PS 1 - The Data Analytics Pipeline

Database Systems, CS-UH 2214

Assigned:	Sept 4, 2025	Due:	Sept 18, 2025

1 Preliminaries

For this problem set[1], you will work individually. You can work through it with your peers but each student needs to submit their own solutions.

You will work with a publicly available dataset on research publications, which is actively maintained and updated by Michael Ley.

The DBLP data set can be found and searched here: `http://dblp.uni-trier.de`. The full dataset can be downloaded for offline analysis here: `http://dblp.uni-trier.de/xml/`. As this file is a bit large, you should download the .xml and the .dtd files from the DBLP directly.

A quick documentation of this dataset is here `http://dblp.uni-trier.de/xml/docu/dblpxml.pdf`. You will download the xml, prepare the dataset, load it into a postgres database and later visualize it using any tool of your choice including Excel, python, D3, etc.

1.1 Submission

You will package your solution into a folder with the title: 'PS1-netID1'. **Zip this folder** and submit via DropBox at the following link:

`https://bit.ly/db-fall2025-ps1`

We will not grade any submission that does not strictly follow the submission rules.

[1]This problem set is based of a problem set from CSE 544 - Principles of Database Systems taught @ University of Washington.

1.2 Install Postgres

On Mac OS X, if you have never installed a postgres instance before, I recommend installing postgres from here. Use the latest version 17: http://postgresapp.com/ or using brew

```
brew install postgresql
```

You will need to set your paths as well, see instructions here: http://postgresapp.com/documentation/cli-tools.html to be able to run the command line tools from Terminal. After brew terminates it will show you a set of instructions to set up your paths, follow those instructions.

You will also benefit from installing pgAdmin4 but for later assignments. Go here: https://www.pgadmin.org/download/ or use brew.

```
brew install --cask pgadmin4
```

Finally, if you don't have a Mac, you can buy one and follow the steps above ;) OR You can install postgres from here, http://www.postgresql.org/download/, pick your OS and install the pre-built binary. I do not recommend compiling from source.

1.3 Check your installation works

Create a Database

In your shell (i.e. not inside the psql shell):

```
createdb dblp
```

If for any reason, you want to start fresh, just drop the whole database:

```
dropdb dblp
```

To open the postgres shell at the created dblp database:

```
psql dblp
```

You might need to provide other information like the host (-h localhost), a username or a password if you created ones.

In psql shell, you can directly type SQL commands:

```sql
create table foo(bar numeric);
select * from foo;
```

Other psql commands can help you such as:

```
\q -- quit (exit psql)
\h -- help
\d -- list relations
\? -- help for internal commands
```

2 Database Design

2.1 Entity-Relationship Diagram

We briefly touch on ER design in class focusing mostly on database normalization. ***Thus, you need to read chapter 2 of the textbook to answer this question.***
Design an ER diagram, consisting of the following entity sets and relationships **[5 points]**.

1. Author - has the following attributes: id (a key; must be unique), name, and homepage (a URL).
2. 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:
   (a) Article - has the following extra attributes: journal, month, volume, number
   (b) Book - has the following extra attributes: publisher, isbn
   (c) Incollection - has the following extra attributes: booktitle, publisher, isbn
   (d) Inproceedings - has the following extra attributes: booktitle, editor
3. There is a many-many relationship Authored from Author to Publication.

Draw the ER 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.

Turn in: In your submission folder include the solution in a file named **ER.pdf**

2.2 Create the database relations with SQL

Construct the PubSchema with SQL statements such as **[5 points]**:

```
create table author ( ... );
...
```

These statements implement the ER diagram. Note:

- You can chose to use only `int` or `numeric` or `text` for data types.
- Create keys, foreign keys, constraints, etc. Include these here (and in the deliverable) but drop them prior to loading your data and then re-insert them.
- Do not use the inherit/pivot functionality available with postgres (implement subclasses as separate tables instead)
- Store your sql command in a SQL file: `createPubSchema.sql`. Execute all the commands at one go:

  ```
  psql -f createPubSchema.sql dblp
  ```

- You can drop any table as follows:

```
drop table author; -- or drop table author cascade;
                   -- to handle foreign key constraint violations;
```

