IIC3432 — Advanced topics in databases

Neo4j

1. Before we start (actually we are already starting)

We already know a bit about how Neo4j and its query language, Cypher, work. The idea of this class is to get comfortable working with Neo4j and have no issues in creating graphs, loading them into the system, or asking a range of graph queries in this system.

Before we start playing with the system itself, take a few minutes to think about how the system could be implemented and what would be your idea of how one can efficiently manage nodes (and edges) that carry a lot of data, while at the same time being capable of working with the graph structure and running graph queries.

2. Basic and complex graph patterns in Cypher

Load Neo4j and run the small movie database demo to learn how to create graphs and run basic queries.

To see what is the graph you are working with, you can always run the following query:

```
MATCH (n) RETURN n
```

Now load the bigger movie database and we will start playing with it.

2.1. Cypher

Cypher is the query language of the Neo4j database. You can find the Cypher reference card at `http://neo4j.com/docs/cypher-refcard/current/` and its full manual at `https://neo4j.com/docs/developer-manual/3.2/cypher/`. You will probably need to use those to complete this guide. Google is also your friend.

The basic structure of Cypher queries is quite similar to SQL.

It is also very user friendly when you need to define basic graph patterns. To do this, we need to learn how to specify patterns in Cypher.

Patterns. These can be defined in Cypher as follows:

- `(p) --> (q)` and `(p) <-- (q)`, that represent a directed edge going from `p` to `q` or from `q` to `p`, respectively, and `(p) --> ()`, that represent edges from `p` to *any* node. Here `p` and `q` represent node IDs.
- `(p) -[n:LABEL]-> (q)`, representing a directed edge with the ID `n` going from `p` to `q`, and carrying the label `LABEL`.
- One can combine such patterns; e.g. `(p) <-- (q) --> (r)` or if we wish to use labels: `(p) <-[n:LABEL]- (q) <-- (r)`

Queries. To specify a query that actually runs one has to `MATCH` a pattern onto the graph and `RETURN` some node/edge IDs. This is similar to the SQL SELECT FROM WHERE queries. The basic form of Cypher queries is the follows (`p` is a pattern as above, and n_i IDs in this path):

```
MATCH p
RETURN n1, n2, ...
```

To return all the variables in the pattern simply use `*`.

Trying basic queries. Try writing a few basic queries and run them over the movie database. Try to retrieve all the movies. Try to retrieve all the actors. Try to find all the movies where a specific actor acted. Try to find all the movies that have a director specified. Try to find all the movies where the director also acted in that same movie.

Time for Bacon. This will refer to the examples we have seen in class last time. Write the following queries/answer the following question:

- Find all pairs of actors that starred together in the movie Unforgiven.
- Check the result of the query above. Is the same actor ever repeated? Why is this? What semantics is used by Cypher?
- Can we simulate homomorphism based semantics in Cypher? Try chaining two `MATCH` clauses. E.g. `MATCH p1 MATCH p2 RETURN *`. What happens with the query above if we do this? Can we always use this trick?
- Find all the movies where Clint Eastwood either acted or directed them (or both). Use `UNION ALL`.
- Find all the movies where Clint Eastwood acted, but that he did not direct. Which operator would you use here?
- If you don't know what it is, google Bacon number.
- Find all the movies where Kevin Bacon acted.
- Find all the actors that acted with Kevin Bacon in the same movie.
- Find all the actors that have a Bacon number less then or equal to two. Can you do this without using `UNION ALL`?
- Find actors that acted in at least two different movies with Kevin Bacon.
- Stacking two `MATCH` clauses also allows to define graph patterns that are not on a "straight line". Try to define some such query over our database.

Crazy Neo. Try running the following query: `MATCH c= (p) -[n]-> (q) RETURN * LIMIT 10` What is the `c` that is being returned?

3. Navigational queries (paths)

Next we move onto path queries. The idea is to see what type of navigational features is offered in Cypher and what are some of their limitations.

The main navigational feature in Cypher are regular path queries. E.g. one can use `*` to denote arbitrary path, or use `knows*` to denote the regular path query traversing only edges labelled with `knows`. There is also an option to specify how many times an edge label should be repeated (e.g. `[knows*2..7]`).

For this exercise create an empty database (an empty folder to load into Neo4j) and then create the social network graph we showed in class. Now try and answer the following queries over this graph:

- The friend-of-friends relation over this graph. All pairs of people who are friends-of-friends (transitively).
- All pairs of nodes connected by any path. Is some pair repeated? How come?
- Add some more friends, posts and tags to your database. Now find all friends of friends of Julie that liked a post with the tag she follows.
- One can also return paths specified by a star. Try returning all friends of friends of Julie, together with the path witnessing this. Do you get infinitely many paths? Why?
- Try the query above, but returning only shortest paths.
- Can you express a regular path query defined by the regex `a+ b+` ? Try some such query in your database.
- Can you express any regular path query using Cypher? Think about what you can not do.

Load either the small or the big movie graph (or create one by yourself) and express the following queries:

- All pairs of nodes that are connected to each other.
- All nodes connected to Kevin Bacon.
- Find all people that have a Bacon number.
- Verify if there is a path between Kevin Bacon and some actor X of your choosing.
- Find all people that have a Bacon number between 2 and 5.

Do all of these queries run well? When do you get into problems?

Returning paths. Recall that you can return paths in Cypher. You can also return recursive paths. Try to see what happens with some of the above queries when you want to return a path. Can you make this work?

Cypher also allows you to iterate over the elements of a path you return. How does one do this? Try it with some paths.

4. Aggregation

Load either the small or the big movie graph (or create one by yourself) and express the following queries:

1. Number of the nodes in the graph.
2. The number of movies Kevin Bacon acts in.
3. The average number of movies an actor stars in.
4. The actor that acts in most of movies (i.e. no other actor acts in more movies that this).
5. The actor with the most paths to Kevin Bacon.

Can all of these be expressed? Do they all run efficiently?

5. Panama papers

Note that these are not legal documents, so if you find someone implied in dirty business here it is not going to send them to jail.

Since you downloaded this dataset as well load it and do the following:

- Describe how the data is structured here. What are the basic types of entities? What are the edge labels?
- It was found that the president of Azerbaijan, Ilham Aliyev, has some interesting links in this dataset. Find a node corresponding to him.
- Can you find the president Ilham's relatives in the database?

- Find offshore companies in the dataset. Find if the president Ilham is somehow connected to them.
- Given the structure of the database, what would be an interesting pattern to find that is not just a one way path? Express some such query.