Data Management Project

Option 2: NoSQL tool vs relational DBMS

vs

Lorenzo Russo 2091186

Introduction

Once a dataset was chosen, the main purpose of this work was to compare the performance between a NoSQL tool and a relational DBMS.
Neo4j, which uses graph database, was used as the NoSQL tool; instead PostgreSQL was used as the relational DBMS.

To analyze the performance, several queries were run on both systems. In particular, on Neo4j Cipher is used to write queries; instead in PostgreSQL SQL is used.

Finally, the results of the analyzes were compared, highlighting the advantages and disadvantages of each approach in terms of efficiency.

2


PostgreSQL
VS
neo4j
Dataset

Dataset: 2015 Flight Delays and Cancellations

The chosen dataset contains all the flights carried out in 2015 in the United States, in particular for each flight there are any delays and cancellations, and the reasons for these delays or cancellations.
There are 3 files in this dataset .csv:

- flights.csv: 5.82 million flights (31 columns)
- airports.csv: 322 airports (7 columns)
- airlines.csv: 14 airlines (2 columns)

airlines.csv (359 B)

Detail Compact Column 2 of 2 columns

About this file

IATA airline codes and names

IATA_CODE Airline Identifier	AIRLINE Airport's Name
14 unique values	14 unique values
UA	United Air Lines Inc.
AA	American Airlines Inc.
US	US Airways Inc.
F9	Frontier Airlines Inc.
B6	JetBlue Airways
OO	Skywest Airlines Inc.

flights.csv (592.41 MB)

Detail Compact Column 10 of 31 columns

About this file

IATA airline codes and names

YEAR Year of the Flight Trip	MONTH Month of the Flight Trip	DAY Day of the Flight Trip	DAY_OF_WEEK Day of week of the Flight Trip	AIRLINE Airline Identifier	FLIGHT_NUMBER Flight Identifier	TAIL_NUMBER Aircraft Identifier	ORIGIN_AIRPORT Starting Airport	DESTINATION_A… Destination Airport
2015 – 2015	1 – 12	1 – 31	1 – 7	WN 22% DL 15% Other (3681343) 63%	986.40 - 1183.48 Count: 300,506 1 – 9855	4898 unique values	ATL 6% ORD 5% Other (5186359) 89%	ATL ORD Other (5186269)
2015	1	1	4	AS	98	N407AS	ANC	SEA
2015	1	1	4	AA	2336	N3KUAA	LAX	PBI
2015	1	1	4	US	840	N171US	SFO	CLT
2015	1	1	4	AA	258	N3HYAA	LAX	MIA
2015	1	1	4	AS	135	N527AS	SEA	ANC
2015	1	1	4	DL	806	N3730B	SFO	MSP
2015	1	1	4	NK	612	N635NK	LAS	MSP
2015	1	1	4	US	2013	N584UW	LAX	CLT
2015	1	1	4	AA	1112	N3LAAA	SFO	DFW
2015	1	1	4	DL	1173	N826DN	LAS	ATL
2015	1	1	4	DL	2336	N958DN	DEN	ATL
2015	1	1	4	AA	1674	N853AA	LAS	MIA
2015	1	1	4	DL	1434	N547US	LAX	MSP
2015	1	1	4	DL	2324	N3751B	SLC	ATL
2015	1	1	4	DL	2440	N651DL	SEA	MSP

airports.csv (23.87 kB)

