Problem Set 1: SQL

Assigned: 2/13/2012

Due: 2/21/2012 11:59 PM

Submit to the 6.830 Stellar Site (`https://stellar.mit.edu/S/course/6/sp12/6.830/homework/`)

1 Introduction

The purpose of this assignment is to provide you with hands-on experience with 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.

We will be using the SQLite open source database, which provides a standards-compliant SQL implementation. In reality, there are slight variations between the SQL dialects of different vendors—especially with respect to advanced features, built-in functions, and so on. The SQL tutorial at `http://sqlzoo.net/`, 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 few questions may be a bit tricky). You may also wish to refer to Chapter 5 of "Database Management Systems." This assignment mainly focuses on *querying* data rather than modifying it. `http://sqlzoo.net/` includes a reference section that describe how to create tables and modify records; you should read it so that you are familiar with these aspects of the language.

SQLite a very easy database to use because a database in simply stored in a file. We have put a database file on athena for you; you may download this file to your local machine and run SQLite there, or log in to athena and use it. More details are given in Section 3 below.

2 Publication Data

In this problem set, you will write a series of queries using the `SELECT` statement over a publication database containing information about approximately 2,000,000 papers published in computer science conferences and journals (this data was derived from the DBLP system, maintained by Michael Ley at `http://www.informatik.uni-trier.de/~ley/db/`).

This database consists of four tables: an `authors` table, containing the names of authors, the `venues` table, containing information about conferences or journals where papers are published, the `papers` table, describing the papers themselves, and the `paperauths` table which indicates which authors wrote which papers. The "Data Definition Language" (DDL) commands used to create these tables are as follows:

```
CREATE TABLE authors (
    id INTEGER PRIMARY KEY,  -- id of author
    name TEXT  -- name of author
);

CREATE TABLE venues (
    id INTEGER PRIMARY KEY,  -- id of venue
    name TEXT NOT NULL,  -- type of venue
    year INTEGER NOT NULL,  -- year of publication
    school TEXT,  -- school of publication (for theses)
    volume TEXT,  -- volume of publication (for journals)
    number TEXT,  -- number of publication (for journals)
    type INTEGER NOT NULL  -- type of publication (journal, conference, thesis)
```

```
);

CREATE TABLE papers (
    id INTEGER PRIMARY KEY,  -- id of paper
    name TEXT NOT NULL,  -- title of paper
    venue INTEGER,  -- venue in which paper was published
    pages TEXT,  -- page numbers in venue
    url TEXT,  -- url of digital copy of paper
    FOREIGN KEY (venue) REFERENCES venues(id)
);

CREATE TABLE paperauths (  -- mapping from papers to authors
    paperid INTEGER,
    authid INTEGER,
    FOREIGN KEY (authid) REFERENCES authors(id),
    FOREIGN KEY (paperid) REFERENCES papers(id)
);
```

`CREATE TABLE name` defines a new table called `name` in SQL. Within each command is a list of field names and types (e.g., `id INTEGER` indicates that the table contains a field named `id` with type `INTEGER`). Each field definition may also be followed by one or more modifiers, such as:

- `PRIMARY KEY`: Indicates this is a part of the primary key (i.e., is a unique identifier for the row). Implies `NOT NULL`.
- `NOT NULL`: Indicates that this field may not have the special value `NULL`.

In addition, `FOREIGN KEY (field) REFERENCES` indicates that this field is a foreign key which references an attribute (i.e., column) in another table. Values of this field must match (join) with a value in the referenced attribute. Phrases following the "`--`" in the above table definitions are comments.

Notice that the above tables reference each other via several foreign-key relationships. Papers are published in a particular venue, indicated by the `venue` attribute of the `papers` table, which references the `id` attribute of the `venues` table. Each venue contains many papers. Papers are authored by one or more authors, and each author has written one or more papers. Because this is a many-to-many relationship, we need to represent it using the intermediate table `paperauths`. Each row of `paperauths` indicates that a particular author (`authid`) wrote a particular paper (`paperid`).

Figure 1 is a simplified "Entity-Relationship Diagram" that shows the relationships (diamonds) between the tables (squares). This is a popular approach for conceptualizing the schema of a database; we will not study such diagrams in detail in 6.830, but you should be able to read and recognize this type of diagram. Figure 2 shows a simple example database.

