CMPSCI 645: Homework 2

Due: Wednesday, February 28 2018, 11:59pm

This assignment will get you familiar with several components of the data analytics pipeline: schema design, data acquisition, data transformation, and querying. You will be using PostgreSQL to complete the assignment, and you will analyze publication data. The data is DBLP, the reference citation website created and maintained by Michael Ley. You will need to submit your queries through Gradescope. The autograder will get you instant feedback, and you will be able to modify your submission until the deadline.

Important note: We reserve the right to evaluate submissions after the deadline using additional tests and data samples. If a submission fails on these additional tests, assignment grades will be adjusted accordingly.

What to turn in: You will need to use Gradescope to submit your assignment. You will need to submit a total of 10 `.sql` files. Please follow the instructions carefully on how to name your files and what to include and what to not include to avoid errors.

Dataset: The dataset is more than 400MB and, depending on your connection, may take some time to download.

dblp.xml dblp.dtd

Important note: DBLP is a "live" dataset, which gets updated continuously, as new articles get published. As a result, the dataset will differ depending on when you download it, so your results may vary somewhat. In addition, new entries to the dataset may cause our scripts to break at any time if the new data contains some unexpected formatting. Please let us know if you encounter such problems and we'll do our best to fix them quickly.

Starter code: We provide you with starter code to create the relational schema for your data, and wrapper code to transform the xml data.

createRawSchema.sql createPubSchema.sql wrapper.py

Page 2 of this document includes instructions and important notes for using the autograder. Pages 3–4 guide you through processing and loading the data into the initial "raw" schema.

Submission instructions

- You will need to submit your work through Gradescope. When you upload your submission, the autograder will check it and will tell you which queries you got right and which you got wrong. You may submit as many times as you like until the deadline. *Your final submission will be the one to be counted; make sure that it includes answers to all queries.*
- When you select upload submission, you can drag and drop all your `.sql` files into the submission window, or upload them all together into a `.zip` file. You don't have to submit all queries each time, but you can check any number of queries that you have ready.
- The autograder may take a couple of minutes to complete its evaluation. If it has been more than a few minutes, try reloading the page, as we've noticed the interface get stuck on occasion.
- **Important note:** You may include comments in PostgresSQL syntax in the files you submit. However, ensure that comments do not contain any queries or semicolon (;) characters.
- **Important note:** You may include multiple SQL statements in a single file to define views and temporary tables that are used towards your query. However, make sure that for all questions that need to produce an output table, only a single query in the corresponding file produces that output, otherwise you may confuse the autograder.
- **Important note:** Submission will remain open for a week past the due date to accommodate grace days and late submissions. DO NOT upload submissions past the deadline if you do not wish to use up grace days (or receive a late submission penalty past the grace days). If you upload past the due date, it will count against your grace days quota. As a reminder, grace days count in 24 hour periods. So, submitting a few minutes after the deadline counts as a full grace day used.

Setup: Install Postgres and Python

You will have to install PostgreSQL on your machine, if you don't already have it installed. Also make sure you have Python installed on your system, because the wrapper we provide that processes the DBLP XML data (more on this in a bit) is written in Python. (Our scripts have been tested with Python 2.7.)

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

Setup: Data acquisition

Typically, this step consists of downloading data, or extracting it with a software tool, or inputting it manually, or all of the above. Then it involves writing and running a wrapper script that reformats the data into some CSV format that we can upload to the database. The starter code provides a Python wrapper for you. (The script has been tested in Python 2.7.)

Import DBLP into Postgres

Step 1: 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
```

Step 2: Download `wrapper.py`, which is provided with the 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: if you don't know python already, this homework is a good excuse to learn it. [brief python tutorial]
- 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 (quick illustration). 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`.

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.

Step 3: Copy `createRawSchema.sql`, which is provided in the starter code, 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 psql, 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 "On Propagation of Deletions and Annotations Through Views":

