Storage Systems Comparison between MySQL, MongoDB and Neo4j using Spark

Ana Daniel Federico Fiorini
University of Trento

The Data

The data was obtained from the Bureau of Transportation Statistics of the United States of America and it consists of information on every commercial flight made between 1987 and 2015 within the USA.

- 160+ million records
- Flights are grouped by month
- 338 files zip compressed csv's
- 827 MB compressed / 18 GB uncompressed

Problem

Relational DB vs Document DB vs Graph DB

MySQL vs MongoDB vs Neo4j

Which storage system that performs and suits best for the flights dataset?

How do we process 160+ million records?

Spark Aggregation

We used Apache Spark, an open source cluster computing framework, to aggregate the 160+ million records.

- It has a multi-stage in-memory **MapReduce** implementation
- Spark's Resilient Distributed Dataset (**RDD**) objects are collection of elements on which operations can be performed in a **parallel** way making processing **faster**.

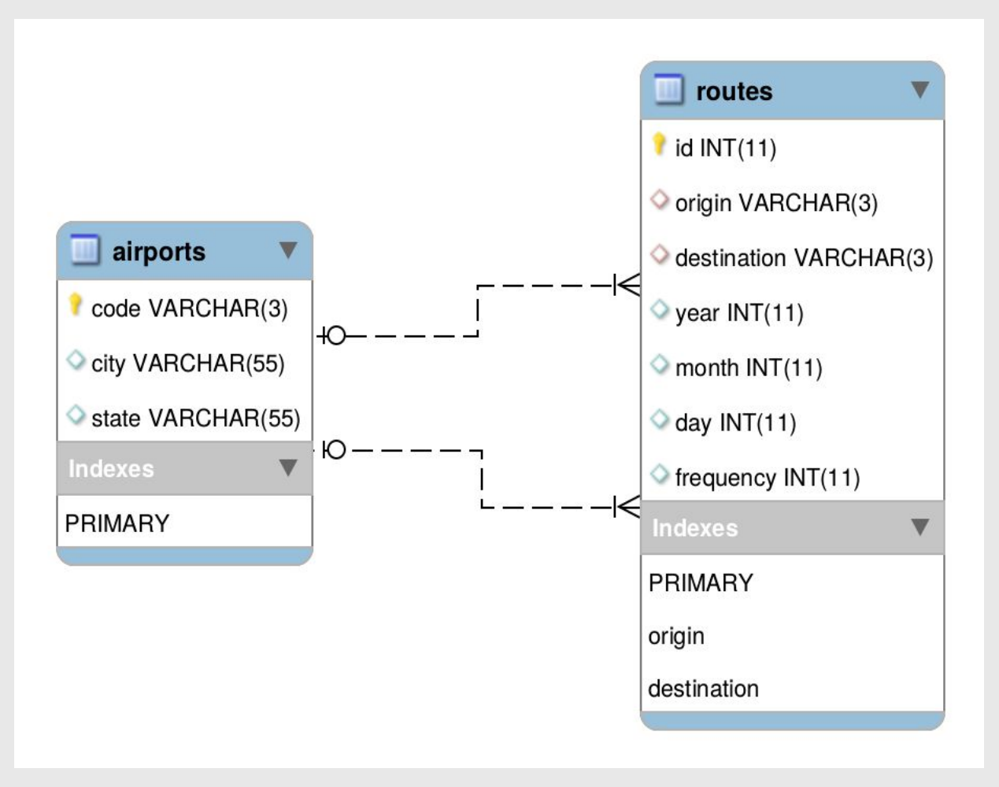
Aggregation steps:

1. Load uncompressed **csv** files, one at a time
2. Create **RDD**
3. Map and reduce to clean and obtain flight route **daily frequency**

