Exercise 3 (Summer 2019)

6.0 VU Advanced Database Systems

Information

General

Work through the exercises below and write a report on your answers. Submit the report as a single zip file containing your report as pdf and the JSON files for Exercise 2 in TUWEL and register for an exercise interview. **You can only receive points for this exercise if you attend the interview.** We expect you to explain your work in your report, i.e., it is not enough to only write the final answers. Show how you arrived at your answers and, where applicable, discuss your results. **We will not accept any handwritten reports!**

Deadlines

at latest June 17th	**12:00**	Upload your submission on TUWEL
at latest June 17th	**23:55**	Register for an exercise interview in TUWEL

Exercise Interviews

In the solution discussion, the correctness of your solution as well as your understanding of the underlying concepts will be assessed. **Every group member has to be able to explain all parts of your submission.**

The scoring of your submission is primarily based on your performance at the solution discussion. Therefore it is possible (in extreme cases) to get 0 points even though the submitted solution was technically correct.

Please be punctual for your solution discussion. Otherwise we cannot guarantee that your full solution can be graded in your assigned time slot. Remember to bring your student id to the solution discussion. It is not possible to score your solution without an id.

Question Sessions

About a week before the submission deadline, we offer question sessions to help you with any problems you have with the exercises. The goal of these sessions is to help you understand the material, not to check your solutions or solve the exercises for you. In this spirit we also ask that you engage with the exercise sheet before coming to the session. **Participation is completely optional.**
Exact times and locations will be announced on TUWEL.

TUWEL Forum

You can use the course forum on TUWEL for clarifying questions regarding the exercise sheets. Please do not post your solutions (even partial) on the forum.

Exercises

Exercise 1 (Distributed Joins) *[2 points]* Find the distributed execution strategy for the joins that minimizes communication cost (number of bytes transferred in this case). To do this, compute the communication costs for the strategies presented in the lecture (slide 46 onward of the DDBMS slides) and compare the results.

For the whole exercise assume a distributed database with 4 sites and three relations `Users, Orders, Products`. The details of the scenario are described in Figure 1. You can assume that all relations are non-fragmented and all records (and attributes) are fixed-size.

Relation	Site	# Records	Record Size (byte)
Users	1	40 000	800
Orders	2	900 000	30
Products	3	1500	2 000

Relation	Attribute	Size
Orders	price	4
Orders	prod	8
Orders	user	8
Users	uid	8
Users	name	100
Products	pid	8

Figure 1: Scenario for Exercise 1

(a) Compute the communication cost of all strategies from the slides (and listed below) and find the strategy that minimizes communication cost for the Query 1 if we want the result at site 4. Assume a selectivity of $\frac{1}{360\,000}$ for the join.

Strategies:

- Send both tables to site 4 and join there.
- Send `Users` to site 2, join there and send the result to site 4.
- Symmetrically: Send `Orders` to site 1, join there and send the result to site 4.
- Send only the join attributes of `Users` to site 2, semi join with `Orders`, send the result back to site 1 to compute the full join. Finally transfer the full join to site 4.
- The semi-join strategy in the opposite direction.

$$\pi_{name,price}(Users \bowtie_{uid=user} Orders) \tag{1}$$

Hint: For the result size of a semi-join $A \ltimes B$*, you can use* $\min\{|A \bowtie B|, |A|\}$ *as a worst-case estimate*

(a*) Vary the strategies above in such a way, that you always project to only the necessary columns as early as possible. How does the amount of data that needs to be transferred change?

(b) Now consider how to generalize the strategies of task (a*) above to situations with more than 2 relations and find the strategy with optimal communication cost for Query 2 given below. Again we want the result at site 4. Again, assume a selectivity of $\frac{1}{360\,000}$ for the join between `Users` and `Orders`. For the outside join assume a selectivity of $\frac{1}{5000}$.

For this task you can skip calculating the communication costs for all those strategies that cannot result in minimal communication costs. If you do so, *explain why you can skip the respective calculations.*

$$Products \bowtie_{pid=prod} \pi_{name,price,prod}(Users \bowtie_{uid=user} Orders) \tag{2}$$

Exercise 2 (Denormalization) *[3 points]*

Consider a library lending system implemented in a traditional RDBMS and assume that you want to migrate this system to use a document store using JSON representation instead. The relational model has 4 tables `book, edition, person, borrowed`; an example instance (the key is underlined in each relation) is given below. Conceive a data model for your document store system and translate the example instance to JSON files fitting your data model. You are allowed to add new attributes, denormalize relations, and make other similar changes for your new data model.

As discussed in the lecture, data models are designed with specific workloads in mind. Tradeoffs may be necessary, if so, describe them in your report. For this exercise consider the following parameters:

- Two queries should be fast:
  - Listing all people who have borrowed a book for more than two weeks and not returned it yet (not returned means no `to` value in the `borrowed` relation).
  - Asking if any edition of a book is currently available to be borrowed. (i.e., owned - borrowed but not yet returned books > 0)
- The `borrowed` relation is updated frequently (and needs to be efficient). For all other relations, updates are rare and can be more expensive.

