CMPSCI 645: Database Design and Implementation

CMPSCI 645 Homework 1: Basic data analysis pipeline

Objectives:
To get familiar with the main components of the Data Analytic pipeline: schema design, data acquisition, data transformation, querying, and visualizing.
Assignment tools:
postgres, any visualization tool (e.g., excel)
Due date:
Monday, March 4, 2015, at 11:59pm. Please use Moodle to turn in your assignment
What to turn in:
These files: pubER.pdf, solution.sql, graph.pdf., and, optionally, extracredit.sql. Your solution.sql file should be executable using the command `psql -f solution.sql`, and should contain comments that explain what you do. This will facilitate grading and partial credit.
Dataset:
dblp.xml, dblp.dtd.
Starter code:
createRawSchema.sql, wrapper.py

In this homework you will analyze publication data. The data is DBLP, the reference citation website created and maintained by Michael Ley. The analysis will be done in postgres. The visualization can be done in excel, or any other tool of your choice.

1. Schema Design

Design and create a database schema about publications. We will refer to this schema as PubSchema, and to the data as PubData.

1. **E/R Diagram.** Design the E/R diagram, consisting of the following entity sets and relationships.
   - `Author` - has the following attributes: id (a key; must be unique), name, and homepage (a URL).
   - `Publication` - has the following attributes: pubid (the key -- an integer), pubkey (an alternative key, text; must be unique), title, and year. It has the following subclasses:
     - `Article` - has the following extra attributes: journal, month, volume, number
     - `Book` - has the following extra attributes: publisher, isbn
     - `Incollection` - has the following extra attributes: booktitle, publisher, isbn
     - `Inproceedings` - has the following extra attributes: booktitle, editor
   - There is a many-many relationship `Authored` from `Author` to `Publication`.

   Draw the E/R diagram for this schema. Identify all keys in all entity sets, and indicate the correct type of all relationships (many-many or many-one); make sure you use the ISA box where needed. Aim to produce a textbook-grade perfect scheme.

   You have to turn in your E/R Diagram in a file named pubER.pdf

2. **Create a Database.** You will have to install postgres and python on your machines, if you don't already have them installed. Start psql, and create a database called `dblp` by running the following command:

   ```
   create database dblp;
   ```

   If for any reason you need to delete the entire database and restart, run:

   ```
   drop database dblp;
   ```

   To to connect to the database that you have created and run queries, type

   ```
   \c dblp
   ```
   then type in your SQL commands. Remember four special commands:

   \c DBName -- connect to the database DBName
   \q -- quit (exit psql)
   \h -- help
   \? -- help for internal commands

3. **Create the Publication Schema in SQL.** We will refer below to this schema as PubSchema.

Write appropriate statements:

```
create table Author ( . . . ); . . .
```

that implement the E/R diagram above, then run them in postgres. You can either write commands to postgres interactively, or you can store your commands in a file and execute it will the `psql -f` option. For you simplicity, you may choose all your data types to be `int` and `text` only. You need to create keys, foreign keys, and unique constraints; you may either do it now, or postpone this for later.

If for any reason you need to delete the tables and restart, type:

```
drop table Author;
```

Drop them in the reverse order you created (otherwise you will violate foreign key constraints), or type `drop table Author cascade;`

You should include all commands in the file solution.sql.

2. Data Acquisition

Retrieve the dataset and load it to your database

1. **Import DBLP into postgres.**
   - You will need to download dblp.dtd and dblp.xml (or dblp.xml.gz) from http://dblp.uni-trier.de/xml/.

     Before you proceed, extract dblp.xml and make sure you understand it. Look inside by typing:

     ```
     less dblp.xml
     ```

     The file looks like this. There is a giant root element:

     ```
     <dblp> . . . . </dblp>
     ```

     Inside there are publication elements:

     ```
     <article> . . . </article>
     <inproceedings> . . . </inproceedings>
     ```
     etc

     Inside each publication element there are fields:

     ```
     <author> . . . </author>
     <title> . . . </title>
     <year> . . . </year>
     ```
     etc

   - Download `wrapper.py`, which is provided as starter code. Edit wrapper.py appropriately to point to the correct location of the `dblp.xml` file on your drive.

     Execute the wrapper:
     ```
     python wrapper.py
     ```

     This step will take several minutes and will produce two large files: `pubFile.txt` and `fieldFile.txt`.

     Before you proceed, make sure you understand what happened during this step. Note that we are using two tools here: python, and a python XML SAX parser, which is a simple, event driven parser. The advantage of a SAX parser (over a DOM parser) is that it can process the XML data in a streaming fashion, without storing it in main memory. For a quick illustration, see Example 1-1 here. A SAX application like wrapper.py needs to be written to process nested elements in a streaming fashion. For each publication element, like `<article>...</article>`, `wrapper.py` writes one line into `pubFile.txt`, and for each field element, like `<year>...</year>`, it writes one line into `fieldFile.txt`. Notice how `startElement` handles differently a publication element from a field element; also notice that most of the useful work (writing to the files) is done by `endElement`.