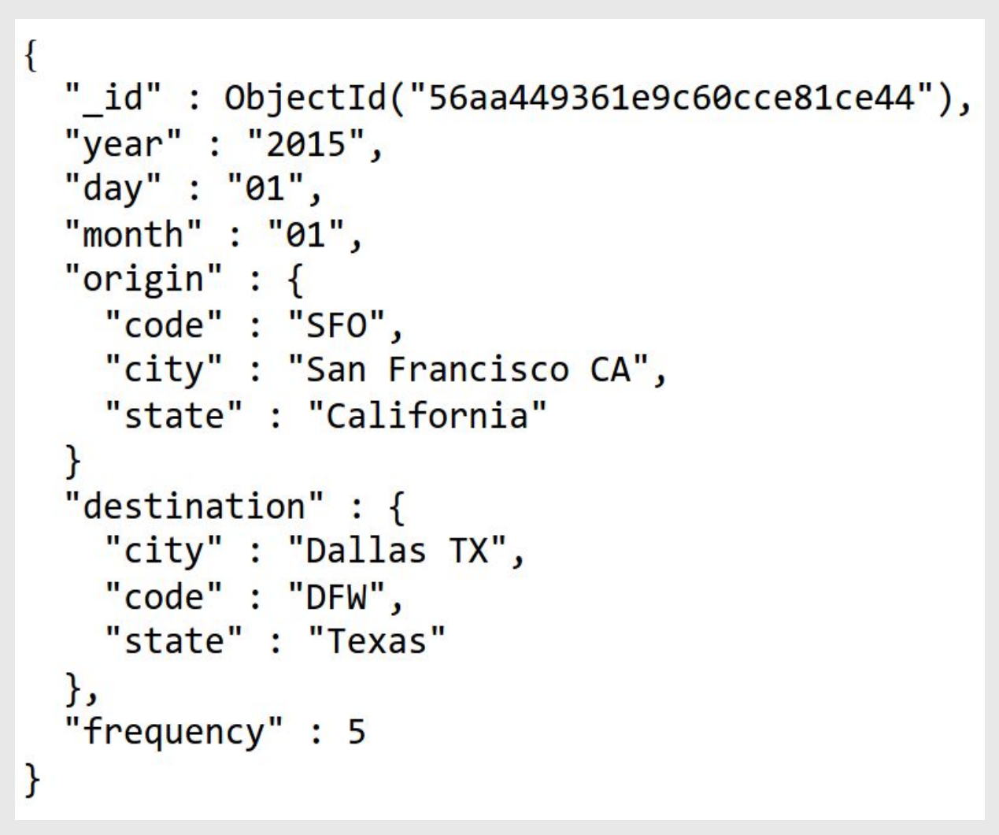
```
Map:                                    Reduce:

( (2015,01,01,DFW,JFK) , 1 )
( (2015,01,03,DFW,JFK) , 1 )     ( (2015,01,01,DFW,JFK) , 2 )
( (2015,01,01,DFW,JFK) , 1 ) → ( (2015,01,02,DFW,JFK) , 2 )
( (2015,01,02,DFW,JFK) , 1 )     ( (2015,01,03,DFW,JFK) , 1 )
( (2015,01,02,DFW,JFK) , 1 )
```