Your submission for this exercise should have two parts. A description of your data model in your report and the resulting JSON files.

Book

id	name	authors
1	Computational Complexity	Papadimitrou
2	Parameterized Complexity Theory	Flum, Grohe
3	Fundamentals of Database Systems	Elmasri, Navathe

Borrowed

who	book	edition	from	to
Charles	2	1	01-03-2009	05-12-2010
Rachel	3	2	22-09-1994	17-01-1997
Rachel	3	7	05-11-2010	01-12-2010
Tom	3	7	01-05-2019	
Charles	1	1	21-03-1995	29-04-1995

An empty `to` value signifies that the book was not yet returned.

Edition

book	edition	year	isbn	owned
1	1	1993	0201530821	3
3	7	2017	1292097612	1
3	6	2010	0136086209	2
3	2	1994	0805317481	6
2	1	2006	3642067573	3

The owned column stores how many books the library owns in total, not how many are currently available. I.e., the number is not updated when somebody borrows a book.

Person

name	address
Charles	420 Paper St
Rachel	90 Bedford St
Tom	41-505 Kalanianaole Highway

Exercise 3 (Graph Databases) *[5 points]*

For this exercise you are supposed to familiarize yourself with Neo4j and the Cypher query language to experience the basic usage of graph databases. The lecture on graph databases contained an introduction to Cypher. Beyond that, we recommend the official tutorial[1] as a supplement, should you need one.

The exercise is based on the graph database representation of a large collection of leaked documents regarding offshore shell corporations and similar constructs created by the International Consortium of Investigative Journalists. The dataset is conveniently provided bundled in a special distribution of Neo4j that automates all the necessary setup. You can find the respective executables for Linux, Windows and OSX here: `https://offshoreleaks.icij.org/pages/database`.[2]

Intro to the Dataset

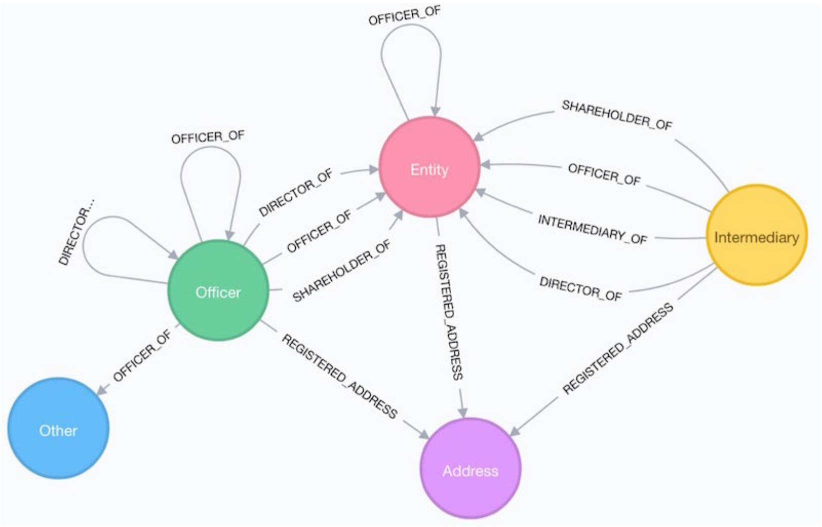


Figure 2: Main data model.

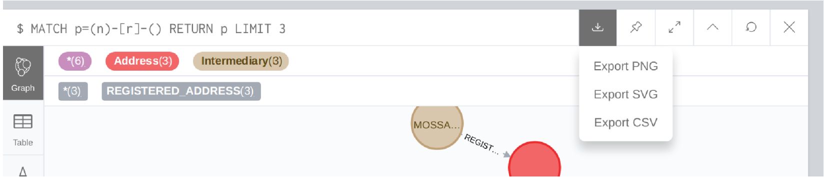
The best introduction to the dataset is through the official tutorial. It should open automatically after opening the Neo4j browser. If it doesn't, you can open it manually by executing the following command in the Neo4j browser.

```
:play https://offshoreleaks-data.icij.org/offshoreleaks/neo4j/guide/index.html
```

Following the few slides there should give you a basic overview of the dataset. In the official documentation the data model is described with the image in Figure 2, **this is outdated**. There are no more edges of types `DIRECTOR_OF` and `SHAREHOLDER_OF`, they have been all been combined together in the `OFFICER_OF` type. You can always see the full meta graph with `CALL db.schema()`.

