Kasper P, Kasper O, Christian S, Oliver L


# Introduction

We merged our project with the test-project, so we had a TDD method for writing code.

The database engine we used was MySQL and Neo4J.

# How did we import?

## Extraction from dataset

We had two datasets, one with all the cities with more than 15000 residents and the other was all the books downloaded from project gutenberg.
From the cities we extracted the ID, first variation of the city name, latitude and longitude and saved all to a file “cities.csv”.
From the books we extracted the title, author and all cities mentioned by referencing the extracted data from cities.csv, to “allformatted.txt”. They are “#” separated because authors and cities can include commas.
We then extracted all authors from the extracted data from books and assigned each an ID and saved to “formattedauthors.txt”. Also “#” separated.
Then we created a file containing IDs for books, which were assigned while creating the file, the author’s ID and the books title, to “formattedbooks”. Also “#” separated.
At last we created a file containing book ID and mentioned city ID for each line, called “book_city.txt”.

## MySQL

For MySQL we used the data we got from the extraction and wrote a Java-program which generated INSERT-statements.

We found that the fastest way[1] to perform the insertion was to use a single statement which had operations needed to add the data.

The program took about 5-15 minutes depending on machine and VM.

## Neo4J

For Neo4J we used the same data as MySQL which we got from the extraction. We started by using the same Java-program to import the data. It worked but the time (about 8 hours) and the memory usage was very high, so we looked for a different approach.

[1] https://stackoverflow.com/a/1793209/1514875

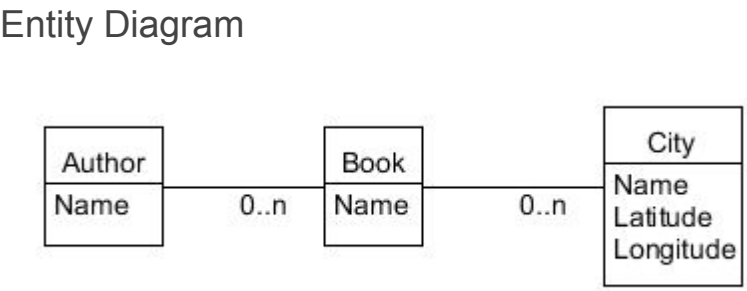
We found 2 different approaches. We could use `LOAD CSV..`, but that method was also really slow on large dataset (as the book<->city relation). The other one we found was the import-tools from Neo4J.

We found that the import-tools was the quickest, by a magnitude of millions. We transformed our data to a csv-format, that the import-tool could understand and ran the import. The **entire** import of both nodes and relations took about 8-15 seconds, with very low memory usage.

# How is the data modeled?

## MySQL

The data is split up into 3 main tables and 2 relationship-tables.