Turn in: In your submission folder include **createPubSchema.sql**

2.3 Data Acquisition & Loading

You need to download dblp.dtd and dblp.xml from http://dblp.uni-trier.de/xml/.
Make sure you understand dblp.xml. Look inside by typing:

```
more dblp.xml
```

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

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

Inside there are publication elements:

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

Inside each publication element there are fields:

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

Run wrapper.py, which is provided in the starter code. (You may have to first edit wrapper.py appropriately to point to the correct location of dblp.xml file, and of the output files, pubFile.txt and fieldFile.txt.)

```
python wrapper.py
```

This step takes several minutes (≈ 7-10 minutes) and produces two large files: pubFile.txt and fieldFile.txt.
Also, look inside pubFile.txt and fieldFile.txt by typing:

```
more pubFile.txt
more fieldFile.txt
```

These are tab-separated files, ready to be imported in postgres.
Run createRawSchema.sql, which is provided in the starter code. First, modify it with the **absolute** file path to pubFile.txt and fieldFile.txt.

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

This imports the data to postgres (≈ 2-5 minutes). 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 by typing this command:

```
psql dblp
```

and type in some simple queries, like:

```sql
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 Christos H. Papadimitriou's DBLP entry, and click on your favorite paper:

```
@inproceedings{DBLP:conf/stacs/GemiciKMPP19,
  author    = {Kurtulus Gemici and
               Elias Koutsoupias and
               Barnab{\'{e}} Monnot and
               Christos H. Papadimitriou and
               Georgios Piliouras},
  title     = {Wealth Inequality and the Price of Anarchy},
  booktitle = {36th International Symposium on Theoretical Aspects of Computer Science,
               {STACS} 2019, March 13-16, 2019, Berlin, Germany},
  pages     = {31:1--31:16},
  year      = {2019},
  crossref  = {DBLP:conf/stacs/2019},
  url       = {https://doi.org/10.4230/LIPIcs.STACS.2019.31},
  doi       = {10.4230/LIPIcs.STACS.2019.31},
  timestamp = {Thu, 02 May 2019 17:40:17 +0200},
  biburl    = {https://dblp.org/rec/bib/conf/stacs/GemiciKMPP19},
  bibsource = {dblp computer science bibliography, https://dblp.org}
}
```

The key of this entry is conf/stacs/GemiciKMPP19. Use it in the SQL query below:

```sql
select *
from Pub p, Field f
where p.k='conf/stacs/GemiciKMPP19'
and f.k='conf/stacs/GemiciKMPP19';
```

2.4 Queries on raw data

To see how long a query takes, in psql enable timing by typing
`timing`.

Write SQL Queries to answer the following questions using Pub p and Field f:

1. 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), ... (not the real answer) **[2 points]**.

2. 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 field). Find the fields that occur in **all** publications types. Your query should return a set of field names: for example it may return 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) **[2 points]**.

3. 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 **[2 points]**.

Turn in: In your submission folder include **solution-raw.sql**, which would include SELECT-FROM-WHERE queries and CREATE INDEX statements. In addition, insert into the file all answers to the queries, in form of SQL comments.

3 Data Transformation

Transform the DBLP data from RawSchema to PubSchema **[5 points]**.

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 designed schema (you designed it yourself!), 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. Here are a few tips:

1. You may create temporary tables (and indices) to speedup the data transformation. Remember to drop all your temp tables when you are done. Keep track of these as well as the drop statements in solution.sql

2. 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). Way faster...

3. 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;
```

4. 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 Azza's official name in DBLP is 'Azza Abouzeid'. Here's how you find out her homepage (P.S. it's outdated :( ):

   ```
   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='Azza Abouzied';
   ```