     Also, look inside pubFile.txt and fieldFile.txt by typing:

     ```
     less pubFile.txt
     less fieldFile.txt
     ```

     These are tab-separated files, ready to be imported in postgres.

- Copy `createRawSchema.sql`, which is provided in the starter code to your Edlab directory, or download it to your computer. Modify it with the path to your `pubFile.txt` and `fieldFile.txt`. Then run:

```
psql -f createRawSchema.sql dblp
```

This imports the data to postgres, and will also take several minutes to complete. It creates two tables, `Pub` and `Field`, which we call the RawSchema, and we call their data the RawData.

Before you proceed, make sure you understand what you did. Inspect `createRawSchema.sql`: you should understand every single bit of this file. Also, start an interactive postgres, and type in some simple queries, like:

```
select * from Pub limit 50;
select * from Field limit 50;
```

Here is one way to play with this data, in its raw format. Go to DBLP and check out one of your favorite papers, then click on the Bibtex icon for that paper. Say, you check out Peter Buneman's DBLP entry, and click on the paper you had to review:

```
@inproceedings{DBLP:conf/pods/BunemanKT02,
  author    = {Peter Buneman and
               Sanjeev Khanna and
               Wang Chiew Tan},
  title     = {On Propagation of Deletions and Annotations Through Views},
  booktitle = {Proceedings of the Twenty-first {ACM} {SIGACT-SIGMOD-SIGART} Symposium
               on Principles of Database Systems, June 3-5, Madison, Wisconsin, {USA}},
  pages     = {150--158},
  year      = {2002},
  crossref  = {DBLP:conf/pods/2002},
  url       = {http://doi.acm.org/10.1145/543613.543633},
  doi       = {10.1145/543613.543633},
  timestamp = {Wed, 23 May 2012 16:53:24 +0200},
  biburl    = {http://dblp.uni-trier.de/rec/bib/conf/pods/BunemanKT02},
  bibsource = {dblp computer science bibliography, http://dblp.org}
}
```

The key of this entry is `conf/pods/BunemanKT02`. Use it in the SQL query below:

```
select * from Pub p, Field f where p.k='conf/pods/BunemanKT02' and f.k='conf/pods
/BunemanKT02';
```

2. Write SQL Queries to answer the following simple questions, using directly the RawSchema:
   - For each type of publication, count the total number of publications of that type. Your query should return a set of (publication-type, count) pairs. For example: (article, 20000), (inproceedings, 30000), ...
   - We say that a field "occurs" in a publication type, if there exists at least one publication of that type having that field. For example, "`publisher` occurs in `incollection`", but "`publisher` does not occur in `inproceedings`" (because no `inproceedings` entry has a publisher). Find the fields that occur in *all* publications types. Your query should return a set of field names: for example it may return the field `title`, if `title` occurs in all publication types (`article`, `inproceedings`, etc. notice that title does not have to occur in every publication instance, only in some instance of every type), but it should not return `publisher` (since the latter does not occur in any publication of type `inproceedings`).
   - Your two queries above may be slow. Speed them up by creating appropriate indexes, using the CREATE INDEX statement. You also need indexes on Pub and Field for the next question; create all indices you need on RawSchema at this point. Please note that creating indexes on such large datasets will take a *loooong* time. The benefit is that once you have the indexes your queries will be much much faster.

Include your queries in the file solution.sql. This consists of SELECT-FROM-WHERE queries and CREATE INDEX statements. In addition, insert into the file all answers to the queries, in form of SQL comments (line starting with two dashes "`--`").

3. Data Transformation

Transform the DBLP data from RawSchema to PubSchema. Your transformation will consist of several SQL queries, one per PubSchema table. For example, to populate your Article table, you will likely run a query like:

```
insert into Article (select ... from Pub, Field ... where ...)
```

Since PubSchema is a well design schema, you will need to go through some trial and error to get the transformation right: use SQL interactively to get a sense of RawData, and find how to map it to

PubData. We give you a few hints:

- You may create temporary tables (and indices) to speedup the data transformation. Remember to drop all your temporary tables when you are done.
- Databases are notoriously inefficient at bulk inserting into a table that contains a foreign key, because they need to check the foreign key constraint after each insert. Hint: do not declare foreign keys in PubSchema; instead, populate the tables first, then run the `ALTER TABLE` command (see `\h ALTER TABLE` in postgres).
- PubSchema requires you to generate an integer key for every author, and for every publication. Use a `sequence`. For example, try this and see what happens:

```
create table R(a text);
insert into R values ('a');
insert into R values ('b');
insert into R values ('c');
create table S(id int, a text);

create sequence q;
insert into S (select nextval('q') as id, a from R);
drop sequence q;

select * from S;
```