## Neo4J

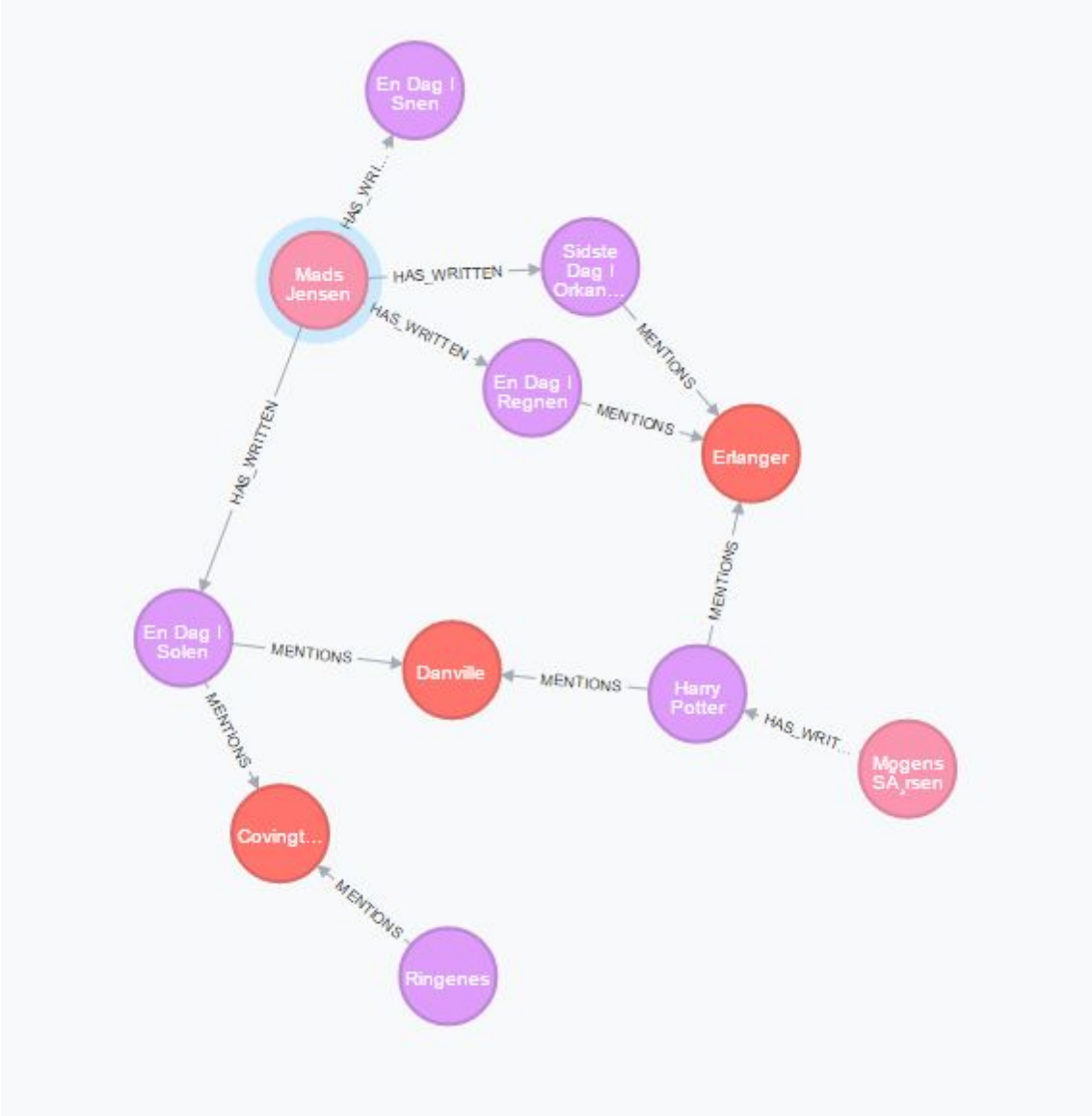
The data model Neo4J looks a lot like the model for MySQL, but since it is a graph database, it is still different.

We have 3 nodes: City, Book, and Author

We have 2 relations:

- Book-City
  - *MENTIONS*
- Book-Author
  - *HAS_WRITTEN*

Kasper P, Kasper O, Christian S, Oliver L
Database - Project - 2017



Kasper P, Kasper O, Christian S, Oliver L
Database - Project - 2017

# Measurements

## GetBooksByCity

### MySQL

Prepared statement:

```
SELECT books.*, authors.* FROM books JOIN book_author ON (book_author.bookId = books.id)
"+ " JOIN authors ON (authors.id = book_author.authorId) "+ " WHERE EXISTS (SELECT 1
FROM book_city JOIN cities ON (cities.id = book_city.cityId) WHERE book_city.bookId =
books.id AND cities.name = ?)
```

| elapsed | label | responseCode | responseMessage | success | bytes | URL |
|---|---|---|---|---|---|---|
| 842 | GET - BooksByCity | 200 | OK | true | 265702 | http://localhost:3000/softDevExam/api/mysql/city?city=Madrid |
| 885 | GET - BooksByCity | 200 | OK | true | 2537140 | http://localhost:3000/softDevExam/api/mysql/city?city=London |
| 605 | GET - BooksByCity | 200 | OK | true | 134156 | http://localhost:3000/softDevExam/api/mysql/city?city=Copenhagen |
| 573 | GET - BooksByCity | 200 | OK | true | 3013 | http://localhost:3000/softDevExam/api/mysql/city?city=Odense |
| 756 | GET - BooksByCity | 200 | OK | true | 1769883 | http://localhost:3000/softDevExam/api/mysql/city?city=Paris |