4. Export (MySQL, MongoSB or csv for Neo4J)

Data Structures

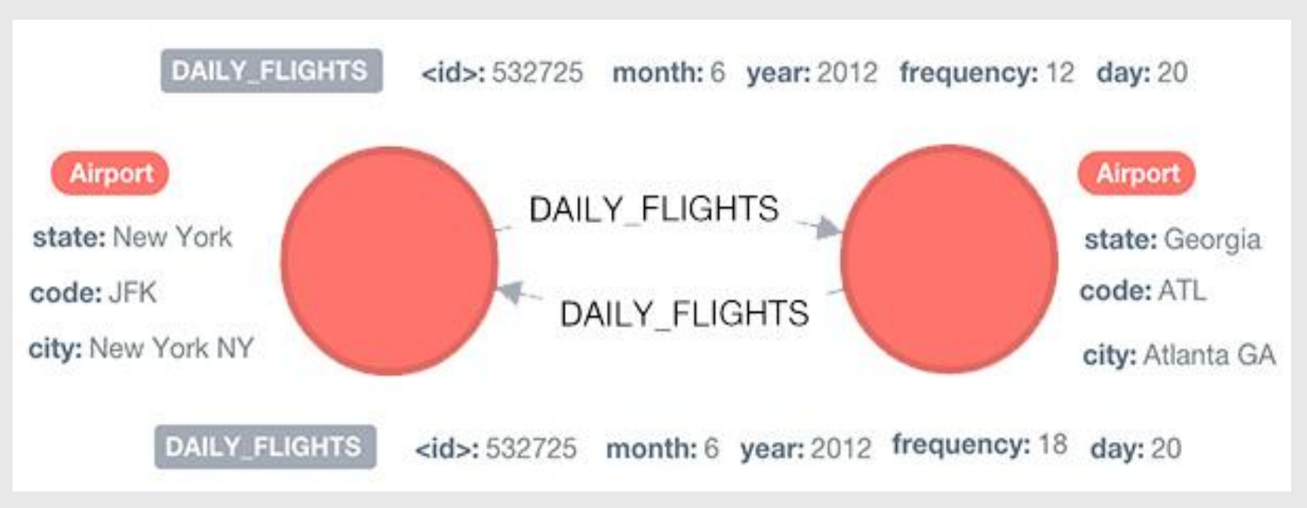


MySQL ERR Diagram



```
{
  "_id" : ObjectId("56aa449361e9c60cce81ce44"),
  "year" : "2015",
  "day" : "01",
  "month" : "01",
  "origin" : {
    "code" : "SFO",
    "city" : "San Francisco CA",
    "state" : "California"
  }
  "destination" : {
    "city" : "Dallas TX",
    "code" : "DFW",
    "state" : "Texas"
  },
  "frequency" : 5
}
```

MongoDB Document


DAILY_FLIGHTS <id>: 532725 month: 6 year: 2012 frequency: 12 day: 20
Airport
state: New York
code: JFK
city: New York NY
DAILY_FLIGHTS
DAILY_FLIGHTS
Airport
state: Georgia
code: ATL
city: Atlanta GA
DAILY_FLIGHTS <id>: 532725 month: 6 year: 2012 frequency: 18 day: 20

