Problem Set 1: SQL

1 Introduction

The purpose of this assignment is to introduce you to the SQL programming language. SQL is a declarative language, or a “relational calculus” in which you specify the data you are interested in in terms of logical expressions.

This assignment is designed to provide you with some hands on experience writing SQL queries. We will be using the PostgreSQL open source database, which provides a fairly standard implementation of SQL (in reality, there are slight variations between the SQL dialects – especially with respect to some advanced features, built-in functions, and so on – between all major vendors.) The SQL tutorial provided with the Postgres documentation at [http://www.postgresql.org/docs/7.4/static/tutorial-sql.html](http://www.postgresql.org/docs/7.4/static/tutorial-sql.html) provides a good introduction to the basic features of SQL; after following this tutorial you should be able to answer most of the problems in this problem set (the last three questions are a bit tricky). We have set up a Postgres database server for you to use; you will need to download and install the Postgres client (psql) on a local machine – more details are given in Section 3 below.

2 Sensor Data

In this problem set, you will write a series of queries using the SELECT statement over a table called expt_table of light and temperature readings collected from three wireless nodes with special sensing hardware. The data represents a day’s worth of light and temperature readings, collected every 30 seconds from each sensor node. Because these sensors that were used are lossy, there are some times when sensor readings are missing. The schema of the sensor data is as follows:

```
result_time : timestamp  | epoch : int  | nodeid : int  | light : int  | temp : int  | voltage : int
```

Where the fields are as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>result_time</td>
<td>The time the record was inserted into the database.</td>
</tr>
<tr>
<td>epoch</td>
<td>The sensor-node sequence number of the record. Different results from different sensors with the same epoch number should arrive in the database at about the same time. If they have different times, it can indicate a failure in the software that runs on the sensor – one of the queries below asks you to look for such failures.</td>
</tr>
<tr>
<td>nodeid</td>
<td>The id of the sensor node that produced this reading. There are three sensors in this data set, numbered 1, 2, and 3.</td>
</tr>
<tr>
<td>light</td>
<td>The light reading from the sensor, in raw units relative to the minimum and maximum brightness the sensor can perceive. This is a 10-bit sensor, with a value of 0 corresponding to darkness 1024 representing maximum brightness. The calib_light table maps raw units into Lux, which is a commonly used measure of brightness.</td>
</tr>
<tr>
<td>temp</td>
<td>The temperature reading from the sensor, in raw units. The calib_temp table maps raw units into degrees Celsius.</td>
</tr>
<tr>
<td>voltage</td>
<td>The voltage on this device, in raw units.</td>
</tr>
</tbody>
</table>

The two tables, calib_light and calib_temp each have two fields:

```
raw : integer  | calib : integer
```

Where raw is the raw sensor value and calib is the calibrated value, in Lux (in the case of calib_light) or degrees Celsius (in the case of calib_temp.)

3 Connecting to the Database

To connect to the database, you will need the psql Postgres client. Some recent versions of Linux come with this tool pre-installed, and you can download binaries for many platforms from the Internet; if you are using a package manager such as rpm,
dpkg/apt, or yum, you should install the packages “postgresql-libs” and “postgresql”. We have made a binary version of the psql tool available for Linux-based Athena machines at http://db.csail.mit.edu/6.893/psql.tar.gz. To use it, download it to your Athena directory, expand the archive (using a command like tar xvf psql.tar.gz) and follow the instructions in the README file in the postgres_client directory. There is also a Windows version of the Postgres client available, though it is somewhat more involved to install; the easiest way is to install the Cygwin system, which can be downloaded from http://www.cygwin.com.

To connect to the database, type the following (assuming psql is in your Unix path):

```
psql -h hancock.lcs.mit.edu 6.893 student
```

You MUST do this from an MIT machine (e.g., with IP Address 18.x.x.x).

You should now be able to type SQL queries directly. All queries in Postgres must be terminated with a semi-colon. For example, to get a list of all records in the `expt` table, you would type:

```
SELECT * FROM expt_table;
```

You can use the `\h` and `\?` commands for help on SQL syntax and Postgres specific commands, respectively. For example, `\h SELECT` will give the syntax of the SELECT command.

## 4 Questions

1. Write a query (using the SELECT statement) that will compute times and ids when any sensor’s light reading was above 550. Show both the query and the first few lines of the result.

2. Write a query that will compute the average light reading at sensor 1 between 6 PM and 9 PM (inclusive of 6:00:00 PM and 9:00:00 PM). Show the query and the result.

3. Write a single query that computes the average temperature and light reading at every sensor between 6 PM and 9 PM, but exclude any sensors whose maximum voltage was greater than 418 during that time period. Show both the query and the result.

4. Write a query that computes the average calibrated temperature readings from sensor 2 during each hour, inclusive, between 6 PM and 9 PM (i.e., your answer should consist of 4 rows of calibrated temperatures.)

5. Write a query that computes all the epochs during which the results from sensors 1 and 2 arrived more than 1 second apart. Show the query and the result. Note that you can use the difference (minus) operator on timestamps in Postgres, and that the string ’1 second’ refers to a period of 1 second.

6. Write a query that determines epochs during which one or two of the sensors did not return results. Show your query and the first few results, sorted in by epoch number. You may wish to use a nested query – that is, a SELECT statement within the FROM clause of another SELECT statement.

7. Write a query that produces a temperature reading for each of the three sensors during any epoch in which any sensor produced a reading. If a sensor is missing a value during a given epoch, your result should report the value of this sensor as the most recent previously reported value. If there is no such value (e.g., the first value for a particular sensor is missing), you should return the special value ’null’. You may wish to read about the CASE and OUTER JOIN SQL statements.

8. Write a query that determines epochs during which all three sensors did not return any results. Note that this is a deceptively hard query to write – you may need to make some assumptions about the frequency of missing epochs.

## 5 Places to Look for More Help on SQL

The Postgres manual provides a good introduction to the basic features of SQL. There are a number of other online guides available; in particular, http://sqlzoo.net provides a “Gentle Introduction to SQL” which is quite thorough and includes many examples.