   Now you know Azza'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='Tim Berners-Lee';
   ```

   Your challenge is to find out how to identify each author's homepage and make sure that is as accurate as possible given the data. (A small number of authors have multiple, but distinct homepages; you may choose any of them to insert in Author).

5. 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, 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.

6. 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.

Turn in: In your submission folder include **transform.sql**, which should include INSERT, CREATE TABLE, DROP TABLE, ALTER TABLE and CREATE INDEX statements.

4 Data Analysis

1. Find the top 20 authors with the largest number of publications. Runtime: under 15s **[2 points]**.

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, UIST, or SIGMOD, VLDB, or SIGGRAPH; note that you need to do some digging to find out how DBLP spells the name of your conference). Runtime: under 5s **[2 points]**.

3. Two of the major database conferences are 'PODS' (theory) and 'VLDB' (systems). Find (a) all authors who published at least 10 VLDB papers but never published a PODS paper, and (b) all authors who published at least 5 PODS papers but never published a VLDB paper. Runtime: under 10s **[2 points]**.

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 10 s **[2 points]**.

5. 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 **[2 points]**.

6. 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: a minute or so **[2 points]**.

7. For each decade, find the second most prolific author in that decade. Runtime: a minute or so **[2 points]**.

Turn in: In your submission folder include **solution-analysis.sql**, which should include your queries and answers as comments.

5 Nearest Neighbor Search

For this part, you will download the abstracts of a sample of arXiv papers to allow you to find papers similar to other papers within the database.

1. Using only the data in the dblp database, create a table of arXiv articles from July 2025 with one additional column for the abstract (empty for now) and an arXiv id column. Hint: ArXiv ids are of this format 2507.00379 and you can lookup up an ArXiv article as follows https://arxiv.org/abs/2507.00379 **[2 points]**.

2. Now populate the abstract field in this table using an external script that connects to your database and updates your abstracts table with abstracts. **[3 points]**

   You can get the abstract of multiple arXiv papers using a call similar to this

   ```
   https://export.arxiv.org/api/query?
   search_query=id:2507.00379+OR+id:2507.07389&max_results=2
   ```

   ArXiv allows you at most one call a second and no more than 3000 calls a day. So make sure your script rate limits accordingly and batches several abstract requests per call no more than 50.

3. Alter your table to now include a vector embedding of each abstract. **[3 points]**

   You would need to install the pg vector extension

```
brew install pgvector
brew services restart postgresql@17
```

   Log back into your database and create the extension

```
CREATE EXTENSION IF NOT EXISTS vector;
SELECT '[0.1, 0.2, 0.3]'::vector;  -- should return a vector
```

   You may need an external script to convert each abstract into an embedding. You can use a transformer from hugging face (e.g. all-MiniLM-L6-v2). You would need to know the length of the vector the transformer creates to set up your embedding column accordingly. For hugging face APIs, you need a token as well.

4. For each paper, find its nearest neighbor, that is the paper most similar to it (No external scripts allowed.)**[2 points]**.

   Hint, two papers are similar to each other if the distance between their embeddings is small. The distance between two embedding can be found using the pgvector extension in postgres as follows:

```
-- the distance between these two vectors is 0
select '[0.1, 0.2, 0.3]'::vector <=> '[0.1, 0.2, 0.3]'::vector as dist;

-- these two vectors are further apart
select '[1, 1, 1]'::vector <=> '[0.8, 0.9, 0.1]'::vector as dist;
```

5. For any set of keywords (e.g. 'vector database systems') find the top-5 most similar papers to it by converting the keywords to an embedding first. No external scripts allowed. Hint, install the pgsl-http connection to make a call to your transformer using postgres functions. **[3 points]**.

```
git clone https://github.com/pramsey/pgsql-http.git
cd pgsql-http
make clean
make PG_CONFIG="$PG_CONFIG" CURL_CONFIG="$(brew --prefix curl)/bin/curl-config"
make install
```

```
create extension http;

-- here is how you can make http calls from within
-- the database.
select status, left(content, 80)
from http_get('https://bbc.com');
```

6. You want your search results to have some diversity, so for an input set of keywords, find the closest paper and then two papers that are far away from each other and the closest paper in the embedding space but all within a distance of < 0.6 to the search keywords. (This query is often called max-min diversification and finding a collection of things that satisfy a global constraint is called a package query. This is an area of active research in our lab!) **[3 points]**

Turn in: In your submission folder include **search.sql**, a **getAbstracts script**, and a **getEmbeddings script** which should include your queries, scripts and answers as comments.