Inf553 Project

Damian Bursztyn, Ioana Manolescu, Michaël Thomazo

October 18, 2016

1 Overview

We provide a set of files here:

`https://drive.google.com/file/d/0B4t1oOFzubodRmZhV1NoOG1DWDQ/view`

Each file comprises the tuples of one relation; the relations, together, model the data from the Internet movie database (IMDB, in short). The smallest has **4** tuples; the largest has **more than 5 million** tuples. The data files have a total size of **a few Gigabytes**. You will have to:

1. Load each separate file into a table in Postgres; you will have to create the tables with the appropriate key and foreign key constraints. Loading the data is not trivial and may take some time, both to get it right, and then to run (actually load data into the server). For this, you will have to provide the commands that you used.

2. Write queries for specific tasks. For this, you will have to provide: the queries; their results; some query plans; and suggestions to improve performance.

2 Data description

The schema design mostly (if not fully) follows the design principles seen in course; *do not change the schema*, rather work with it as it is. Similarly, you may find small errors or inconsistencies in the data; just use it as it is.

Each table has an **id** attribute which is a primary key for that table. An attribute of table **table1** whose name is of the form **table2_id** is a foreign key, referencing **table2.id**.

The core tables are **movie** (the movies, but also TV series and other media products) and **name** (people involved in the movies). A detailed description of all tables follows.

movie is the main table representing movies. It has the title, production year, episode number etc.

movie_type comprises 7 categories of movies such as: movie, TV series, video game etc. Each movie is categorized in one of these types (title has a foreign key referencing kind_type.id).

movie_info has information about the movies, structured as follows: each tuple in movie_info is one piece of information about one movie. Thus, movie_info has a foreign key on movie.id to identify the movie. The information comprised in a movie_info tuple is in the info attribute. To enable interpreting this information (understanding what it is about), each movie_info tuple has another foreign key into info_type (see below).

info_type comprises about 100 different categories of information which may be attached *to a movie or to a person*. For instance, one info_type is *plot*: information items of this type are attached to movies. Another info_type is *birth date*: information items of this type are attached to people.

aka_title is a table of alternative titles (some movies are known under more than one title). This table has a foreign key on movie.id to encode which movie this is an alternative title for.

keyword is a table of keywords associated to the movies.

movie_keyword associates movie titles with keywords (two foreign keys).

movie_link is a set of links between movies. Each tuple in movie_link references the first and second movie by means of foreign keys into title.id; the fourth attribute is the type of the link (foreign key into kind_type).

link_type comprises the categories of various links between movies, such as: movie m_1 is a remake of movie m_2, m_1 is similar to m_2 etc.

movie_rating is a table that comprises rating information about movies. Rating information about a movie can be of up to five different types: (i) the average rating, (ii) the number of votes expressed on the movie, (iii) whether it is in the top 250 of IMDB, (iv) whether it is in the bottom 10 of IMDB, and (v) the detailed vote distribution[1].

company has information about companies involved in the movies; companies have an associated company type (foreign key into company_type, see below).

movie_company associates movies with companies: each tuple in this table has exactly two foreign keys, one into movies and the other into companies.

company_type comprises 18 roles that companies may play with respect to a movie: distributor, producer, special effects etc.

person is a table with information about people involved in movies. It has the people's names, as in *Garbo, Greta*, and their gender.

person_info stores information about people, much in the way movie_info has information about movies.

aka_name is a table of alternate names for people. This table has a foreign key on person.

role_type comprises the categories of people that are involved in a movie: actors, producers, composers, costume designers etc.

char_name comprises movie character names.

cast_info contains associations between a person, a role, and a movie, to signify that the person played that role in that movie. Accordingly, each tuple in cast_info has a foreign key on a person, one on a movie, and one on char_name. Further, a cast_info tuple may also have another foreign key on role_type, to say what kind of role this was (actor, producer etc.) This table also has a field note, which, when not null, gives some extra information. For instance, in a tuple specifying that x was played the role of *director* in movie m, the field *note* may be used to say: "assistant: y" to denote that y was the assistant director. (The table role_type does not have an entry for assistant director.)

comp_cast_type is a table of four "codes" used to characterize *the type and quality of the movie cast information available within IMDB about a specific movie*. This table is a bit unusual. First, it contains a tuple *cast* and a tuple *crew*: cast is the set of people that appear in the movie, whereas crew is the set of all people who did something for the movie (the crew plus e.g. the make-up artists, the music composer etc.) Second, it contains a tuple *complete* and a tuple *complete+verified*, to specify whether the cast information has been verified or not. Clearly, the choices between (cast and crew), on one hand, and between (complete and complete+verified), on the other hand, are orthogonal.

complete_cast characterizes the information available in the database about the complete cast of a certain movie. It has a foreign key to the movie, and: (i) one foreign key to comp_cast_type to specify whether the information available is about the *cast* or *crew*; and (ii) a second one to specify whether the cast is considered *complete* or *complete and verified*.

