Optional SQLite Tutorial

Since Sunshine will be using a SQLite database to store weather data, you should have a basic understanding of SQLite and its commands before continuing this lesson. This is an optional exercise for anyone new to SQL databases or anyone in need of a refresher.

Introduction

SQLite is a relational database management system, which is an implementation of SQL (Structured Query Language). It comes packaged with the Android OS as a C++ library so that apps can have private databases. SQL can be used to create, search and maintain databases. In this tutorial, you will learn about the syntax and usage of SQL and see how to create and manage a small database on your computer. The commands you learn here are similar to the SQL commands you will write for your Sunshine app, in order to store and retrieve weather data from a SQLite database.

Get SQLite

1. Download the SQLite command line shell and install. You can follow this tutorial or go directly to http://sqlite.org/download.html
2. Open terminal and navigate to a folder of your choice where you will save your database. Create a new database file called sunshine.db and start the SQLite shell (which will recognize your SQL commands) by typing:

   ```
   sqlite3 sunshine.db
   ```

3. For a list of all commands:

   ```
   .help
   ```

4. One of the commands is to list out all databases. It should display one database called main and the file location on your computer for sunshine.db.

   ```
   .databases
   ```

   In an app, you can have multiple databases. Our Sunshine app will only have a single database, and that database can contain multiple tables.

Create A Database Table

1. A table is a collection of rows and columns like a spreadsheet. Use the CREATE TABLE statement to create a new database table called "weather." Each row will be one day's worth of weather data. It should have 6 columns of data: ID, date, min temperature, max temperature, humidity, and pressure.

   In the CREATE TABLE statement, each column definition is separated by commas, where you provide the column name and datatype for that column. We also specify that the column should be non-null. We specify the _id column to be the primary key and it's an integer.

   ```
   CREATE TABLE weather( _id INTEGER PRIMARY KEY, date TEXT NOT NULL, min REAL NOT NULL, max REAL NOT NULL, humidity REAL NOT NULL, pressure REAL NOT NULL);
   ```

   The list of possible SQLite data types is a useful resource, or you can see this tutorial.

   Note: SQLite keywords, such as CREATE TABLE or PRIMARY KEY, are capitalized for ease of readability to distinguish them from the table and column names that we've selected, but you can lowercase the keywords if you want.

   Note: This is not the full table you'll be using in Sunshine, this is just a simpler version of the table.

2. Use this command to list out all tables. Ensure that the weather table was created.

   ```
   .tables
   ```

3. Use a SELECT statement to return out all rows in the weather table. The * is a symbol that means "all of the columns". At this time, nothing will be returned because the table is created, but there is no data in the table yet.

   ```
   SELECT * FROM weather;
   ```

4. At any point, you can find out the schema of how the tables were created in the database

   ```
   .schema
   ```

Insert rows

1. Use an INSERT statement to insert a new row of data into the weather table. The following INSERT statement inserts a row into the weather table for June 25th, 2014, which had a low of 16 degrees, a high of 20 degrees, 0 humidity and 1029 pressure. The _id of this row is 1.

   ```
   INSERT INTO weather VALUES(1,'20140625',16,20,0,1029);
   ```

2. Query for all rows in the weather table, and you should see the one row of data you just inserted.

   ```
   SELECT * FROM weather;
   ```

3. To have the column name be printed out as well (for easier readability as to what value corresponds to which column), turn the header on. Then do the query again.

   ```
   .header on
   SELECT * FROM weather;
   ```

4. Experiment by inserting another 3 rows of data into the weather table. `INSERT INTO weather VALUES(2,'20140626',17,21,0,1031);INSERT INTO weather VALUES(3,'20140627',18,22,0,1055);INSERT INTO weather VALUES(4,'20140628',18,21,10,1070);`

   Query for all rows to verify they were inserted properly.

   ```
   SELECT * FROM weather;
   ```

Query rows

1. Practice doing queries where you provide a selection WHERE clause to narrow down the number of rows that are returned in the result. Always remember the semicolon at the end of a statement!

   For all possible SQLite operators, see this link.

   This query returns rows from the weather table where the date column exactly equals the 20140626.

```
SELECT * FROM weather WHERE date == 20140626;
```

2. This query returns rows from the weather table where the date column is between 20140625 and 20140628. However, all columns are not returned, we just return the 4 specified columns (_id, date, min, and max) of the rows that match the query.

```
SELECT _id,date,min,max FROM weather WHERE date > 20140625 AND date < 20140628;
```

3. This query returns rows where the minimum temperature is greater than or equal to 18. Based on those matching rows, we order them based on increasing (also known as ascending or "ASC" for short) max temperature. The first row of the result that is printed out to the command line will be the row (with min temperature >= 18) with max temperature that is lowest out of all rows, so that subsequent rows will have higher max temperature.

```
SELECT * FROM weather WHERE min >= 18 ORDER BY max ASC;
```

Update rows

1. You can also update existing rows in the database with an UPDATE statement. This statement updates the weather table by setting the minimum temperature to be 0 and maximum temperature to be 100 for rows where the date is greater than 20140626 but less than 20140627.

```
UPDATE weather SET min = 0, max = 100 where date >= 20140626 AND date <= 20140627;
```

When you print out the whole weather table again, you can see that 2 rows were changed.

```
SELECT * FROM weather;
```

Delete rows

1. Use a DELETE statement to delete rows from a database table that match the given selection clause. In this case, we delete any rows from the weather table where humidity is not equal to 0.

```
DELETE FROM weather WHERE humidity != 0;
```

Add columns

1. If you have released a version of your app to users, and then decide you need to change the database schema, such as adding columns, then you'll need to upgrade your database. You can alter existing tables, by using the ALTER TABLE command.

   Note: In general, you shouldn't alter a table to remove a column because you're deleting data and other tables could depend on that column. Instead you can just null out all values in that column.

   This statement alters the weather table by adding another column to the table called description, which will always be non-null and contain text. It will also default to the value 'Sunny' if no value is provided. In reality, you would choose a more reasonable default, but this is just for example purposes.

   ```
   ALTER TABLE weather ADD COLUMN description TEXT NOT NULL DEFAULT 'Sunny';
   ```

   Verify that the new description column exists when you query all rows and all columns.

   ```
   SELECT * FROM weather;
   ```

Delete table

1. Delete the weather table by using the DROP TABLE command. Verify there are no more tables in the database.

   ```
   DROP TABLE weather;
   .tables
   ```

These are just the basics. Feel free to play around with SQLite some more. See this link: http://www.sqlite.org/cli.html

When you're done, enter `.quit` to exit, and you can move onto the quiz on the next page!