Neo4j Nodes and Relationships

Spark Integration

- Spark + MySQL: YES

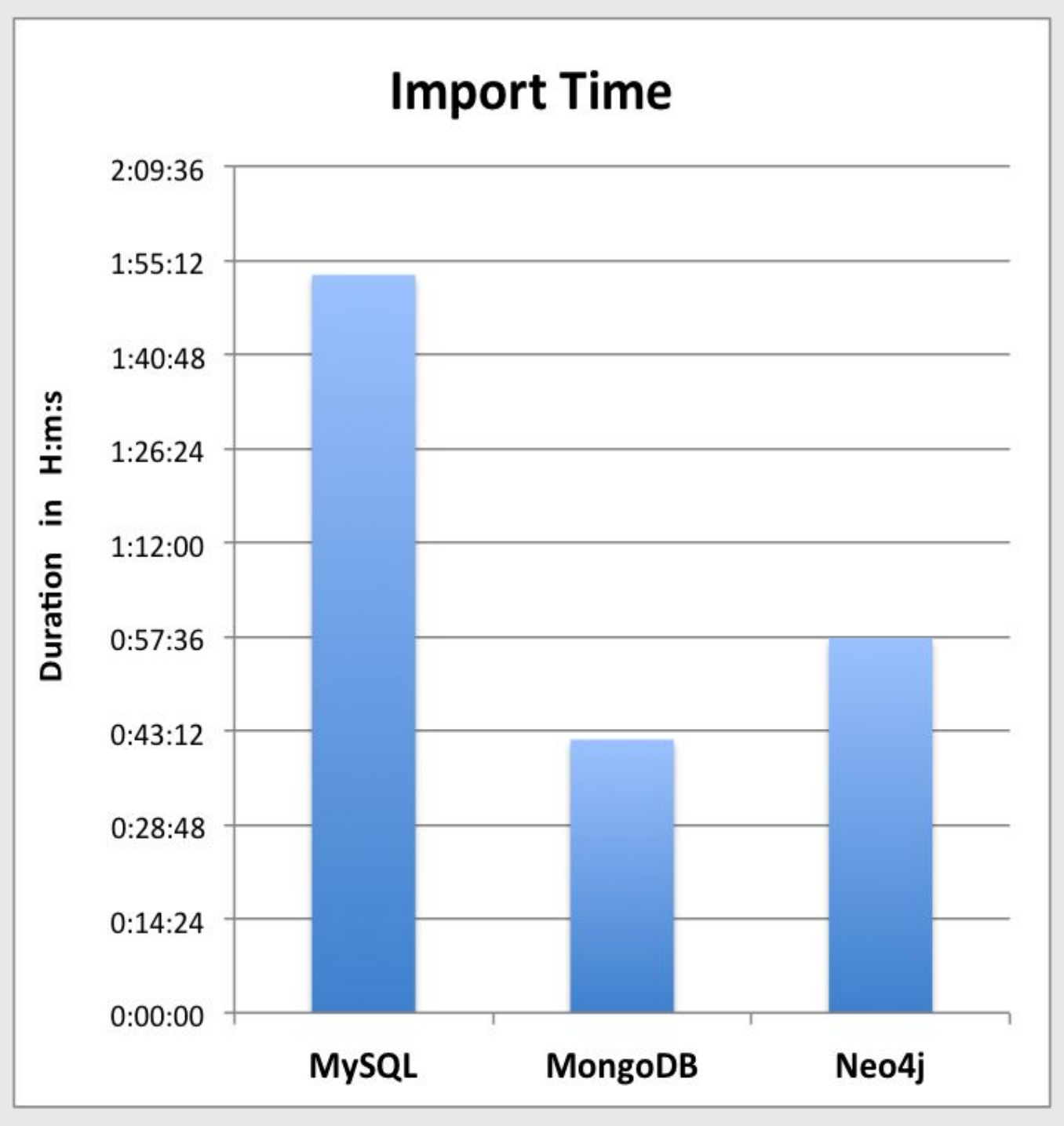
It comes (almost) 'out-of-the-box', it only needs the JDBC connector for MySQL.

- Spark + MongoDB: YES

Integration is done via the Mongo Hadoop Connector.

