University of Toronto
CSC343, Fall 2021

Project Phase 2: Schema Implementation, Data Cleaning, and Import

Due: extended to Friday, 19 November, by 8:00 pm

Schema implementation

After reviewing any feedback from your TA and thinking about what we've been learning since you handed in Phase 1, decide on any changes you want to make to your schema. Then implement your (possibly revised) schema in SQL. Follow these guidelines:

- Define a primary key for every table.
- Wherever data in one table references another table, define a foreign key in SQL.
- Consider a NOT NULL constraint for every attribute in every table. It doesn't always make sense, but it usually does — especially if your design is good.
- If you have a column whose values must come from a specific list, consider making a user-defined type as you saw in the University database. If the list of options is long, this gets unwieldy. In that case, consider defining the options in a table instead.
- Express any other constraints that make sense for your domain.
- Every table must have a comment describing what it means for something to be in the table.

To facilitate repeated importing of the schema as you correct and revise it, begin your DDL file with our standard three lines:

```
drop schema if exists projectschema cascade; -- You can choose a different schema name.
create schema projectschema;
set search_path to projectschema;
```

Be sure that you can import your schema without errors.

Record your design decisions

Address any feedback that you got from TAs in phase 1: Summarize the feedback in your own words, and explain any changes that you chose to make as a result. If you chose not to follow advice from your TA, you must explain why.

If you decided on some changes on your accord, explain these also.

If you changed nothing, explain why. Saying that the schema was already good is not sufficient; you must explain how you know it is good.

Data cleaning and import

In this step, you will create the SQL statements necessary to import your data.

You have learned how to insert a row into a table using an INSERT INTO statement such as this:

```
INSERT INTO Student VALUES (00157, 'Leilani', 'Lakemeyer', 'UTM', 'lani@cs', 3.42);
```

You could populate an entire database with a long series of these statements, however there is an overhead cost associated with executing a SQL statement, and you will incur that cost for every individual INSERT INTO. A more efficient approach is to use the postrgreSQL command `\COPY`. It lets you load all the rows of a table in one statement, so you incur the overhead cost only once, for the whole table. This is not only faster than INSERT INTO, it is also more convenient. You probably already have your data in a csv or formatted text file, and `\COPY` lets you load that data directly (rather than having to covert the data into a series of INSERT INTO statements). For instance, if you had data in a comma-separated csv file called `data.csv`, you might say:

```
\COPY Student from data.csv with csv
```

Similarly, a foreign key constraint can be checked "in bulk" more efficiently than row-by-row. So it might be useful to drop foreign key constraints, load data, and re-create the constraints. What's more, when you load data into a table with existing foreign key constraints, each new row requires an entry in the server's list of pending trigger events (since it is the firing of a trigger that checks the row's foreign key constraint). Loading many rows can cause the trigger event queue to overflow available memory, leading to intolerable swapping or even outright failure of the command. Therefore it may be necessary, not just desirable, to drop and re-apply foreign keys when loading large amounts of data.

More about cleaning the data

Please re-read the section "Resources for data cleaning" from the phase 1 handout for some tips.

As you do the importing, you may find the data doesn't perfectly follow its specifications (or your guess as to what its specifications would be if someone had written that down). As a result, it may sometimes violate constraints that you have expressed. You will have to make decisions about how to handle this. For example, if a foreign key constraint is violated, you could remove the constraint so that SQL won't complain, keep the valid references where available, and replace the invalid references with NULL. If a NOT NULL constraint is violated, you might remove it. Or in either of these cases, you might decide to remove any rows that would violate. Of course this affects that answers you will get to some queries and introduces questions about the validity of any conclusions you make. But that's okay. This is a database project, not a research project. The point is to learn about database design and implementation rather than to come to highly accurate conclusions about your domain.

One way to find the constraint violations is to define all the constraints, import the data, and watch the errors fly by. But an early error can influence subsequent errors, making the process laborious. An alternative is to omit some or all of the constraints from the schema at first, import the data, and then run queries to find data that would violate if the constraint were present. Once you have resolved all the issues, you can clear out the database, import the full schema with constraints, and then import the cleaned up data.

If the data is really huge, you may need to cut it down in order not to overload our database server. Be aware that this may violate some constraints, for example, if you remove rows that are referred to from another table. See above for how to deal with violated constraints.

Record your cleaning process

Keep a written record of the steps you take to clean and import your data, as well as the decisions you make along the way (and why). This should be detailed enough that someone else with the same data could follow your steps and end up with exactly the same database.

Hand in the following:

- A file called `schema.ddl` containing the schema expressed in the SQL Data Definition Language.
- The data itself: Pick one data file that is representative of the format your data is in, paste a few lines of that data into another file, and hand that it. Name it whatever is appropriate, but include "data" somewhere in the filename (eg sample-data.csv, or country-data.json).

- A file called `demo.txt` containing an example interaction with the postgreSQL shell where you (a) load the schema and data successfully, (b) run `\d` for each table and (c) run a SELECT count(*) query on each table to show the number of rows in it, and (d) run a SELECT * query on each table to show a sample of its content. Use a WHERE clause or other tactic to cut down to just a handful of rows. Make all of this very tidy so it's easy for the TA to review quickly.
- A file called `phase2.pdf` with a short report containing the following sections: Design Decisions, and Cleaning Process. (See above for what to put in each section.) Your report should be roughly 2 pages long.