```
@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';
```

Q1: Create the refined schema

What to submit: You do not need to submit anything for this first part; we give you the PubSchema table declarations as part of the starter code.

After the initial setup, your data is loaded into tables `Pub` and `Field`. This initial "raw" schema is not very intuitive, and it is hard to use effectively. You will create a refined schema to transform the data into. This refined schema, which we call PubSchema, is described below:

- `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 pubid and the following extra attributes: journal, month, volume, number
  - ◇ `Book` — has pubid and the following extra attributes: publisher, isbn
  - ◇ `Incollection` — has pubid and the following extra attributes: booktitle, publisher, isbn
  - ◇ `Inproceedings` — has pubid and the following extra attributes: booktitle, editor
- `Authored` — has id and pubid referring to `Author(id)` and `Publication(pubid)` respectively. There is a many-many relationship `Authored` from `Author` to `Publication`.

Import `createPubSchema.sql`, provided with the starter code to create the PubSchema tables. Please note that the table declarations that we provide are missing some important constraints, such as foreign keys. Do not change these declarations and do not declare these constraints for now. The reason is that the declaration of foreign keys at this point will slow down the insertion of data into these tables. Instead, we will add the foreign key constraints **after** we load the data.

Q2: Queries on RawSchema

What to submit: You need to submit files `Q2a.sql` and `Q2b.sql` that compute the queries described below.

Write SQL Queries to answer the following questions, using directly the RawSchema:

`Q2a.sql` For each type of publication, count the total number of publications of that type. Your query should return two fields: the publication type and the total number of publications of that type. Your query should order the results in decreasing order of the publication count. Example output:

PubType	numOfPubs
article	30000
inproceedings	20000

`Q2b.sql` 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 publication types. Your query should return a list of field names: for example it may return the field `title`, if title occurs in all publication types (note 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 query should order the results alphabetically. Example output:

field
title
year

Your queries on the RawSchema may be slow. You will work more effectively if you create appropriate indexes, using the CREATE INDEX statement. We strongly recommend that you 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. You will also need these indexes to speed up the data transformation in the following question.

Please do not include your index creation statements in the queries you submit. The autograder does not evaluate them and does not need them.

Q3: Data transformation

What to submit: You need to submit a single file `Q3.sql`. This file should include all query transformation statements that populate the tables in `PubSchema`, and all constraint creation statements on the `PubSchema` as described below.

Now we will transform the DBLP data from the `RawSchema` to the `PubSchema`. Currently, your `PubSchema` tables are empty. You need to populate them using 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-designed schema, you will need to go through some trial and error to get the transformation right: use SQL interactively to get a sense of `RawSchema`, and find how to map it to `PubSchema`. We give you a few hints:

- You may create temporary tables (and indices) to speedup the data transformation. You should include all necessary declarations, in the right order, in your submission file. Remember to drop all your temporary tables when you are done with them.
- 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 raw data? 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 Buneman'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 RawSchema 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.

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. For this reason, we did not include the foreign key declarations in `createPubSchema.sql`. **After** you populate the tables of `PubSchema` you will need to add these missing constraints with the `ALTER TABLE` command (see `\h ALTER TABLE` in postgres). Add these declarations at the end of `Q3.sql`.

The autograder will evaluate your data transformation using 4 distinct tests on your produced `PubSchema`. Three of these tests check the result of various queries on `PubSchema`, and the fourth checks that you have implemented the appropriate foreign key constraints.

Q4: Queries on PubSchema

What to submit: You need to submit 7 files: `Q4a.sql`, `Q4b.sql`,...`Q4g.sql`. Each file should produce the answer to the corresponding query as described below.

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

`Q4a.sql` Find the top 17 authors with the largest number of publications. Your query should return the name of each author and the number of publications by that author. Order the results by decreasing number of publications, breaking ties alphabetically using the author names. (Expected runtime < 10 sec.)

`Q4b.sql` Find the top 20 authors with the largest number of publications in STOC. Your query should return the name of each author and the number of publications by that author in STOC. Order the results by decreasing number of publications, breaking ties alphabetically using the author names. (Expected runtime < 10 sec.)

`Q4c.sql` Two major database conferences are 'PODS' (theory) and 'SIGMOD Conference' (systems). Find all authors who published at least 10 SIGMOD papers but never published a PODS paper. For your submission, change your query to return all authors who have published at least 2 SIGMOD papers, but no PODS papers. Make sure your query returns the names of the authors, and the number of SIGMOD papers each author published. (Expected runtime < 10 sec.)

`Q4d.sql` A decade is a sequence of ten consecutive years, e.g., 1980–1989, 1981–1990, 1982–1991, etc. For each decade, compute the total number of publications in DBLP in that decade. Hint: you may want to compute a temporary table with all distinct years. The output of your query should have 2 columns: the decade depicted as a range (with the start year, followed by a dash, and then the end year), and the total number of publications for that decade. The output should be ordered based on the decades and should include a row for each decade, starting from the earliest publication year in the data, up to the latest publication year in the data. E.g., if the latest publication year is 2017, the last row of your result should be the decade 2017–2026. An example of how your output should be is shown below. (Expected runtime < 1 min.) Example output:

Decade	numOfPubs
1940-1949	15
1941-1950	13
1942-1951	16
⋮	
2017-2026	27

`Q4e.sql` Find the top 20 most collaborative authors. That is, for each author determine the number of his/her collaborators, then find the top 20. Hint: for this and some later question you may want to compute a temporary table of coauthors. Note: even if author A and author B have worked on multiple publications together, they only count once as each other's collaborator. Your query should return the names of the authors and the number of their collaborators. The result should be ordered in decreasing order of the number of collaborators, and ties should be broken alphabetically by author names. (Expected runtime < 3 min.)

`Q4f.sql` For each year, find the most prolific author in that year. Hint: you may want to first compute a temporary table, storing for each year and each author the number of publications of that author in that year. Your query should return the year, the name of the author who is the most prolific in that year, and the number of publications of the author in that year. The results should be given in the order of the years, and then alphabetically based on the author names in case of ties. (Expected runtime < 10 min.) Example output:

year	author	numOfPubs
1940	John Doe	5
1941	Jane Smith	3
1941	Joe Shmo	3
1942	Bob Loblaw	6

`Q4g.sql` Find the institutions that have published the most papers in PVLDB; return the top 20 institutions. Hint: To get information about institutions you can use homepages; convert a homepage such as `http://www.cs.umass.edu/~ immerman/` to `www.cs.umass.edu`. This way, you can group all authors from our college, and use this URL as surrogate for the institution. Don't worry about reconciling URLs such as `www.cs.umass.edu` and `www.umass.edu`; you can consider those as separate institutions. Google for `substring`, `position`, and `trim` Postgres. Your query should return the URL of the institution and the number of papers published by that institution in PVLDB. Order the results in decreasing number of publications. Make sure the URL you return does not contain `http://` or `https://` in the beginning, nor slashes (`/`) at the end. (Expected runtime < 1 min.) Example output:

Institution	numOfPubs
www.harvard.edu	10
www.mit.edu	9
www.umass.edu	9