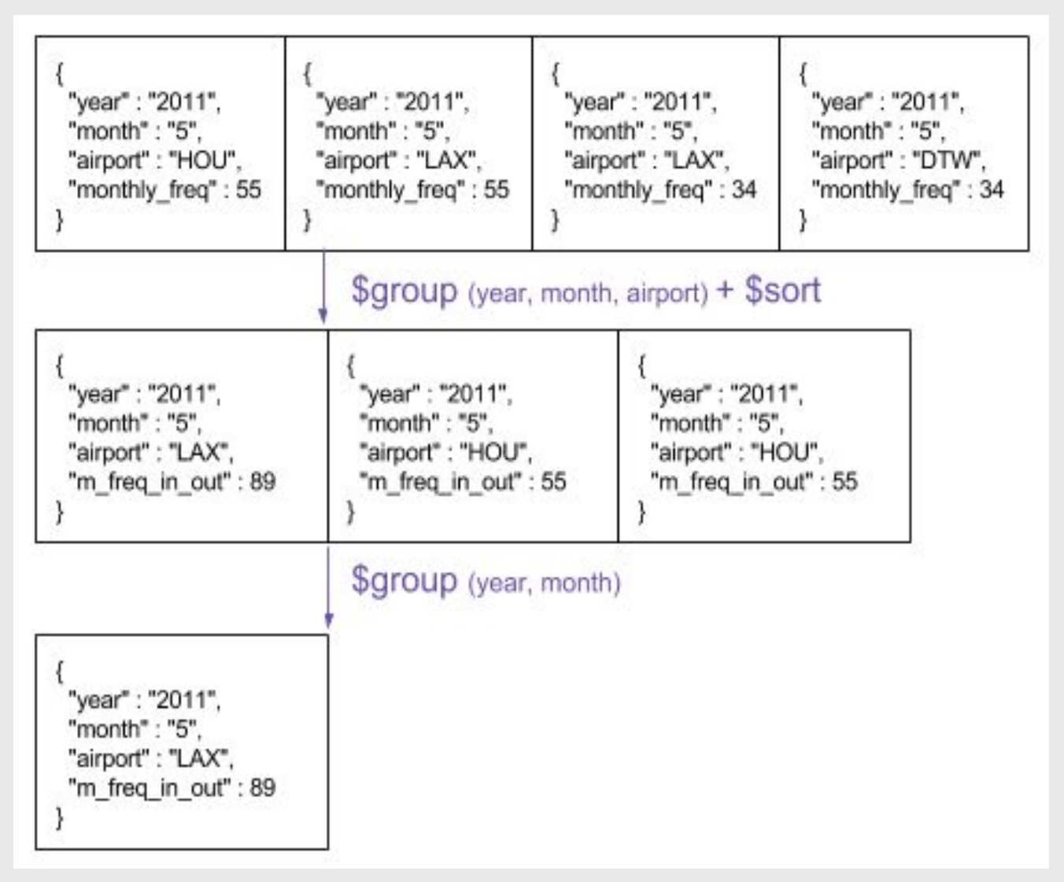
- Spark + Neo4j: N/A.

Spark is only used to prepare the csv files that Neo4j imports, no actual integration is done. In this case, import time refers to the time spent pre-aggregating with Spark + the time spent executing the import command.



NoSQL Query Structure

MongoDB: The Aggregation Pipeline
In the pipeline documents enter several stages and during each stage they are transformed to be consumed by the following stage.

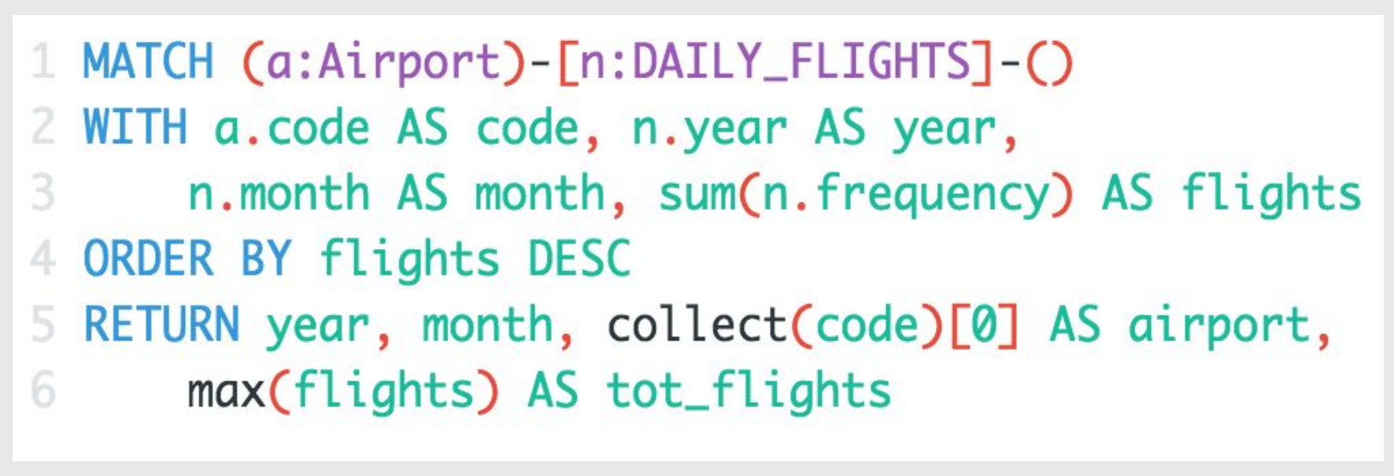


```
{
 "year" : "2011",
 "month" : "5",
 "airport" : "HOU",
 "monthly_freq" : 55
}
{
 "year" : "2011",
 "month" : "5",
 "airport" : "LAX",
 "monthly_freq" : 55
}
{
 "year" : "2011",
 "month" : "5",
 "airport" : "LAX",
 "monthly_freq" : 34
}
{
 "year" : "2011",
 "month" : "5",
 "airport" : "DTW",
 "monthly_freq" : 34
}
```

```
{
 "year" : "2011",
 "month" : "5",
 "airport" : "LAX",
 "m_freq_in_out" : 89
}
{
 "year" : "2011",
 "month" : "5",
 "airport" : "HOU",
 "m_freq_in_out" : 55
}
{
 "year" : "2011",
 "month" : "5",
 "airport" : "HOU",
 "m_freq_in_out" : 55
}
```

```
{
 "year" : "2011",
 "month" : "5",
 "airport" : "LAX",
 "m_freq_in_out" : 89
}
```