| | TID i Sekunder | Størrelse i MB |
|---|---|---|
| Gennemsnit | 0,7322 | 0,9419788 |
| MAX | 0,885 | 2,53714 |
| MIN | 0,573 | 0,003013 |

### Neo4J

```
MATCH (a:Author)-[r:HAS_WRITTEN]->(b:Book)-[m:MENTIONS]->(c:City)
WHERE (b)-[:MENTIONS]->(:City { name: $cityName })
RETURN b, c, a
```

| elapsed | label | responseCode | responseMessage | success | bytes | URL |
|---|---|---|---|---|---|---|
| 1136 | GET - BooksByCity | 200 | OK | true | 1770112 | http://localhost:3000/softDevExam/api/neo4j/city?city=Paris |
| 174 | GET - BooksByCity | 200 | OK | true | 265614 | http://localhost:3000/softDevExam/api/neo4j/city?city=Madrid |
| 576 | GET - BooksByCity | 200 | OK | true | 2538562 | http://localhost:3000/softDevExam/api/neo4j/city?city=London |
| 90 | GET - BooksByCity | 200 | OK | true | 134158 | http://localhost:3000/softDevExam/api/neo4j/city?city=Copenhagen |
| 67 | GET - BooksByCity | 200 | OK | true | 3013 | http://localhost:3000/softDevExam/api/neo4j/city?city=Odense |

| | TID i Sekunder | Størrelse i MB |
|---|---|---|
| Gennemsnit | 0,4086 | 0,9422918 |
| MAX | 1,136 | 2,538562 |
| MIN | 0,067 | 0,003013 |

The winner of the query is: MySQL with 323ms

# CitiesByBook

## MySQL

Prepared statement:

```
SELECT cities.name, X(cities.location) as longitude, Y(cities.location) as latitude FROM
books
JOIN book_city ON (book_city.bookId = books.id)
JOIN cities ON (cities.id = book_city.cityId) " + "WHERE books.name = ?
```

| elapsed | label | responseCode | responseMessage | success | bytes | URL |
|---|---|---|---|---|---|---|
| 486 | GET - CitiesByBook | 200 | OK | true | 3728 | http://localhost:3000/softDevExam/api/mysql/book?book=Legends%20of%20Vancouver |
| 44 | GET - CitiesByBook | 200 | OK | true | 435 | http://localhost:3000/softDevExam/api/mysql/book?book=The%20Tree%20of%20Life |
| 42 | GET - CitiesByBook | 200 | OK | true | 1019 | http://localhost:3000/softDevExam/api/mysql/book?book=Wildfire |
| 37 | GET - CitiesByBook | 200 | OK | true | 7614 | http://localhost:3000/softDevExam/api/mysql/book?book=The%20Knights%20Templars |
| 37 | GET - CitiesByBook | 200 | OK | true | 4847 | http://localhost:3000/softDevExam/api/mysql/book?book=Porto%20Rico |

### Result:

| | TID i Sekunder | Størrelse i MB |
|---|---|---|
| Gennemsnit | 0,1292 | 0,003529 |
| MAX | 0,486 | 0,007614 |
| MIN | 0,037 | 0,000435 |