3 Running SQLite on the DBLP Dataset

This section describes how to get SQLite database running on the DBLP database. For users unexperienced in Unix-like operating systems, or without their machine with Linux or MacOS, we recommend that you use athena, either by going to one of the clusters, or by ssh'ing into one of the athena machines (e.g., athena.dialup.mit.edu).

Obtaining SQLite: You first need to obtain a copy of the sqlite3 binary, which is the program that reads (and manipulates) SQLite database.

If you are running on a *64 bit Linux* machine (such as athena.dialup.mit.edu, and most of the Athena workstations), we have put a pre-built binary at `http://db.csail.mit.edu/sqlite3.tar.gz` (this binary includes readline support, which makes using the interactive SQL shell much easier).

To download and decompress it, open a command-line and type something like

```
wget http://db.csail.mit.edu/sqlite3.tar.gz
tar -xvzf sqlite3.tar.gz
```

Alternatively, you can download a binary to your local machine by going to `http://www.sqlite.org/download.html` and downloading the "precompiled command-line shell" for your platform.

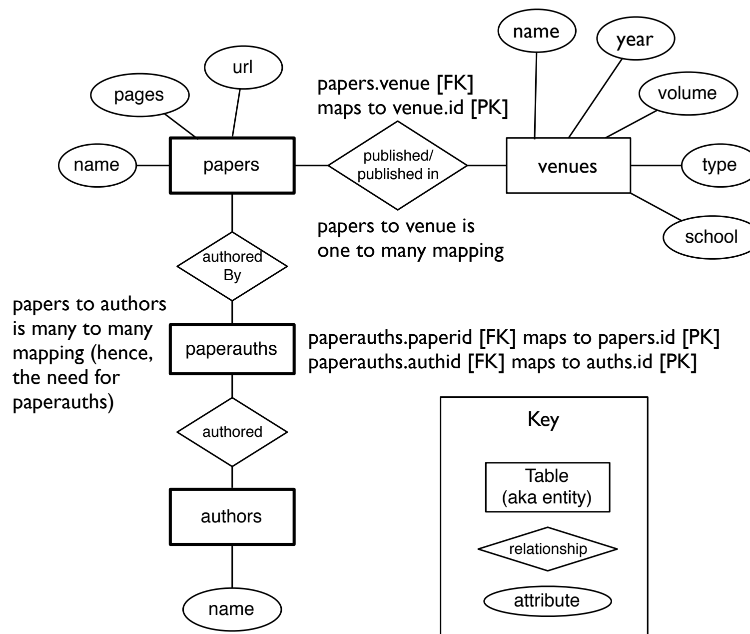

Figure 1: A simplified entity-relationship diagram for the papers database.

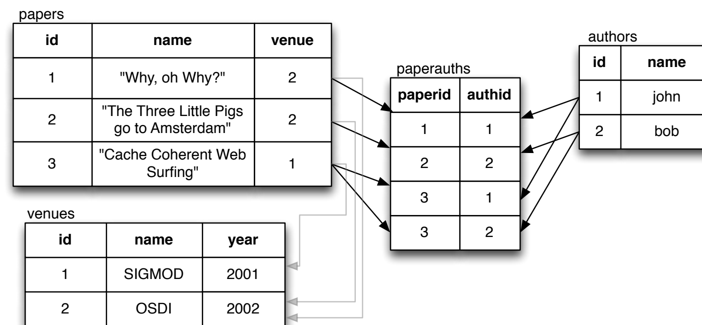

Figure 2: A simple example papers database (note that not all fields of the tables are shown).

Connecting to the database: Assuming you are in the same directory as the sqlite3 binary you just downloaded, and on a machine with afs access (such as any athena machine), you should now be able to type:

```
./sqlite3 /afs/csail.mit.edu/group/db/6830/dblp
```

You should see a prompt that says:

```
sqlite>
```

Note that this database file is quite large (700 MB uncompressed, 275 MB compressed) and you may not have sufficient quota to copy it to your local athena directory. If you would like to download it to your local machine (to run sqlite3 locally on a non-afs equipped machine), we have placed a compressed copy at:

```
http://db.csail.mit.edu/dblp.gz
```

You will need to decompress this using the gunzip command line tool (should be installed by default on Mac and Linux systems). From the command line you can download and decompress by typing something like:

```
wget http://db.csail.mit.edu/dblp.gz
gunzip dblp.gz
```

Then to run SQLite, you would type (assuming sqlite3 is in the current directory):

```
./sqlite3 dblp
```

Please contact us if you have trouble installing or running SQLite.