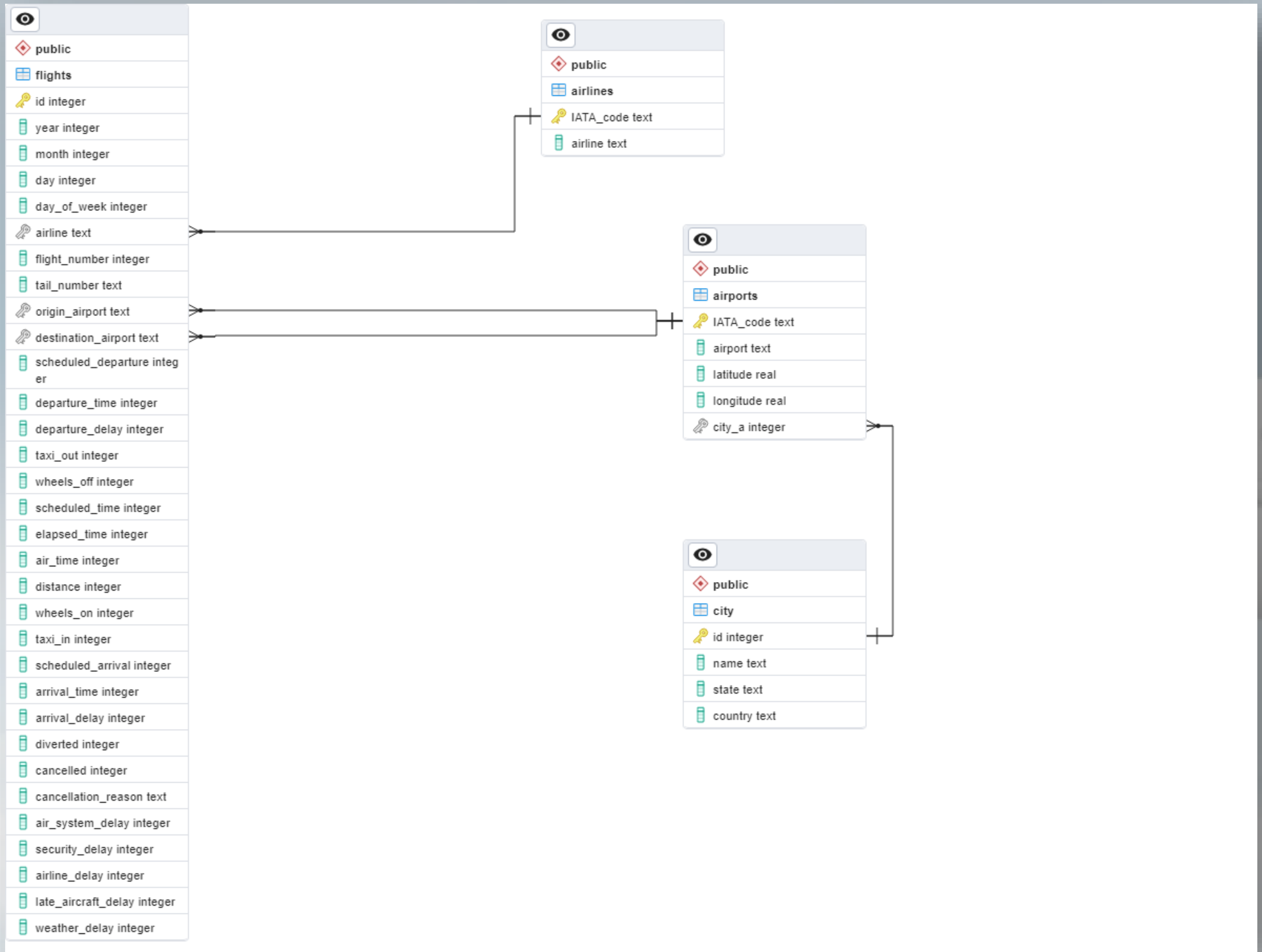
Detail Compact Column

About this file

IATA airport codes and names

IATA_CODE Location Identifier	AIRPORT Airport's Name	CITY Atlanta	STATE Georgia	COUNTRY Country Name of the Airport	LATITUDE Latitude of the Airport	LONGITUDE Longitude of the Airport
322 unique values	322 unique values	308 unique values	TX 7% CA 7% Other (276) 86%	1 unique value	13.5 – 71.3	-177 – -64.8
ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447
ACK	Nantucket Memorial Airport	Nantucket	MA	USA	41.25305	-70.06018
ACT	Waco Regional Airport	Waco	TX	USA	31.61129	-97.23052
ACV	Arcata Airport	Arcata/Eureka	CA	USA	40.97812	-124.10862
ACY	Atlantic City International Airport	Atlantic City	NJ	USA	39.45758	-74.57717

ER Diagram



The ER Diagram has 4 tables (entities): flights, airlines, airports and city.