## Neo4j

Parameters: $bookName

```
MATCH (b:Book {title: $bookName})-[m:MENTIONS]->(c:City)
RETURN c
```

| elapsed | label | responseCode | responseMessage | success | bytes | URL |
|---|---|---|---|---|---|---|
| 453 | GET - CitiesByBook | 200 | OK | true | 6641 | http://localhost:3000/softDevExam/api/neo4j/book?book=Porto%20Rico |
| 88 | GET - CitiesByBook | 200 | OK | true | 5122 | http://localhost:3000/softDevExam/api/neo4j/book?book=Legends%20of%20Vancouver |
| 86 | GET - CitiesByBook | 200 | OK | true | 551 | http://localhost:3000/softDevExam/api/neo4j/book?book=The%20Tree%20of%20Life |
| 88 | GET - CitiesByBook | 200 | OK | true | 1365 | http://localhost:3000/softDevExam/api/neo4j/book?book=Wildfire |
| 113 | GET - CitiesByBook | 200 | OK | true | 10492 | http://localhost:3000/softDevExam/api/neo4j/book?book=The%20Knights%20Templars |

| | TID i Sekunder | Størrelse i MB |
|---|---|---|
| Gennemsnit | 0,1656 | 0,0048342 |
| MAX | 0,453 | 0,010492 |
| MIN | 0,086 | 0,000551 |

The winner of the query is: MySQL with 36ms

# GetBooksAndCitysByAuthor

## MySQL

Prepared statement:

```
SELECT books.*, authors.*, cities.*, X(cities.location) as longitude, Y(cities.location)
as latitude FROM authors
JOIN book_author ON (book_author.authorId = authors.id)
JOIN books ON (books.id = book_author.bookId)
JOIN book_city ON (book_city.bookId = books.id)
JOIN cities ON (book_city.cityId = cities.id)
WHERE authors.name = ?
```

### Result:

| elapsed | label | responseCode | responseMessage | success | bytes | URL |
|---|---|---|---|---|---|---|
| 20 | GET - BooksAndCitysByAuthor | 200 | OK | true | 275 | http://localhost:3000/softDevExam/api/mysql/author?author=Charles%20McRae |
| 16 | GET - BooksAndCitysByAuthor | 200 | OK | true | 276 | http://localhost:3000/softDevExam/api/mysql/author?author=Louise%20Gerard |
| 22 | GET - BooksAndCitysByAuthor | 200 | OK | true | 384 | http://localhost:3000/softDevExam/api/mysql/author?author=Gail%20Hamilton |
| 16 | GET - BooksAndCitysByAuthor | 200 | OK | true | 300 | http://localhost:3000/softDevExam/api/mysql/author?author=Henry%20Vaughan |
| 17 | GET - BooksAndCitysByAuthor | 200 | OK | true | 774 | http://localhost:3000/softDevExam/api/mysql/author?author=Arthur%20Ransome |

| | TID i Sekunder | Størrelse i MB |
|---|---|---|
| Gennemsnit | 0,0182 | 0,0004018 |
| MAX | 0,022 | 0,000774 |

| | | |
|---|---|---|
| MIN | 0,016 | 0,000275 |

## Neo4j

Parameters: $authorName

```
MATCH (a:Author)-[r:HAS_WRITTEN]->(b:Book)-[m:MENTIONS]->(c:City)
WHERE a.name = $authorName
RETURN b, c, a
```

| elapsed | label | responseCode | responseMessage | success | bytes | URL |
|---|---|---|---|---|---|---|
| 81 | GET - BooksAndCitysByAuthor | 200 | OK | true | 10923 | http://localhost:3000/softDevExam/api/neo4j/author?author=Arthur%20Ransome |
| 83 | GET - BooksAndCitysByAuthor | 200 | OK | true | 2826 | http://localhost:3000/softDevExam/api/neo4j/author?author=Charles%20McRae |
| 66 | GET - BooksAndCitysByAuthor | 200 | OK | true | 2774 | http://localhost:3000/softDevExam/api/neo4j/author?author=Louise%20Gerard |
| 56 | GET - BooksAndCitysByAuthor | 200 | OK | true | 15933 | http://localhost:3000/softDevExam/api/neo4j/author?author=Gail%20Hamilton |
| 55 | GET - BooksAndCitysByAuthor | 200 | OK | true | 10122 | http://localhost:3000/softDevExam/api/neo4j/author?author=Henry%20Vaughan |

| | TID i Sekunder | Størrelse i MB |
|---|---|---|
| Gennemsnit | 0,0682 | 0,0085156 |
| MAX | 0,083 | 0,015933 |
| MIN | 0,055 | 0,002774 |

