Middle East Technical University | Department of Computer Engineering

CENG 352

Database Management Systems

Spring 2019-2020
Project 2 : Data Cleaning, Tuning, Triggers

Due date: March 22, 2020, Sunday, 23:59

1 Objectives

In this project, you are asked to perform some of the basic DBMS administration tasks:

- *Data transformation*: given a raw XML data, create relational database from it by finding distinct entities.
- *Database tuning*: given a database and a workload, tune the database to make the workload faster.
- *Making the database active*: add triggers to a database that automatically reacts to actions.

The data is from **DBLP**, the database for computer science bibliography, which contains information about millions of publications. It is semi-structured data and it is in XML format. The information of a publication consists of fields such as *author*, *title*, *year* etc. It is a dynamic database and for consistency, you will work with a snapshot of that database, **which can be downloaded from here**. Before starting the project, look inside the given file. Make sure you understand it.

2 Creating the database

The database name will be **"ceng352_mp2_data"** for this mini project. You will have 3 steps while creating the database:

- Read raw XML file and insert values to 1st-level tables.
- Create 2nd-level tables
- Populate 2nd-level tables by running queries on 1st-level tables.

2.1 General information about the data

- Every publication has **pub_key** field, in other words, pub_key is non-empty value for all publications. pub_key values are strings which defines the publication.
  e.g. *journals/4or/AardalHKMS03*.
- Other fields may not exist for all publications, see the XML file to see what this means.
- One publication can have 1 or more authors.
- One book may have many isbn numbers

2.2 Reading raw data from XML file

The given **"dblp.xml"** file must be parsed to extract the publications data. The parser is given to you and you don't need to change any lines of that. When you see the data, you will probably think that the data needs more cleaning, however, that is out of the project's scope.

The parser is named **"parser.py"** and you will run it with:

```
>_ python parser.py
```

This parser searches for "dblp.xml" file in current working directory and creates 2 files named **"pubData.csv"** and **"fieldData.csv"** in the current working directory. In the next step, you will create 2 tables in postres and fill them with these data.

Start **psql** or open up **DBeaver** tool and create a database called **"ceng352_mp2_data"**. If for any reason you need to delete the entire database and restart, use the following statement:

```
DROP DATABASE ceng352_mp2_data;
```

2.3 Inserting values to 1st-level tables

First-level tables are the following tables:

```
Pub (pub_key text,  pub_type text)
Field (pub_key text, field_index int, field_name text, field_value text)
```

The script **"createFirstLevelTables.sql"** includes commands to create these tables. You will do insertions from "pubData.csv" to Pub table and "fieldData.csv" to Field table. Insertion commands are given to you in "createFirstLevelTables.sql" as well. You can either import it just like in the Mini Project 1 or you can run it directly with **psql** command from terminal, however you need to create a user in the database.

```
CREATE USER <username> WITH PASSWORD <password>;
GRANT ALL PRIVILEGES ON DATABASE ceng352_mp2_data to <username>;
```

After user creation, you can run insert code like below:

```
>_ psql -f createFirstLevelTables.sql ceng352_mp2_data
```

After some time, the values will be imported and you can write your queries on them. Before you proceed, check the data loaded by typing some simple queries like:

```
SELECT * FROM Pub p LIMIT 10;
SELECT * FROM Field F LIMIT 50;
```

3 Tasks

3.1 Task 1 - First Level Table Queries (15 pts)

After 1st level tables are created, write SQL queries to answer the following questions:

1. 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. Write your query in a file called **"task_1_1.sql"**.

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 *book*, 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. Write your query in a file called **"task_1_2.sql"**.

3. You queries may be slow. To work more efficiently **create appropriate indexes** using CREATE INDEX statements. Run the same queries after creating indexes to understand the importance of having indexes in the database. Write your index creation statements in a file called **"task_1_indexes.sql"**.

3.2 Task 2 - Data Transformation (15 pts)

The script **"createSecondLevelTables.sql"** includes commands to create 2nd-level tables. You need to create 2nd-level tables by using "createSecondLevelTables.sql". Second level tables are:

```
Author (author_id serial primary key, name text)
Publication (pub_id serial primary key, pub_key text, title text, year int)
Article (pub_id int primary key, journal text, month text, volume text, number text)
Book (pub_id int primary key, publisher text, isbn text)
Incollection (pub_id int primary key, book_title text, publisher text, isbn text)
Inproceedings (pub_id int primary key, book_title text, editor text)
Authored (author_id int, pub_id int)
```

This time, you are responsible for filling these tables with values. You will write **INSERT INTO** statements to find distinct entities from Pub and Field tables and insert them to 2nd-level tables. Pay attention to the following:

- Note that the table definitions are missing some important constraints such as foreign keys. **At this point, do not create any indexes.** This is because constraints and indexes will slow down the insertion of data into these tables.
- **author_id** and **pub_id** are automatically generated values after each insert operation. You will not query them directly. The reason we have them is simply to do integer comparisons rather than string comparisons while writing queries in Task 3.
- As stated in **section 2.1**, one field that exists for a publication may not exist for another publication, and the only guaranteed non-empty field is **pub_key**. Therefore, it is better to use **LEFT JOIN**s on pub_keys (or pub_ids) while inserting distinct entities.
- As stated in **section 2.1**, one book may have many isbn numbers in the Field table. **However, you need to consider only the one that has the max isbn value. Using max() is enough, it does the proper string comparison.**

At the end, you should have:

- 999.999 publications
- 940.577 authors
- 968.221 articles
- 108 books
- 27.210 incollections
- 4.320 inproceedings
- 2.892.67[67]uthor-pub id relationships

Write your **INSERT INTO** statements inside **"task_2.sql"**.

3.3 Task 3 - Second Level Table Queries (60 pts)

While answering the following questions, if you think you need to create temporary tables for the question, include those **CREATE TEMPORARY TABLE** statements inside **"task_3_temp.sql"**. **You have to explain why do you need to create the temporary table for related question. Write your explanations as comments. These explanations don't need to be long or too-detailed. One well-written sentence is enough.**

You still have tables without indexes or constraints. Now it is time to add appropriate indexes and start answering questions below. Include all index-related **ALTER TABLE** statements inside **"task_3_indexes.sql"**.

1. Find authors who have more than 150, less than 200 publications ([150, 200)). Your query should return the name of each author and the number of publications by that author. Order the results by increasing number of publications, increasing author names when publication counts are equal. Write your query in a file called **"task_3_1.sql"**. You can also inspect query results in mini project files to understand the problem better.

2. Find the top 50 authors with the largest number of publications in IEEE-related journals. Your query should return the name of each author and the number of publications by that author in IEEE-related journal. Order the results by decreasing number of publications, increasing author names when publication counts are equal. For finding relatedness, **use LIKE on journal field**. Write your query in a file called **"task_3_2.sql"**. You can also inspect query results in mini project files to understand the problem better.

3. Find authors who has publications in **'IEEE Trans. Wireless Communications'** journal more than 10 times ($[10, \infty)$) but has no publications for **'IEEE Wireless Commun. Letters'** journal. Order the results by decreasing number of publications, increasing author names when publication counts are equal. Write your query in a file called **"task_3_3.sql"**. You can also inspect query results in mini project files to understand the problem better.

4. A decade is a sequence of ten consecutive years, e.g., 1980-1989, 1981-1990, 1982-1991, etc. For each decade **starting from 1940**, compute the total number of publications in DBLP in that decade. The output of your query should have 2 columns: the decade string which is a concatenation of 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 1940, up to the latest publication year in the data. E.g., if the latest publication year is 2020, the last row of your result should be the decade 2020-2030. If the decade is 1940-1950, years will be 1940, 1941, ..., 1949; 1950 shouldn't be considered in that decade. Write your query in a file called **"task_3_4.sql"**. You can also inspect query results in mini project files to understand the problem better.

5. Find the top 1000 most collaborative authors. That is, for each author determine the number of his/her collaborators, then find the top 1000. 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 increasing author names when number of collaborators are equal. Write your query in a file called **"task_3_5.sql"**. You can also inspect query results in mini project files to understand the problem better.

6. **For each year from 1940 to 1990 ([1940, 1990])**, find the most **productive author** in that year. Your query should return the year, the name of the author who is the most productive in that year, and the number of publications of the author in that year. There may be more than one author who published most number of publications in that year, so they can share the place #1. The results should be given in the increasing order of the years, and increasing author names when number of publications are equal. Write your query in a file called **"task_3_6.sql"**. You can also inspect query results in mini project files to understand the problem better.

3.4 Task 4 - Triggers (10 pts)

Create a table called **ActiveAuthors** with only field "name". Insert distinct names of authors who has publications in the last 2 years.

Now write a trigger on Authored table. After each insert operation, add author name to the ActiveAuthors table if the authored publication's year is [2018, 2020].

You need to write your trigger creation statement inside a file called **"task_4.sql"**.

4 Regulations

1. **Submission:** Submission will be done via ODTUClass. Please remember that **late submission is allowed 5 days for ALL programming projects.** You can distribute these 5 days to any mini-project your want. You should put the answer query to each question in a separate .sql file and zip those .sql files with following name:

   ```
   e1234567_project2.zip
       -> task_1_1.sql
       -> task_1_2.sql
       -> task_1_indexes.sql (if exists)
       -> task_2.sql
       -> task_3_1.sql
       -> task_3_2.sql
       ...
       -> task_3_indexes.sql (if exists)
       -> task_3_temp.sql (if exists)
       -> task_4.sql
   ```

   Where you should replace "1234567" with your own student number.

2. **SQL Style:** You should write your queries in readable manner. Write each clause on a separate line, add indentation for clear look. You can use online-formatters/beautifiers for better indentation if you want.

3. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible clarifications on a daily basis.