There are 4 relationship:

- flights and airlines: means that each flight has an airline that manage it (One-To-Many Relationship)
- flights and airports: means that each flight has an origin airport (One-To-Many Relationship)
- flights and airports: means that each flight has a destination airport (One-To-Many Relationship)
- airports and city: means that each airport is in a city (One-To-Many Relationship)

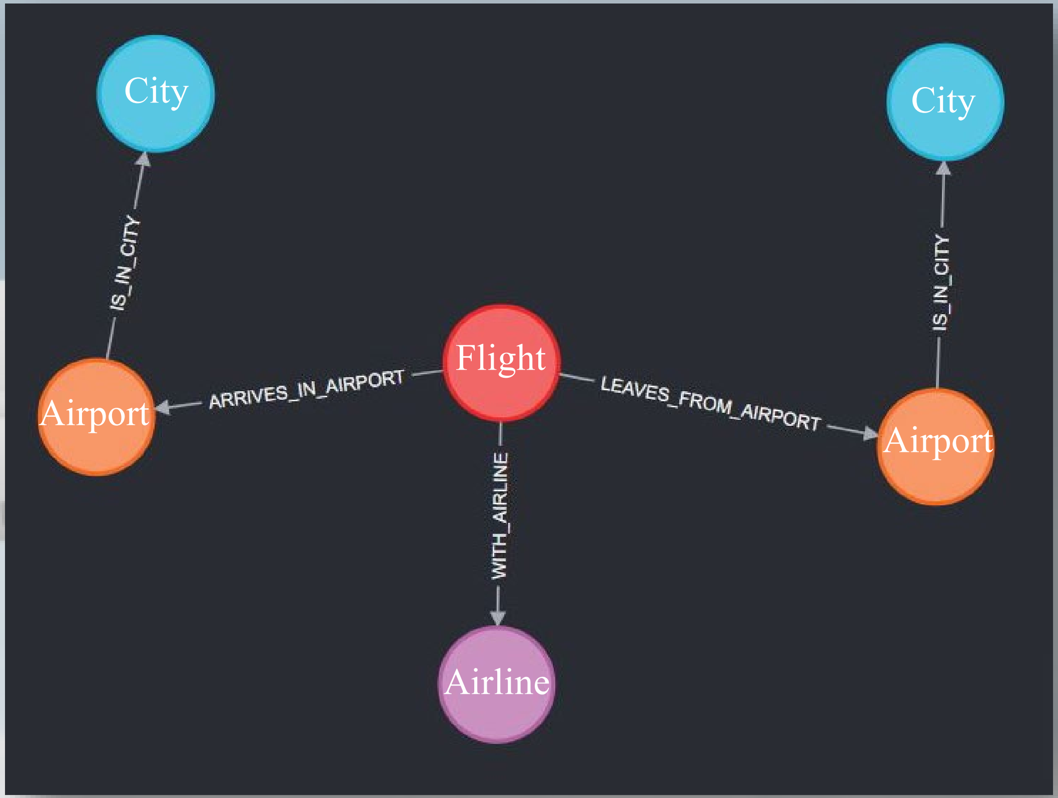
Graph Representation

In our graph database schema, entities are represent as nodes, and relationship as directed edges.

Flight, airport, airline and city are the nodes.