Neo4j Query Language: Cypher
Cypher focuses only on what to select from the database and not so much about how to do it.

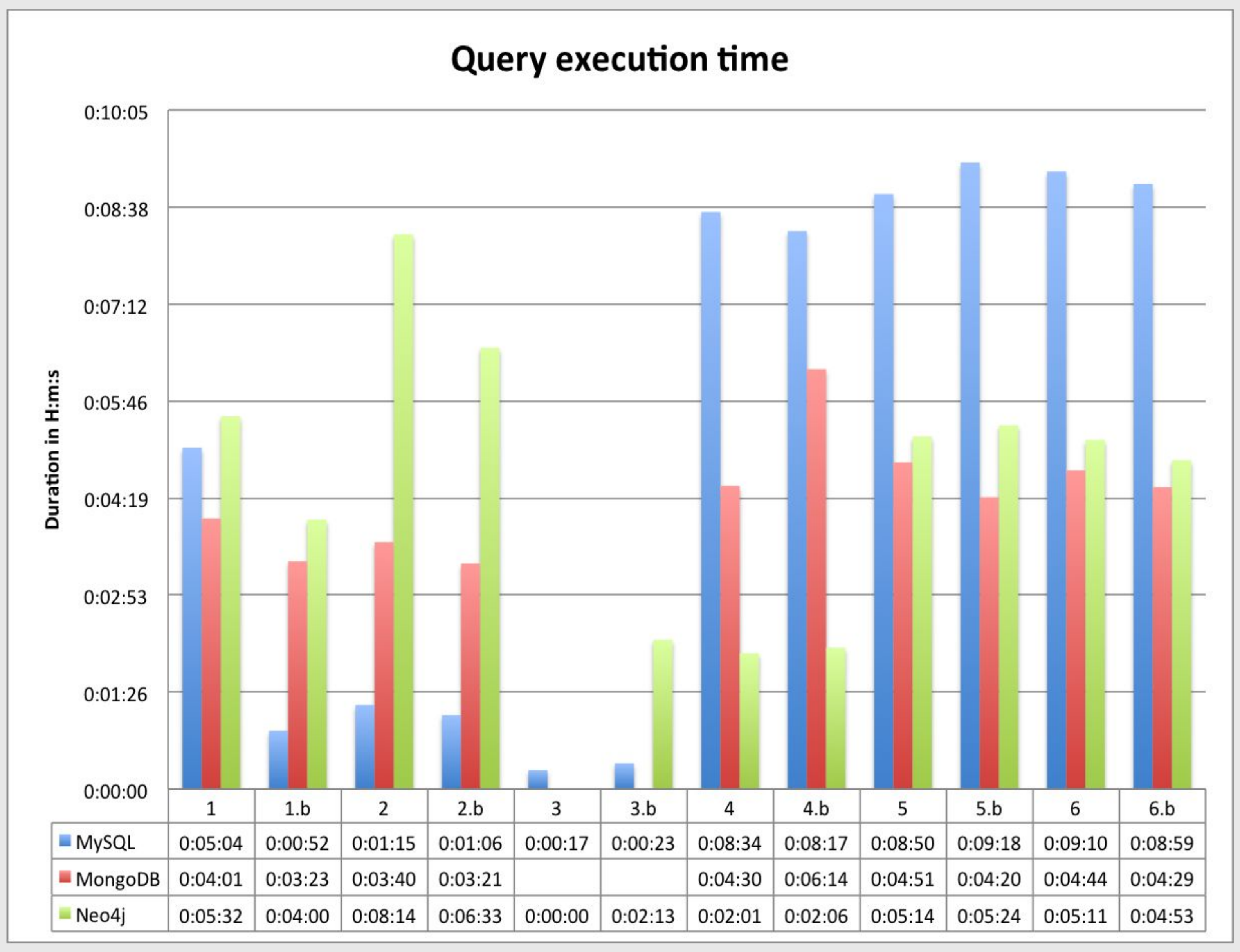


```
MATCH (a:Airport)-[n:DAILY_FLIGHTS]-()
WITH a.code AS code, n.year AS year,
    n.month AS month, sum(n.frequency) AS flights
ORDER BY flights DESC
RETURN year, month, collect(code)[0] AS airport,
    max(flights) AS tot_flights
```

