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` 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 are a bit tricky). You may also wish to refer to Chapter 5 of "Database Management Systems".

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 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 600,000 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 `venue` 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 character varying(200) -- name of author
);

CREATE TABLE venue (
    id integer primary key,  -- id of venue
    name character varying(200) NOT NULL,  -- type of venue
    year integer NOT NULL,  -- year of publication
    school character varying(200), -- school of publication (for theses)
    volume character varying(50),  -- volume of publication (for journals)
    number character varying(50), -- 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 character varying(665) NOT NULL, -- title of paper
    venue integer references venue(id), -- venue in which paper was published
    pages character varying(100), -- page numbers in venue
    url character varying(500) -- url of digital copy of paper
);
```

```
CREATE TABLE paperauths ( -- mapping from papers to authors
    paperid integer references papers(id),
    authid integer references authors(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`.
- `references`: Indicates that this 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 particular a venue, indicated by the `venue` attribute of the `papers` table, which references the `id` attribute of the `venue` 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, detail in 6.830, but you should be able to read and recognize this type of diagram. Figure 2 shows a simple example database.

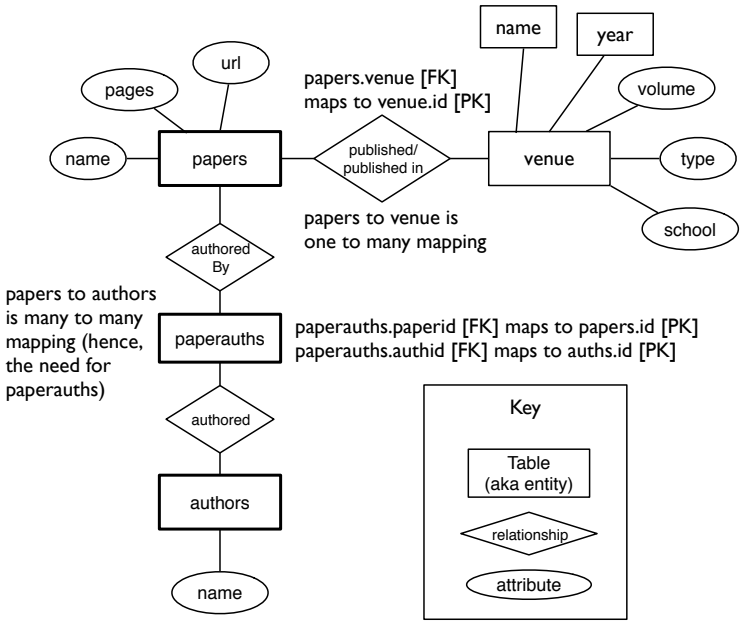

Figure 1: A simplified entity-relationship diagram for the papers database.

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

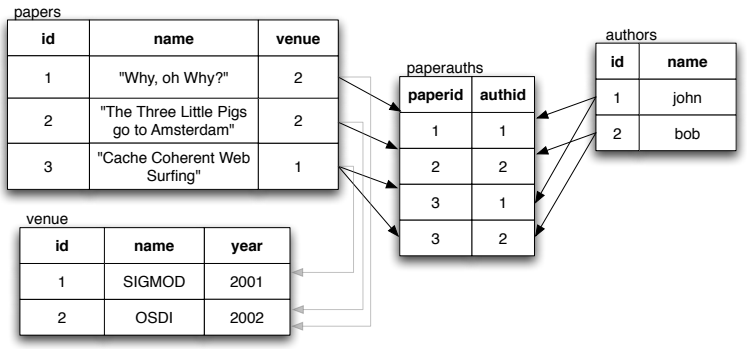

Figure 2: A simple example papers database (note that not all fields of the tables are shown).

`psql` tool available for Linux-based Athena machines at the course server. To use it, download it to your Athena directory, expand the archive (using a command like `tar xvzf 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 6.830 username
```

You MUST do this from an MIT machine (e.g., with IP Address 18.x.x.x). Also, *username* is your athena username that you signed up with on our signup sheet. If you cannot log in, please let us know.

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 `papers` table, you would type (note that running this query will take a VERY long time, since it will return 600,000 answers!):

```
SELECT * FROM papers;
```

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.

You may find the `\dt` command particularly useful; it allows you to see the names of the tables that are available and (when invoked with a particular table name), their schemas.

4 Questions

1. Write a query (using the `SELECT` statement) that will find the complete names of all papers that contain the string "Beer". You may find the "~" operator useful. Show both the query and the result.

2. Write a query that finds the venues in which those papers were published. You will need to write a "join query". Show the query and the result.

   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);
```

Hence, 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 that matched a list of ids, you could replace the "=" sign in the query above with the `IN` keyword; e.g.:

(1)

```
SELECT name
FROM papers
WHERE papers.venue IN (SELECT id  FROM venue 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.

3. Show what the `SELECT ... WHERE ... IN` query labeled (1) above would look like when re-written as a join.
4. Write a query that finds the total number of papers published in each of the conferences identified in Problem 2. Try to avoid actually typing the names of the conferences in your query.

   It is possible to answer this question using either a join or a subquery. Show both versions, and the results of running the query.
5. Write a query that finds the authors of all of the papers whose titles contain the phrase "Beer". Show your query and the results.
6. In the database community, the two most important conferences are SIGMOD and VLDB. Write a query that finds the top 10 most published authors in these two conferences (you should search for the strings 'SIGMOD Conference' and 'VLDB'; don't use the '~' operator – instead, use the '=' operator). Show your query and the top 10 results.
7. The previous query shows people with the highest publication count, but doesn't normalize for the fact that some of these authors are really OLD. Arguably, a more interesting metric would be to divide the number of publications by the number of years since the person published his or her first paper (not in SIGMOD or VLDB, but in any conference.) Write a query that computes the new top 10 list according to this metric, and show your query and your top ten list (as well as the score associated with each person in the list). Note that this is a long query that will take quite a while to run...
8. Write a query that finds people who have published a paper in SIGMOD for each of the past five years (e.g., in SIGMOD 2001 - 2005). See if you can do this without simply combining the results of five different queries (one for each year.) Show your query and the list of names.

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.