Ayan Sarkar

University of Denver

Project 02 PostgreSQL vs. Neo4j

**PostgreSql Database Design**

1. Please use the below python script to generate the sql load scripts used for loading the data in the tables.

Postgresql\_GenerateSqlFiles.py

1. Please follow the below steps in the postgresql Docker command line to set up the database:

createdb SmallTownHospital;

psql SmallTownHospital;

\i /app/postgres/projects/project02/postgres/TableDropCommands.sql

\i /app/postgres/projects/project02/postgres/TableCreateCommands.sql

\i /app/postgres/projects/project02/postgres/LoadDoctor.sql

\i /app/postgres/projects/project02/postgres/LoadPatient.sql

\i /app/postgres/projects/project02/postgres/LoadTreatedBy\_Dcotors.sql

\i /app/postgres/projects/project02/postgres/LoadTreatedBy.sql

\i /app/postgres/projects/project02/postgres/LoadIllness.sql

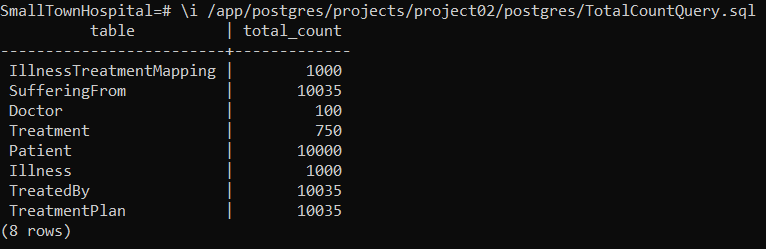
\i /app/postgres/projects/project02/postgres/LoadTreatment.sql

\i /app/postgres/projects/project02/postgres/LoadITMapping.sql

\i /app/postgres/projects/project02/postgres/LoadSufferingFrom.sql

\i /app/postgres/projects/project02/postgres/TotalCountQuery.sql

1. On executing all these scripts, we get the below data count across all the tables:



1. Please find below the details of the database objects that have been created:

* Doctor: This table contains all the details of a doctor that might be recorded in a hospital. A special thing to notice in this table is that the primary keys of this table begin with the letter ‘D’
* Patient: This table contains all the details of a patient that might be recorded in a hospital. A special thing to notice in this table is that the primary keys of this table begin with the letter ‘P’
* TreatedBy: This table maps the patients to the Doctors who are treating them.
* tr\_before\_insert\_or\_update: This is a trigger that ensures the requirement that the Doctor could be a patient too, receiving treatment in the hospital. This trigger is for the table: TreatedBy.
* tr\_before\_insert\_or\_update\_2: This trigger on the table: TreatedBy enforces the requirement that the patient can’t be treated by more than 5 doctors.
* Illness: This table records the name and severity of an illness
* Treatment: This table records the names of the various treatments available at the hospital.
* IllnessTreatmentMapping: This table maps the illnesses to the treatments prescribed for treating them.
* SufferingFrom: This table maps the patient, their doctor and the type of illness that they are getting treated for.
* tr\_after\_insert: This is a trigger on the table: SufferingFrom, which enforces the requirement that any patient who have an illness, should receive at least one treatment.
* TreatmentPlan: This is a table which maps the patient, their doctor and the type of illness they are suffering from and the treatment for that illness.

**Neo4j Database Design**

1. Please use the below python script to generate the csv load scripts used for loading the data in the nodes and relationships.

Project2\_Neo4j\_GenerateCsvFiles.py

1. Please follow the steps in the exact sequence given in the below script which contains the commands to create and load all the nodes and relationships in the graph database: Creation\_Data\_Load\_Script.txt

In this script, the generated csv files for the various nodes and relationships are first copied to the Neo4j’s import folder and then the data is loaded from the files to the database. Then the “Create Index” command has been sued to enforce the primary key/unique constraint conditions.

1. Once all the commands are executed in the script, we get the below nodes and relationships in the database. Please find below their descriptions:

Nodes:

* Doctor: This node represent all the details of a Doctor.
* Patient: This node represent all the details of a Patient.
* Illness: This node represent all the details of an Illness.
* Treatment: This node represent all the details of the treatments.
* PATIENTTREATEDBYDOC: It’s a relation which depicts the relationship of the patients being treated by their respective doctors.
* DOCTREATEDBYDOC: It’s a relation which depicts the relationship of the doctors who are patients, being treated by another doctors.
* MAPPEDTO: It’s a relation which depicts the association of an Illness and its Treatment
* SUFFERINGFROM: It’s a relation which depicts the association of a patient and the illness they are suffering from.
* DOCSUFFERINGFROM: It’s a relation which depicts the association of a doctor as a patient and the illness they are suffering from.
* TREATMENTPLAN: It’s a relation which depicts the association of a patient and the illness they are suffering from and the treatment which has been prescribed for that illness.

**PostgreSql Vs. Neo4j**

My experience with Postgres was pretty ok; some of the relationship requirements of the project were a bit tricky at first to implement, but I finally got them working. The one thing that is very particular of this database, is that the length of the query increases significantly when answering some complex questions on the data, it happens to involve multiple tables and joins and sometimes becomes hard to track, in case any changes are to be made to the logic. This database should definitely not be opted for an audience who are not familiar with SQL and its intricacies to extract data from the database; but on the other hand, Postgresql is very useful when it comes to designing variety of constrains/conditions on your database, only sky is the limit I guess.

When it comes to Neo4j, I found designing database constraints (like the ones done in Postgresql) to be very difficult to envision and in some cases not possible at all. I had to really change of way of thinking to figure out the ways to implement the relationship among the tables in the relational databases to convert into nodes and relations in the graph database. Some of the limitations that I can across in Neo4j was that bi-directional relations are not possible, we need to go around that limitation by using a directionless association in a relation ( )-[ :relation]-( ). Another limitation is that we cannot use a prior defined relation to build another relation. The one thing I liked about this database is that the queries to extract data are much simpler and shorter in size, when compared to Postgresql. Also, I enjoyed working with the Neo4j Web console, its very user friendly. This database is meant for an audience who wants an easy means to extract data and view it and suitable for more simpler designed use cases or domains. Its an ill-suited match for use cases which have a lot of complexity and depth that needs to be designed in the database.

So, these were my 2 cents about my experience with the two databases.