Edges are:

- ARRIVES_IN_AIRPORT: from flight to airport
- LEAVES_FROM_AIRPORT: from flight to airport
- IS_IN_CITY: from airport to city
- WITH_AIRLINE: from flight to airline



Query Analysis: PostgreSQL vs Neo4j

PostgreSQL vs Neo4j

Table/Object	Rows in PostgreSQL	Object in Neo4j
Flight	5819079	5819079 nodes
Airport	322	322 nodes
Airline	14	14 nodes
City	318	318 nodes
ARRIVES_IN_AIRPORT	-	5332914 edges
LEAVES_FROM_AIRPORT	-	5332914 edges
IS_IN_CITY	-	322 edges
WITH_AIRLINE	-	5819073 edges

7 different queries were written in both SQL and Cipher.
SQL queries were executed in PostgreSQL and average execution times were analyzed.
Cipher queries were executed in Neo4j and average execution times were analyzed.
The results obtained on both systems were compared to highlight the advantages and disadvantages.

Query 1

Find the top 10 airlines that have the most canceled flights in relation to the number of flights.

SQL

```
WITH cancflightcount AS (
        SELECT airlines.airline, COUNT(flights.cancelled) AS totcancelledflights
        FROM airlines JOIN flights ON airlines."IATA_code" = flights.airline
        WHERE flights.cancelled = 1
        GROUP BY airlines.airline
),
totalflightscount AS (
        SELECT airlines.airline, COUNT(*) AS totflights
        FROM airlines JOIN flights ON airlines."IATA_code" = flights.airline
        GROUP BY airlines.airline
)
SELECT tfc.airline, tfc.totflights, cfc.totcancelledflights,
(cfc.totcancelledflights / 1.0)/ tfc.totflights AS ratioCancTotFlights
FROM totalflightscount AS tfc JOIN cancflightcount AS cfc ON tfc.airline = cfc.airline
ORDER BY ratioCancTotFlights DESC
LIMIT 10;
```

Cipher

```
MATCH (a:Airline)←[:WITH_AIRLINE]-(f:Flight {cancelled: 1})
WITH a, COUNT(f) AS totcancelledflights
MATCH (a)←[:WITH_AIRLINE]-(f)
WITH a, COUNT(f) AS totflights, totcancelledflights
RETURN a.airlineName, totflights, totcancelledflights,
((totcancelledflights * 1.0) / totflights) AS ratioCancTotFlights
ORDER BY ratioCancTotFlights DESC
LIMIT 10;
```

The query returned 10 rows.

PostgreSQL
The query execution time with clean cache: 3146 ms
The avg time to execute the query: 3063 ms

Neo4j
The query execution time with clean cache: 202512 ms
The avg time to execute the query: 35556 ms

Query 2

Find the top 10 cities with the most canceled flights that departed from it.

SQL

```
SELECT ci.name, COUNT(*) AS totcancelled
FROM flights AS f, airports AS ap, city AS ci
WHERE f."origin_airport" = ap."IATA_code" AND ap."city_a" = ci.id AND
f.cancelled = 1
GROUP BY ci.name
ORDER BY totcancelled DESC
LIMIT 10;
```

Cipher

```
MATCH (c:City)←[:IS_IN_CITY]-(a:Airport)<-[:LEAVES_FROM_AIRPORT]-
(f:Flight {cancelled: 1})
RETURN c.name, COUNT(f) AS cancelledFlights
ORDER BY cancelledFlights DESC
LIMIT 10;
```

The query returned 10 rows.

PostgreSQL
The query execution time with clean cache: 1420 ms
The avg time to execute the query: 1279 ms

Neo4j
The query execution time with clean cache: 274367 ms
The avg time to execute the query: 52688 ms

10

Query 3

Find the airline that operates the most flights departing from a specific city.

SQL

```
SELECT al.airline, COUNT(*) AS flightCount
FROM flights AS f, airports AS ap, city AS ci, airlines AS al
WHERE ap."city_a" = ci.id AND ap."IATA_code" = f."origin_airport" AND f.airline =
al."IATA_code"
AND ci.name = 'New York'
GROUP BY al.airline
ORDER BY flightCount DESC
LIMIT 1;
```

Cipher

```
MATCH (c:City {name: "New York"})←[:IS_IN_CITY]-
(aa:Airport)←[:LEAVES_FROM_AIRPORT]-(f:Flight)-[:WITH_AIRLINE]→(al:Airline)
WITH al.airlineName AS airlineN, COUNT(f) AS flightCount
RETURN airlineN, flightCount
ORDER BY flightCount DESC
LIMIT 1;
```

The query returned 1 rows.