Acknowledgements

Even if the writing performance was the worst we considered that the fact that MySQL support comes almost "out-of-the-box" with Spark.
Such a bad performance was not expected for aggregations done with Neo4j, however if a more thorough aggregation is done with a tool like Spark it can be very promising for this kind of data.
We needed data that required a more complex structuration in order to benefit from the flexibility brought by MongoDB document based records.

Querying



	1	1.b	2	2.b	3	3.b	4	4.b	5	5.b	6	6.b
🟦 MySQL	0:05:04	0:00:52	0:01:15	0:01:06	0:00:17	0:00:23	0:08:34	0:08:17	0:08:50	0:09:18	0:09:10	0:08:59
🟥 MongoDB	0:04:01	0:03:23	0:03:40	0:03:21			0:04:30	0:06:14	0:04:51	0:04:20	0:04:44	0:04:29
🟩 Neo4j	0:05:32	0:04:00	0:08:14	0:06:33	0:00:00	0:02:13	0:02:01	0:02:06	0:05:14	0:05:24	0:05:11	0:04:53

Query description

1. Find the most frequent route per month
   1.b Per year
2. Find the airport with more flights (in and out) per month
   2.b Per year
3. Given a departure date find the **shortest path** from airport A to airport B
   3.b. Consider only high frequency routes
4. Find the state with more internal flights per month
   4.b Per year
5. Find the state with more external departure flights per month
   5.b Per year
6. Find the state with more external arrival flights per month
   6.b Per year

Insights

- MongoDB does better in queries like Q5 where the state of teh airport is referenced
- Without JOINs, MySQL performs much better than MongoDB (Q2 vs Q5)
- MongoDB's aggregation pipeline is not as fast when there are several steps of aggregation and manipulation of the objects
- Neo4j was capable to execute the shortest path query (Q3) in less than 1 second but when it require some aggregation (Q3.b) it performed worse than the MySQL implementation.