[1] Each detailed vote distribution information is *a string of length 10*, where each character may be either between 0 and 9, or '.' (dot), or '='. The character at position i, $1 \leq i \leq 10$, states which % of the ratings of the movie had value i (movies can be rated from 1 to 10). Thus:

- '.' specifies that 0% of the voters gave rating i;
- 0 specifies that at most 9% of the voters gave rating i;
- 1 specifies that at least 10% and less than 20% of the voters gave rating i and so on,
- 9 specifies that at least 90% but less than 100% of the voters gave the rating i,
- '=' specifies that 100% of the voters gave the rating i.

3 Server set-up and loading

3.1 Server set-up

This project requires you to install **PostgreSQL 9.5 or later** (`https://www.postgresql.org/download/`). One the installation is complete, you can follow the steps described at: `https://wiki.postgresql.org/wiki/First_steps` to check that everything is running fine. You will need to start a **server**, and then run commands through a **client**. You may also (but do not have to) install a graphical interface to administer the databases, such as pgAdmin (`https://www.pgadmin.org/`).

If not already done, start the installation as soon as possible.

In case of problems: `http://www.stackoverflow.com` will probably prove useful. If this is not enough, check among your colleagues. If the problem is yet not solved, we will be happy to help you: contact us by mail, sufficiently in advance to ensure that your installation is working fine for the first lab session.

Postgres' online documentation is helpful: `https://www.postgresql.org/docs/9.5/static/index.html`

3.2 Loading the data

The data comprises one file per table. Each file holds the triples in `csv` (comma-separated values) format:

- the first line contains the name of the columns, separated by a comma;
- all the remaining lines correspond each to a row of the considered table.

For instance, the file link_type.csv contains 19 lines, the first one containing "id,link", the remaining ones containing an id and a text field, describing a kind of relationship between two movies. To load the data, you should:

1. create a database;
2. create a table for each file; note that foreign keys may introduce order constraints between the different table creations;
3. import the content of each file into the respective table, through a **copy** (`https://www.postgresql.org/docs/9.5/static/sql-copy.html`) operation.

For instance, for the file link_type.csv, assuming the database is already created

- create a table having an id column of type integer and a link table of type text;
- run: `copy link_type from path/to/link_type.csv DELIMITER ',' HEADER CSV`.

Should you decide to index a table, doing it *after* the data is loaded in the table is probably faster. *Loading the data may take hours, especially if not done in the most efficient way. You may want to prepare a loading script and let it run e.g. during the night.*

4 Queries

1. Find the title and the year of production of each movie in which Nicholas Cage played. Order the results by year in descending order, then by title in ascending order.
2. Find the name of each actor who played the character Morpheus in a video game, together with the name of the game. Order the results by year in descending order, then by title in ascending order.
3. Find the name of all the people that have played in a movie they directed, and order them by their names (increasing alphabetical order).
4. Find the name of all the people that are both actors and directors, and order them by name.
5. Find the titles of the twenty movies having the largest number of directors and their number of directors, ordered by their number of directors in decreasing order.

6. Find the titles of the movies that have only 1 and 10 as ratings, and order them by average rating (decreasing).

7. Find the average number of cinema movies Dolores Fonzi played in, in her active years. A year is active if she plays in at least one movie produced that year.

8. Find all the pairs of titles of movies m1 and m2 such that m1 directly or indirectly references m2, ordered first by the title of m1, then by the title of m2. We say there is an indirect reference from m1 to m2, if either (*i*) m1 references m2, or (*ii*) m1 references m3, and m3 indirectly references m2.

5 What to turn in

Turn in the Moodle a file called `FirstnameLastname.tar.gz` or `FirstnameLastname.zip`, which is a .tar.gz or .zip archive of the following set of files:

1. For each query:
   - an SQL expression computing the query, in a file name `Qi.sql`, where `i` is the query number
   - the output of the SQL expression on the provided database, in a file name `Qi.out` (obtained for instance through `psql inf553 -f Qi.sql > Qi.out`)
2. A plain-text file called `description.txt` with three sections:

   A. Loading Provide here all the commands you used to load the data and possibly create indexes on it.

   B. Query plans Pick any 2 queries and provide the query plans that Postgres uses to run them (use the `explain` command).

   C. Improving query performance Pick any 2 queries (those above, or different ones) and propose for each an optimization so as to make it run faster. Provide: the query numbers, the time it took before and after the optimization.