4 Using the Database

Once connected, you should be able to type SQL queries directly. All queries in SQLite must be terminated with a semi-colon. For example, to get a list of all records in the `papers` table, you would type (note that running this query will take a VERY long time, since it will return over a million answers, so please don't do it – keep reading instead!):

```
SELECT * FROM papers;
```

A less expensive way to sample the contents of a table (which you may find useful) is to use the `LIMIT` SQL extension which specifies a limit to the number of records the database should return as answer to a query. This is not part of the SQL standard, but is supported by many relational DBMSes, including SQLite. For example, to view 20 rows from the `papers` table, use the following query:

```
SELECT * FROM papers LIMIT 20;
```

The `LIMIT` clause above returns only the first 20 rows of the result for the query `SELECT * FROM papers`. However, keep in mind that the relational model does not specify an ordering for rows in a relation. Therefore, there is no guarantee on which 20 rows from the result are returned, unless the query itself includes an `ORDER BY` clause, which sorts results by any output column or an expression on columns used in the query. Nevertheless, `LIMIT` is very useful for playing around with a large database and sampling data from tables in it.

SQLite includes a number of simple commands to help you query the metadata of the database and to understand query performance. You can use the `.help` to see a list. The most useful are `.tables` to a see a list of tables and `.schema tablename` to see the schema of a given table.

Note that the output of SQLite can be a bit hard to read by default. You may find it more readable if you run the following commands from the sqlite prompt:

```
.mode column
.headers on
```

Note that in column mode, you may need to explictly control the width of columns to prevent truncation. If you want the first two columns to each be 50 characters wide, you can write:

```
.width 50 50
```

5 Questions

For each question, please include both the **SQL query** and the **result** in your answer. Some of the more complex queries can take quite a while to run, so be patient!

Q1. Write a query (using the `SELECT` statement) that finds papers with both the string "banana" (in upper or lower case) and the string "spider" (in upper or lower case) in their title. You may find the "`LIKE`" operator as well as the function "`lower`" useful. Show both the query and the result.

Q2. Write a query that finds the names of papers, authors, and venues that have published a paper whose title contains the string "coffee" (in upper or lower case) and with an author whose name contains the string "sam" (in upper or lower case). You will need to write a "join query" between four tables (`papers, authors, paperauths`, and `venues`). Show the query and the result.

Subqueries and nesting: In SQL, a subquery is a query over the results of another query. You can use subqueries in the `SELECT` list, the `FROM` list, or as a part of the `WHERE` clause. For example, suppose we want to find the name of the paper with the smallest id. To find the smallest id, we would write the query `SELECT min(id) FROM papers`, but SQL doesn't provide a way to get the other attributes of that minimum-id paper without using a nested query. We can do this either with nesting in the `FROM` list or in the `WHERE` clause. Using the nesting `FROM` list, we would write:

```
SELECT name
```

```
FROM papers,
     (SELECT min(id) AS minid
      FROM papers) AS nested
WHERE papers.id = nested.minid;
```

Using nesting in the `WHERE` clause, we would write:

```
SELECT name
FROM papers
WHERE papers.id = (SELECT min(id) FROM papers);
```

As we discussed in class, there are usually several possible ways to write a given query, and some of those ways may provide different performance (despite the best efforts of database system designers to build optimizers that yield query performance that is independent of the query's formulation).

Note that if you were interested in finding papers with authors that matched a list of ids, you could replace the "=" sign in the query above with the `IN` keyword; e.g.:

```
SELECT papers.name
FROM papers,paperauths
WHERE papers.id = paperauths.paperid
AND paperauths.authid IN (SELECT id FROM authors WHERE ...)
```

It is usually the case that when confronted with a subquery of this form, it is possible to un-nest the query by rewriting it as a join.

Q3. Show what the `SELECT ...WHERE ...IN` query above would look like when the `IN` portion of the query is "un-nested" by rewriting it as a join. Use "..." in your query to denote the conditions in the `WHERE` predicate of the original query.