- DBLP knows the Homepage of some authors, and you need to store these in the Author table. But where do you get the homepages from the RawData? DBLP uses a hack. Some publications of type www are not publications, but instead represent homepages. For example, here's how you find out Peter Buneman's homepage (this command must run very fast, in 1 second or so; if it doesn't, then you are missing some indexes):

```
select z.* from Pub x, Field y, Field z where x.k=y.k and y.k=z.k and x.p='www' and
y.p='author' and y.v='Peter Buneman';
```

  Get it? Now you know Peter's homepage. However, you are not there yet. Some www entries are not homepages, but are real publications. Try this:

```
select z.* from Pub x, Field y, Field z where x.k=y.k and y.k=z.k and x.p='www' and
y.p='author' and y.v='Dan Suciu';
```

  Your challenge is to find out how to identify each author's correct Homepage. (A small number of authors have two correct, but distinct homepages; you may choose any of them to insert in Author).
- What if a publication in RawData has two titles? Or two publishers? Or two years? (You *will* encounter duplicate fields, but not necessarily these ones.) Your PubSchema is textbook-perfect, and does not allow multiple attributes or other nonsense; if you try inserting, you should get an error at some point. There are only few repeated fields, but they prevent you from uploading PubSchema, so you must address them. It doesn't matter how you resolve these conflicts, but your data should load into PubSchema correctly.
- Once you are done loading PubData, make sure you add all foreign keys and unique constraints that you have omitted for performance reasons. Hint: use `ALTER TABLE`.

Include all your INSERT, CREATE TABLE, ALTER TABLE, ... statements, in the file solution.sql.

4. Queries.

Now, using your new schema (PubSchema), write SQL queries to answer the following questions:

1. Find the top 20 authors with the largest number of publications. Runtime: under 10s.
2. Find the top 20 authors with the largest number of publications in STOC. Repeat this for two more conferences, of your choice (suggestions: top 20 authors in SOSP, or CHI, or SIGMOD, or SIGGRAPH; note that you need to do some digging to find out how DBLP spells the name of your conference). Runtime: under 10s.
3. Two major database conferences are 'PODS' (theory) and 'SIGMOD Conference' (systems). Find (a) all authors who published at least 10 SIGMOD papers but never published a PODS paper, and (b) all authors who published at least 5 PODS papers but never published a SIGMOD paper. Runtime: under 10s.
4. A decade is a sequence of ten consecutive years, e.g. 1982, 1983, ..., 1991. For each decade, compute the total number of publications in DBLP in that decade. Hint: for this and the next query you may want to compute a temporary table with all distinct years. Runtime: under

1minute.

5. For each decade, find the most prolific author in that decade. Hint: you may want to first compute a temporary table, storing for each decade and each author the number of publications of that author in that decade. Runtime: about 1 hour.

6. Find the top 20 most collaborative authors. That is, for each author determine its number of collaborators, then find the top 20. Hint: for this and some question below you may want to compute a temporary table of coauthors. Runtime: a couple of minutes.

7. Find the institutions that have published the most papers in PVLDB; return the top 20 institutions. Then repeat this query with your favorite conference or journal (STOC, or CHI, or ...), and see which are the best places and you didn't know about. Hint: where do you get information about institutions? Use the Homepage information: convert a Homepage like `http://www.cs.umass.edu/~immerman/` to `http://www.cs.umass.edu`, or even to `www.cs.umass.edu`: now you have grouped all authors from our department, and we use this URL as surrogate for the institution. Google for `substring`, `position` and `trim` postgres.

Include your queries in the file solution.sql.

5. Data Visualization.

Compute and draw two histograms: the histogram of the number of collaborators, and the histogram of the number of publications. In each case, start by writing a SQL query that returns a set (k, f(k)), where k=1,2,3,... and f(k) = number of authors that have exactly k collaborators (for the first histogram) and f(k) = number of authors that have k publications (for the second histogram). You can use the copy, or the `\copy` command to output the result into a CSV file, and from there import it into Excel (or any other tool of your choice), and draw the scatter plot.

For each of the two histograms, indicate whether the scatter plot is exponential, or a power law, or something else. You may want to try either a log scale, or a log-log scale in your plot to best determine the type of law. Turn in the plots using the best scale that illustrates your finding (e.g. if you used a log-log scale to determine the type of distribution, then your plot should be in log-log).

Include your two queries in solution.sql, and submit your plots in graph.pdf.

6. Extra credit.

Find out something interesting, or surprising from DBLP. Write the SQL query (or queries), and include it in the extracredit.sql file, together with a brief description of your finding and why it is surprising.

If you decide to do the extra credit, turn in a file called extracredit.sql