IIC3432 — Advanced topics in databases

Homework 1: explanation and projects

1. Administrative stuff

You should form groups of two people each. Your group will select one of the projects detailed below, investigate the topic, and present the findings in the class on 31st of August. Each group will have precisely 30 minutes for a presentation, including discussion and questions.

2. FAQ

Do we need to hand in the work we do?
No, you will be evaluated based on the presentation.

What should be the structure of my presentation?
You have 30 minutes, so there is sufficient time to quickly explain the setting and background, and then detail the specific problem that is tackled, and how is this resolved. The idea is that all the students in the class should be able to follow your presentation, not just the lecturer.

We have a size 9 font in our presentation so that all the information fits on the slides. Is this OK?
No! An important part of the evaluation is your ability to summarize the work you did in a short presentation. This is no easy task. In particular, a good short presentation requires better understanding of the topic than a bad long presentation. In this course you will need to be able to isolate the important details of your work and present it in an accessible way.

I don't like any of the proposed topics.
Propose a new topic. For this, send me an email (dvrgoc@ing.puc.cl).

We're stuck, we don't know how to proceed, or can not decide what to do next.
That's what I'm here for. During next week, I'll be around during the lecture hours, so if you decide to come to the lecture room and work at that time, you can ask me questions directly (I'll be in my office, or in the lecture room). Also, you can send an email at other times. Some projects have a lot of work in them, so if you do not finish all of it you still might get the highest score if sufficient effort is put into it.

3. Projects

A representative of the group should send me an email ccing the other person in the group with two preferences for a project they would like to do. It is recommendable that you do this before Monday 21st of August at 15pm to get the topic you wish to work on. One topic can not be taken by two groups.

Bellow follows a list of projects.

3.1. How is Neo4j implemented?

There is an open source version of Neo4j available at `https://github.com/neo4j/neo4j`. According to Neo Technologies, the community edition of Neo4j is completely open source. Your task is to clone their repository, and go through the source code to investigate two things:

1. How is the property graph data model implemented. What are the storage structures, how is the data loaded, etc.

2. How are basic queries implemented. Namely, how is a basic MATCH pattern evaluated, and how are the recursive patterns MATCH (a) -[n–m]- (b) and MATCH (a) -[*]- (b) implemented.

3.2. Indexing in Neo4j

Explain and analyze what sort of indexing facilities are available in Neo4j. When is such an index useful? Demonstrate the usefulness of a particular index through experiments.

For this, you can either generate synthetic data, or find a real world dataset to load into Neo4j (some are available on the Neo Technologies website). In both cases you should design a set of testing queries that show when the index is useful, and when it is of no use. In the case you chose to generate your own property graphs you can experiment with the size of the graph to see when an index is becoming useful.

The best place to start with this would be `http://neo4j.com/docs/developer-manual/current/cypher/schema/`.

3.3. Can Neo4j beat a SQL database?

We have discussed why a graph database can be more efficient when answering a certain type of queries. It is time to put this to a test. In this project you are asked to test how the performance of a graph database compares to that of a SQL database. You should install a SQL database, such as PostgreSQL, as well as Neo4j, and run tests to determine which one runs better and in which scenario.

There are two test scenarios you should include in your study:

- The gMark graph database benchmark available at `https://github.com/graphMark/gmark`. This is a graph database benchmark generating both graphs and queries. It can generate queries in Cypher and SQL. More details behind the project can be found in the paper `https://arxiv.org/pdf/1511.08386.pdf`. Clone the benchmark and see what types of queries it will generate. (Note: There is a small error when running the github demo; namely some XML files are missing the `size` parameter. The file `test.xml` has the correct configuration, so you should check that one and imitate it. Size is just the number of graphs that will be used. The important thing is to configure the parameters at the very end of the XML file.)

- You should also manually generate the queries to test. This can be done either on a fixed real world dataset you find online, or over a graph database you generate manually. In both cases you should explore when the graph structure is advantageous, and when SQL runs better. In both cases you should include a recursive Neo4j query ([*] in a match path) and compare it to a recursive SQL query, as well as an iterated recursive query ([1..k] in Neo4j and a long join in SQL). Make sure your databse has non trivial paths for this.

3.4. Formalizing the core of Cypher

In class we show the benefits of having a formally defined data model, syntax and semantics of a query language. This was still not formally done for Cypher. In this task you are asked to take a set of several core operations of Cypher and define formally their syntax and semantics (similarly as we have done in the class for regular path queries). This should faithfully reflect what Cypher actually does, which semantics it uses (no repeated edges), etc. A quick reference card of all the operator available in Cypher is available at `http://neo4j.com/docs/cypher-refcard/current/`.
Your formalization must include the MATCH-RETURN operations, and has to be able to handle how multiple MATCH operations are stacked together to simulate arbitrary semantics (as we have shown in class).

3.5. Path manipulation in Cypher

We showed some basic queries that return and manipulate paths in Cypher. In this task you are asked to do an extensive analysis of such queries that can return paths, iterate over its elements, look for shortest paths, etc. In particular, you should answer the following questions: What types of paths does Cypher return? What happens when there are loops? How does one look for shortest paths? Can one find more than one shortest path? What other path functions are there? A very brief discussion on this can be found at `https://arxiv.org/pdf/1610.06264.pdf`, appendix A1 and A2 (here you can find an NP-hard query that is actually implemented in Cypher).

In this task you should isolate such features, explain how they operate (informal semantics), and explore how they are evaluated both over real world datasets (line e.g. the movie database

we played with), and synthetic graphs you will generate to test when such queries run into issues. A good idea for the latter is to show a graph of how the running time of a particular query is affected by the size of the database (plus a database should have the feature the query is exploring; e.g. loops, etc).

3.6. Other graph databases: Gremlin

Neo4j is not the only graph database system. In this task you are asked to explore another graph database system/language: Gremlin. Gremlin is a free graph database available at `http://tinkerpop.apache.org/`. You should download and install Gremlin and compare its functionalities with those of Cypher. In particular, you should explore basic and complex graph patterns and how are these expressed in Gremlin, and then do a more detailed survey of path queries available in Gremlin. Some interesting questions here: Can all regular path queries be expressed in Gremlin? What Neo4j functionalities can be expressed in Gremlin, which ones can not?

You should also do a small comparative test of how the two graph engines (Neo4j and Gremlin) compare in terms of efficiency. For this you can use a real world graph (e.g. export the movie graph from Neo4j to Gremlin), or create your own graphs and show when one system runs better than the other and discuss why this might be the case.

3.7. Propose your own topic

Quite simple: send me an email (dvrgoc@ing.puc.cl) to confirm the topic you would like to explore.