Temporary Tables: In addition to allowing you to nest queries, SQL supports saving the results of a query as temporary table. This table can be queried just like a normal table, providing similar functionality to nested queries, with the ability to reuse the results of a query. The command to create a temporary table is:

```
CREATE TEMP TABLE name AS SELECT ...
```

where "..." are the typical `SELECT` arguments. This creates a table called `name`. `TEMP` causes the table to automatically be deleted (or "dropped" in SQL nomenclature) when the session is over, such as when you quit `sqlite3`.

Temporary tables can be useful when interactively developing a SQL query, since you can explore or build on previous results. It is usually possible to use nesting in place of temporary tables and vice versa; nesting will (generally) lead to better performance as query optimizers include special optimizations to "de-nest" queries that cannot be easily applied on temporary tables.

In addition to nesting (or temporary tables), to answer the next few questions, you must learn two concepts: *self-joins* and *aggregates*.

Self joins: A self-join is a join of a table with itself. This is often useful when exploring a transitive relationship. For example, the following query:

```
SELECT pa2.authid
FROM paperauths AS pa1, paperauths AS pa2
WHERE pa1.authid = 5
AND pa1.paperid = pa2.paperid;
```

would return the list of the ids of all co-authors of an author with id 5.

Aggregates: SQL includes many useful aggregate functions like `SUM`, `COUNT` and `AVG` that can compute statistical measures over a dataset. The aggregate functions can be used by themselves, but are more commonly used in conjunction with a `GROUP BY` clause that specifies a grouping of the data by a particular attribute: the aggregate function computes the sum, count, average or other statistic over the members of each group. For example, the query:

```
SELECT year, COUNT(*)
FROM papers,venues
WHERE papers.venue = venues.id
GROUP BY year
```

counts the number of papers written in each year that occurs in the database.

You will need to use a combination of the above features in answering the following queries. For each query, please show the text of your query and the result it produces (which should answer the question posed.)

Q4. Find the names of venues that Professor Hari Balakrishnan has published in that Professor Michael Stonebraker has not.

Q5. Find authors who have written papers in the venue named "SIGMOD Conference" for at least three years in a row, and for each author list the names of the papers they published each year, the year, and the author's name.

Q6. Count the number of pairs of authors who have written exactly n papers together, for all values of n. For example, if the `paperauths` table contained the values shown in Table 1(a), the results are shown in Table 1(b). Be careful: authors cannot co-author a paper with themselves, and the pairs of ids (100, 101) and (101, 100) are identical, so they should only be counted once.

Q7. The author who publishes the most papers in a calendar year is given the title of *beast* (fictional, of course). Find the author who has been a *beast* the most number of times. Also find the years in which he/she won this title, and the number of papers he/she published in each of those years.

Q8. *Grand Slam Winners*: Four major conferences in the area of database research: CIDR, ICDE, SIGMOD and VLDB, also collectively called the Slams, are considered the most prestigious venues for publication in this area. A researcher who publishes in all four conferences in the same calendar year is said to have won the elusive GRAND SLAM.

Find all authors who have managed to achieve this feat not just once, but at least twice and also with the years in which they have done so.

Note that the provided DBLP database does not always record conferences by their well-known names. While CIDR and ICDE go by their respective names, SIGMOD may be either of "SIGMOD Conference" or "SIGMODConference" and VLDB may be either "VLDB" or "PVLDB".

Q9. *The Gray Code:* On a grey day five years ago, Jim Gray, a Turing Award winning database researcher set out on a sailing trip in the Pacific. Several months later he was declared lost at sea and a year later, the University of California Berkeley hosted a tribute to which only researchers with a Gray code of 1-3 were invited.

A researcher has a Gray code of $k + 1$ if k is finite and is the lowest number of any co-author he/she has written papers with. Jim Gray is considered to have a Gray code of zero and all direct collaborators of Gray are assigned Gray code 1.

How many researchers were invited, and how many had Gray codes of 2 and 3?

Table 1: Example `paperauths` data and the resulting co-author counts

(a) `paperauths`

`paperid`	`authid`
4	100
4	101
5	100
5	102
7	100
7	101
7	102

(b) co-author counts

n	number of pairs
1	1
2	2