The winner of the query is: MySQL with 5ms

Kasper P, Kasper O, Christian S, Oliver L


# GetBooksByLocation

## MySQL

Prepared statement:

```
SELECT books.*, authors.*, cities.*, X(cities.location) as longitude, Y(cities.location)
as latitude FROM cities
JOIN book_city ON (book_city.cityId = cities.id)
JOIN books ON (books.id = book_city.bookId)
JOIN book_author ON (book_author.bookId = books.id)
JOIN authors ON (authors.id = book_author.authorId)
WHERE (3959 * OS(COS(RADIANS(longitude)) * COS(RADIANS(X(cities.location))) *
COS(RADIANS(Y(cities.location)) -RADIANS(latitude)) + SIN(RADIANS(longitude)) *
SIN(RADIANS(X(cities.location))))) < radiusInKilometers
```

### Result:

| elapsed | label | responseCode | responseMessage | success | bytes | URL |
|---|---|---|---|---|---|---|
| 6620 | GET - BooksByLocation | 200 | OK | true | 401 | http://localhost:3000/softDevExam/api/mysql/geo?latitude=10&longitude=10 |
| 6711 | GET - BooksByLocation | 200 | OK | true | 11705 | http://localhost:3000/softDevExam/api/mysql/geo?latitude=11&longitude=11 |
| 6555 | GET - BooksByLocation | 200 | OK | true | 38493 | http://localhost:3000/softDevExam/api/mysql/geo?latitude=55&longitude=12 |
| 7015 | GET - BooksByLocation | 200 | OK | true | 3329297 | http://localhost:3000/softDevExam/api/mysql/geo?latitude=40&longitude=-73 |
| 6511 | GET - BooksByLocation | 200 | OK | true | 16322 | http://localhost:3000/softDevExam/api/mysql/geo?latitude=40&longitude=-111 |

| | TID i Sekunder | Størrelse i MB |
|---|---|---|
| Gennemsnit | 6,6824 | 0,6792436 |
| MAX | 7,015 | 3,329297 |
| MIN | 6,511 | 0,000401 |

## Neo4j

Parameters: $latitude, $longtide

```
MATCH (a:Author)-[r:HAS_WRITTEN]->(b:Book)-[m:MENTIONS]->(c:City)
WHERE distance(point(c), point({ longitude: $longitude, latitude: $latitude
}))/1000 <= 50
RETURN b, c, a
```

| elapsed | label | responseCode | responseMessage | success | bytes | URL |
|---|---|---|---|---|---|---|
| 11876 | GET - BooksByLocation | 200 | OK | true | 155 | http://localhost:3000/softDevExam/api/neo4j/geo?latitude=40&longitude=-111 |
| 11764 | GET - BooksByLocation | 200 | OK | true | 374 | http://localhost:3000/softDevExam/api/neo4j/geo?latitude=10&longitude=10 |
| 11906 | GET - BooksByLocation | 200 | OK | true | 342 | http://localhost:3000/softDevExam/api/neo4j/geo?latitude=11&longitude=11 |
| 11460 | GET - BooksByLocation | 200 | OK | true | 155 | http://localhost:3000/softDevExam/api/neo4j/geo?latitude=55&longitude=12 |
| 11671 | GET - BooksByLocation | 200 | OK | true | 155 | http://localhost:3000/softDevExam/api/neo4j/geo?latitude=40&longitude=-73 |

| | TID i Sekunder | Størrelse i MB |
|---|---|---|
| Gennemsnit | 11,7354 | 0,0002362 |
| MAX | 11,906 | 0,000374 |
| MIN | 11,46 | 0,000155 |

The winner of the query is: MySQL with 5s and 530ms

# Recommendations

We will recommend Neo4J if you want to import the data very quickly, but for querying we think that MySQL is better than Neo4J.