2019

Junior Data Scientist

DATA LOADING & ANALYSIS WITH POSTGRESQL

SURYA VARA PRASAD EDUPUGANTI

1 Overview

We provide a set of files here:

https://drive.google.com/open?id=1-t46vy8xexNKruJFe4zBCzRRbG29weyp

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 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

3. ER diagram

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 di erent 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 rst 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 ve di erent 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 com-pany 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 e ects 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 eld 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 eld 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 speci c 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+veri ed, to specify whether the cast information has been veri ed or not. Clearly, the choices between (cast and crew), on one hand, and between (complete and complete+veri ed), 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.

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 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.

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, 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 le; note that foreign keys may introduce order constraints between the di erent table creations;
3. Import the content of each file into the respective table

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 rst 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 & Where

1. For each query:
    a. a 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.csv
2. A SQL file called all.sql showing creating table & loading data
3. ER diagram
4. Excel diagrams diagrams.xlsx

6 Comparison of chosen relational (PostgreSQL) to a NoSQL database (MongoDB)

For this exercise, I have chosen MongoDB as a NoSQL fit for comparing to PostgreSQL.

Introduction to both the databases

PostgreSQL: PostgreSQL is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards compliance. PostgreSQL is ACID-compliant, transactional, has updatable and materialized views, triggers, and foreign keys. It also supports functions and stored procedures.

PostgreSQL uses tables, constraints, triggers, roles, stored procedures and views as the core components that you work with. A table consists of rows, and each row contains a same set of columns. PostgreSQL uses primary keys to uniquely identify each row (a.k.a record) in a table, and foreign keys to assure the referential integrity between two related tables.

MongoDB: MongoDB uses JSON-like documents to store schema-free data. In MongoDB, collections of documents do not require a predefined structure and columns can vary for different documents.

MongoDB has many of the features of a relational database, including an expressive query language and strong consistency. However, since it is schema-free MongoDB allows you to create documents without having to create the structure for the document first.
A useful comparison with relational database management systems (RDBMS) in which you have: Table | Column | Value | Records. In comparison, in MongoDB you have: Collection | Key | Value | Document. This means that collections in MongoDB are like tables in RDBMS.

Documents are like records in a RDBMS. Documents can easily be modified by adding or deleting fields without having to restructure the entire document.

Are Indexes Needed?

Indexes enhance database performance, as they allow the database server to find and retrieve specific rows much faster than without an index. But, indexes add a certain overhead to the database system as a whole, so they should be used sensibly.

Without an index, the database server must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more costly operation.

PostgreSQL: PostgreSQL includes built-in support for regular B-tree and hash indexes. Indexes in PostgreSQL also support the following features:

- Expression indexes - created with an index of the result of an expression or function, instead of simply the value of a column
- Partial indexes - index only a part of a table

MongoDB: Indexes are preferred in MongoDB. If an index is missing, every document within the collection must be searched to select the documents that were requested in the query. This can slow down read times.

What Types Of Replication / Clustering Are Available?

Replication enables you to have multiple copies of the data copied automatically from 'master' to 'slave' databases. Multiple benefits to this process include:

Backup
Spreading the load to improve performance
Analytics team can work on one of the slave databases, thus not hurting the performance of the main database with long-running and intensive queries

Clustering, in the context of databases, refers to using shared storage and putting more database front-ends on it. The front end servers share an IP address and cluster network name that clients use to connect, and they decide between themselves who is currently in charge of serving clients requests.

PostgreSQL: PostgreSQL replication is synchronous (called 2-safe replication), so that it utilizes two database instances running simultaneously where your master database is synchronized with a slave database. Unless both databases crash simultaneously, data won't be lost.

With synchronous replication, each write waits until confirmation is received from both master and slave. For more information, please refer to the detailed wiki.

MongoDB: A replica set in MongoDB is a group of mongod processes that maintain the same data set. Replica sets provide redundancy and high availability, and are the basis for all production deployments. For more information, please refer to the detailed manual.

Which Database Is Right For You?

PostgreSQL: PostgreSQL seems to be gaining more popularity. If you're looking for a solution that is standard compliant, transactional and ACID compliant out of the box and has wide support for NoSQL features, then you should check out PostgreSQL.

MongoDB: MongoDB can be a great choice if you need scalability and caching for real-time analytics; however, it is not built for transactional data (accounting systems, etc.). MongoDB is frequently used for mobile apps, content management, real-time analytics, and applications involving the Internet of Things. If you have no clear schema definition, MongoDB can be a good choice.