THE UNIVERSITY OF SYDNEY

School of Information Technologies
Dr. Ying Zhou

COMP5338: Advanced Data Models **2.Sem./2018**

Week 8: Neo4j Indexing and Data Modeling

18.09.2018

Learning Objectives

In this week, we will use the movie data set from the build in tutorial to observe various query execution plans with or without index. We will also practice graph data modeling by augmenting the original data set with tag and genre information.

Question 1: Neo4j Query plan and Indexing

In this exercise we will use the Movie Graph in the buid in tutorial to illustrate query plan and index usage. You can view the query plan by adding `explain` or `profile` command in front of any query. The `explain` command does not execute the query, you will see a plan with estimated results size at each stage. The `profile` command displays the query plan along with actual result size at each stage.

a) This question assumes that there is no index on the Movie Graph. Use the `:schema` command to check available indexes and drop any that you may have created in week 7 tutorial. The following three command illustrates query plans and estimated cost with full nodes scan and with node label scan. If you want to run the query and get the actual running cost, replace the operator `explain` with `profile`.

- Profile a query using full nodes scan

  ```
  explain
  MATCH (cloudAtlas {title: ''Cloud Atlas''})<-[:DIRECTED]-(directors)
  RETURN directors.name
  ```

  This query finds all directors who have directed something titled "Cloud Atlas". Since we did not specify the label of the node, the query starts with a full node scan. It is followed by a filtering stage to find all nodes with title "Cloud Atlas" and the nodes are saved in a temporary variable `cloudAtlas`. The expand stage follows the `DIRECTED` relationship of the `cloudAtlas` nodes to finds the `directors` nodes. The project stage extracts only the `name` property of the directors nodes.

- Profile a query with node label scan

  ```
  explain
  MATCH (bacon:Person {name:''Kevin Bacon''})-[*1..4]-(hollywood)
  RETURN DISTINCT hollywood
  ```

This query finds all things that are within 4 degrees to a `Person` named "Kevin Bacon". The query starts with all nodes with label `Person` with a stage called `NodeByLabelScan`. It is followed by three other stages: `Filter`, `VarLengthExpand(All)` and`Distinct`. In the first stage, the total number of node scanned is smaller than that in the previous query.

- Profile a query with at least two node label scan plans

```
Explain
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)
where ''Neo'' in r.roles
RETURN p.name, m.title
```

This query finds all person that has played the role "Neo" in some movie. The query may start with all `Movie` nodes, or with all `Person` nodes. Because there are a lot more `Person` nodes (about 133) than `Movie` nodes (about 40) in the graph. The query planner picks the plan starting with all `Movie` nodes.

b) Now create the following indexes on Person and Movie nodes:

```
CREATE INDEX ON :Person(name)
    CREATE INDEX ON :Movie(released)
```


Profile the following two queries to compare their execution plan.

- Profile a query using index

```
explain
MATCH (bacon:Person {name:"Kevin Bacon"})-[*1..4]-(hollywood)
RETURN DISTINCT hollywood
```


This query is the same as the second query in the last question. The execution cost is quite different because the targeting graph has an index on the name property now. The query execution starts with a single node having the property.

- Profile a query not using index

```
explain
MATCH (bacon{name:"Kevin Bacon"})-[*1..4]-(hollywood)
RETURN DISTINCT hollywood
```

This query would get the same result as the previous one. Because the node label is not specified, the index cannot be used. It starts with a full node scan.

Where sub-clause can also utilize index, for instance, the execution of the following query start with a `NodeIndexSeekByRange` stage:

```
match(m:Movie) where m.released > 2000 return m
```

Now write a query to find out the person that have co-starred the most with Tom Hanks and inspect the query plan.

Question 2: Neo4j Data Modeling Practice

Neo4j schema design is very similar to domain modeling. In general the main activities involve making decisions on whether certain piece of data should be modeled as `node`, `relationship`, `property` or just `label/type`. In this exercise, you will be asked to augment the original data with two additional piece of information: genres and tags. A movie may belong to one or many genres. Each genre is represented as a string. Tags can be assigned to movies by any Person. The database needs to keep the information of who assigns what tag(s) to which movie. Table 1 shows a few sample data related with genre and tags.

Table 1: Sample Genre Data

Title	Geners
Sleepless in Seattle	Comedy |Drama |Romance
The Da Vinci Code	Mystery |Thriller
Apollo 13	Adventure |Drama |History

Table 2: Sample Tag Data

Person Name	Movie Title	Tags
Jessica Thompson	Sleepless in Seattle	Tom Hanks, Secret
Jessica Thompson	Apollo 13	Tom Hanks
Mary Sharp	Sleepless in Seattle	Seattle, Romance
Mary Sharp	Apollo 13	Space, Tom Hanks
Angela Scope	The Da Vinci Code	Holy Grail, opos dei
Angela Scope	Sleepless in Seattle	Tom Hanks

Write Cyhper queries to update the graph based on your design and try to run some queries to find out

1. all movies with tag "Tom Hanks"
2. The number of movies with tag "Tom Hanks"
3. all movies in the "Drama" genre
4. all the users who tagged a movie in "Drama" genre
5. The average rating of all movies in "Drama" genre
6. The average rating of each movie in "Drama" genre
7. Assuming a few tags have been added to movie "Apollo 13", find the most frequent tag of this movie