter.
	
	`public static GaragzeDbAdapter getDbAdapter() {`
	`  return dbAdapter;`
	`}`

5.	Run and test. Review log cat for logging from the creation methods. Find the DDL for creating the database. Verify that the database file has been created by using the File Explorer in DDMS. Look for the file "garagze.db" in the "/data/data/com.garagze/databases" directory. If you need to re-run the test, be sure to increment the version number in "GaragzeDbAdapter" to drop and re-create the database.
6.	Write code to query the database. A database query is returned as a "Cursor". Results must be extracted from the cursor. This is similar in concept to "ResultSet" in JDBC but the API is not the same.

a.	Create the query method in the adapter "GaragzeDbAdapter.java"
	
	`private Cursor getAllEntries() {`
	`  String[] columns = new String[10];`
	`  columns[0] = "id";`
	`  columns[1] = "date";`
	`  columns[2] = "title";`
	`  columns[3] = "street";`
	`  columns[4] = "city";`
	`  columns[5] = "rating";`
	`  columns[6] = "distance";`
	`  columns[7] = "description";`
	`  columns[8] = "latitude";`
	`  columns[9] = "longitude";`
	`  return db.query("events", columns, null, null, null, null, null);`
	`}`
	
	Note: Review the javadoc for the query method to understand the null parameters.

b. The application should not have to interact direction with the Database API so put a wrapper around the "getAllEntries" method to return the data as event objects. The "getAllEntries" method can't be called from outside the adapter since it is private.

Add the following method to "GaragzeDbAdapter".

```
public List<Event> getAllEvents() {
   ArrayList<Event> events = new ArrayList<Event>();
   Cursor cursor = getAllEntries();
   if (cursor.moveToFirst()) {
      do {
         Event event = new Event();
         event.setId(cursor.getString(0));
         event.setDate( new java.util.Date( cursor.getLong(1)) );
         event.setTitle(cursor.getString(2));
         event.setStreet(cursor.getString(3));
         event.setCity(cursor.getString(4));
         event.setRating(Float.parseFloat(cursor.getString(5)));
         event.setDistance(Double.parseDouble(cursor.getString(6)));
         event.setDescription(cursor.getString(7));
         event.setLatitude(cursor.getDouble(8));
         event.setLongitude(cursor.getDouble(9));
         events.add(event);
         Log.v("GaragzeDbAdapter", "Get event: id="+event.getId());
      } while (cursor.moveToNext());
   }
   return events;
}
```

7. Create a method for adding data to the database.

a. Create a new method called "insertEvent" in "GaragzeDbAdapter". It should take a Event object as a parameter.

```
public long insertEvent(Event event) {
      ContentValues values = new ContentValues();
      values.put("id", event.getId());
      values.put("date", Long.toString( event.getDate().getTime()) );
      values.put("title", event.getTitle());
      values.put("street", event.getStreet());
      values.put("city", event.getCity());
      values.put("rating", event.getRating());
      values.put("distance", event.getDistance());
      values.put("description", event.getDescription());

      values.put("latitude", event.getLatitude());

      values.put("longitude", event.getLongitude());

      Log.v("GaragzeDbAdapter", "Insert event: id="+event.getId());
      return db.insert("events", null, values);
}
```

This code creates a new "android.content.ContentValues" which is simply a map holding the name/value pairs for the columns in the database. The "insert" method takes the map and adds it as a new row to the specified table. Notice that the application doesn't need to work with "ContentValues" only events.

	Note: Be sure to review the javadoc for the "insert" method. Optional: We have not implemented all the CRUD method. If you have time, implement the following additional methods. • deleteEvent(String eventId) • getEvent(String eventId) • updateEvent(Event event) Be sure to create unit tests to verify that your new methods are working correctly.
8.	Change the "EventService" to use the database rather than a "List" for persisting the event data. By using a database, we can make the new event data persist when we re-run the app.

a.	Modify the "getAllEvents" method in "com.garagze.serivce.EventService.java" to put items in the database and lookup items from the database. Add the following code to the end of the "getAllEvents" method.
	
	`for (Event event : events) {`
	`  GaragzeApplication.getDbAdapter().open().insertEvent(event);`
	`}`
	
	`events = GaragzeApplication.getDbAdapter().open().getAllEvents();`
	
	Note: This code will add the events from the HTTP request to the database every time it is called. You could write code so that the HTTP all is only done once and if the events list is not null, just return it.

b.	Modify the "addEvent" method in "com.garagze.serivce.EventService.java" to add the event to the database by adding the following code to the end of the method. `GaragzeApplication.getDbAdapter().open().insertEvent(event);`
9.	Now that there is actual data in the database. We can use some of the Android SQLITE tools to explore the database and run SQL against it. You can find out how to use the tools at the following URLs. http://developer.android.com/guide/developing/tools/adb.html http://developer.android.com/guide/developing/tools/sqlite3.html
a.	Change to the tools directory. The tools are in "<sdk_home>/platform-tools"

b.	Start the adb shell
	
	`adb shell`
	
	You should see a "#" prompt on the command line.
	
	Note: Mac users should run the following command (in the correct directory) instead:
	
	`./adb shell`

c.	Start the sqlite admin shell `sqlite3 /data/data/com.garagze/databases/garagze.db`

	You should see a “sqlite>” prompt on the command line.

d.	Verify that the data is in the database
	`.schema events`
	`select * from events;`

e.	Exit both shells.
	`.exit`
	`exit`