[1] `https://neo4j.com/developer/cypher-query-language/`

[2] You will likely be prompted with the option to update the client and/or dataset. You can complete the exercise with or without the updates.



```
$ MATCH p=(n)-[r]-() RETURN p LIMIT 3
```

Figure 3: Export as Image From Neo4j Browser

Your Tasks

Provide the requested queries in your solution and where sensible also provide pictures of the output graph. The neo4j browser has in-built support for exporting the graph as an image (cf. Figure 3).

(a) Two basic queries to start with:

- List all the distinct countries for which addresses are registered in the database. *Hint: Output some nodes with the output format set to table to see what attributes are usually present in the nodes.*
- Choose one of these countries and list 5 entities which are based in that country.

Note: Neo4j (and most other) graph databases are schemaless by default. This means that the database does not verify that every Address or Entity does in fact have specific attributes. In all tasks of exercise 3 you can ignore nodes where necessary properties are missing. In the Neo4j Enterprise Edition it is in fact possible to add aspects of having a schema via constraints, for more information you can check out the Neo4j constraints guide (*not necessary for the exercise*): `https://neo4j.com/docs/cypher-manual/current/schema/constraints/`.

(b) i First, find the 10 `top intermediaries`, i.e. those 10 intermediaries that have the most outgoing `INTERMEDIARY_OF` edges. Output the name of the intermediary as well as the number of relevant edges.

ii Extend the query of i to **also** count outgoing `OFFICER_OF` edges to determine the top intermediaries. Output the name of the intermediary as well as the number of relevant edges.

iii For the top intermediary from query ii, output all the outgoing edges, except for those that have type either `OFFICER_OF` or `INTERMEDIARY_OF`. Also output the respective nodes at the other side of the edges. *Hint: You can access the type of an edge e in Cypher using `type(e)`.*

(c) Find a shortest *undirected* path between an Address in Luxembourg and an Address in Cyprus where the path has a length of at least 16 and at most 30. (*Hint: such a path exiss, i.e., the result should not empty.*) Neo4j provides the `shortestPath` function for these type of problems. Be aware that this will not return the singular shortest path between any nodes. Rather, `shortestPath` returns a shortest path for each match. For an example look at the output of:

```
MATCH p=shortestPath( (e:Entity)-[*]-(i:Intermediary) )
WHERE e.name=i.name AND i.name='A+ Fund'
RETURN e.node_id,i.node_id,length(p)
```

You will see that because there are three Entities that fit the criteria, we get the shortest path to each of them returned. (If you want to see the full relevant subgraph visually, change return statement to read just `RETURN p`)

(d) Find one (use `LIMIT 1`) subgraph of the form specified in Figure 4 in the database. Furthermore, the two entities should be in different countries and at least one of the entities should have different values in its `jurisdiction` and `country_codes` attributes.

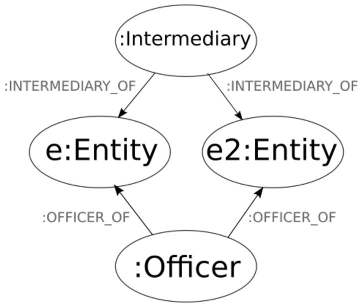


Figure 4: Subgraph for Query

(e) For the final task we want to make the `countries` attribute a real part of the graph. The best way to do this is in Neo4j is using the `MERGE` keyword inside of a `MATCH`. The task is restricted to the nodes with label `Other` to avoid processing an unnecessarily large amount of data.

- First, recall what you learned about the `MERGE` keyword in the lecture. As a supplementary source of information about `MERGE`, we recommend the documentation: `https://neo4j.com/docs/cypher-manual/current/clauses/merge/`.
- Write a query that for each node *o* with label `Other` performs the following actions:
  - Add an edge with type `IN_COUNTRY` going to a node with label `COUNTRY` that has a name attribute that matches the contents of `o.countries`.
  - Make sure that you **don't create duplicate countries or duplicate edges**.
  - If the node *o* has no `countries` attribute, do nothing for this node.
- When done, write a query to show all the nodes connected to your favorite country in the database. The output should look like the example in Figure 5.

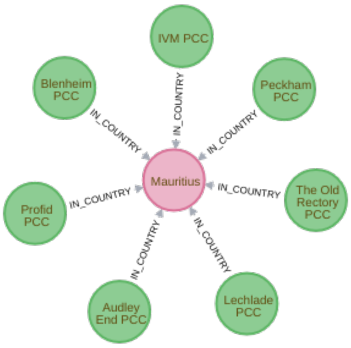


Figure 5: Example of Final Query Output

Hint 1: In some cases, the field `countries` contains multiple countries, separated by semicolons. You are allowed to simply exclude those cases (e.g., `NOT o.countries contains ';'`). If you want to challenge yourself, try finding a way to add the relationship for all the countries in the attribute. (This is completely optional, you can get full credits without doing this.)

Hint 2: If you've created wrong entries while experimenting, you can delete them using `MATCH (c:Country)-[r]-() DELETE r DELETE c;`