PostgreSQL
The query execution time with clean cache: 1369 ms
The avg time to execute the query: 1264 ms

Neo4j
The query execution time with clean cache: 11677 ms
The avg time to execute the query: 1449 ms

Query 4

For each airport count the number of flights departing from it and the number of flights arriving at it.

SQL

```sql
SELECT ar.airport, count(f1.id) AS origin, tb.destination
FROM  airports ar, flights f1,
(SELECT ar.airport as an, count(f2.id) AS destination FROM airports ar, flights f2
WHERE ar."IATA_code" = f2.destination_airport group by ar.airport) AS tb
WHERE ar."IATA_code" = f1.origin_airport and ar.airport = tb.an
GROUP BY ar.airport, tb.destination
ORDER BY ar.airport;
```

Cipher

```cypher
MATCH (ar:Airport)
OPTIONAL MATCH (ar)<-[:LEAVES_FROM_AIRPORT]-(f1:Flight)
WITH ar, COUNT(f1) AS origin
OPTIONAL MATCH (ar)<-[:ARRIVES_IN_AIRPORT]-(f2:Flight)
WITH ar, origin, COUNT(f2) AS destination
RETURN ar.airportName AS Airport, origin, destination
ORDER BY Airport;
```

The query returned 322 rows.

PostgreSQL
The query execution time with clean cache: 7911 ms
The avg time to execute the query: 7205 ms

Neo4j
The query execution time with clean cache: 8721 ms
The avg time to execute the query: 5273 ms

Query 5

Find airlines that work for the New York-Los Angeles route, but not for the Atlanta-Albany route.

SQL

```sql
SELECT DISTINCT a1.airline
FROM flights AS f1
JOIN airports AS ap1 ON f1.origin_airport = ap1."IATA_code"
JOIN airports AS ap2 ON f1.destination_airport = ap2."IATA_code"
JOIN city AS c1 ON ap1.city_a = c1.id
JOIN city AS c2 ON ap2.city_a = c2.id
JOIN airlines AS a1 ON f1.airline = a1."IATA_code"
WHERE (c1.name = 'New York' AND c2.name = 'Los Angeles')
AND NOT EXISTS (
   SELECT 1
   FROM flights AS f2
   JOIN airports AS ap3 ON f2.origin_airport = ap3."IATA_code"
   JOIN airports AS ap4 ON f2.destination_airport = ap4."IATA_code"
   JOIN city AS c3 ON ap3.city_a = c3.id
   JOIN city AS c4 ON ap4.city_a = c4.id
   WHERE (c3.name = 'Atlanta' AND c4.name = 'Albany')
   AND f1.airline = f2.airline );
```

Cipher

```
MATCH (al:Airline)<-[:WITH_AIRLINE]-(f:Flight)-[:LEAVES_FROM_AIRPORT]->(:Airport)-[:IS_IN_CITY]->(cityA:City {name: 'New York'})
MATCH (f)-[:ARRIVES_IN_AIRPORT]->(:Airport)-[:IS_IN_CITY]->(cityB:City {name: 'Los Angeles'})
WITH COLLECT(al) AS airlinesAB

MATCH (al:Airline)<-[:WITH_AIRLINE]-(f:Flight)-[:LEAVES_FROM_AIRPORT]->(:Airport)-[:IS_IN_CITY]->(cityC:City {name: 'Atlanta'})
MATCH (f)-[:ARRIVES_IN_AIRPORT]->(:Airport)-[:IS_IN_CITY]->(cityD:City {name: 'Albany'})
WITH airlinesAB, COLLECT(al) AS airlinesCD

WITH [al IN airlinesAB WHERE NOT al IN airlinesCD | al.airlineName] AS AirlinesBetweenABNotBetweenCD
UNWIND AirlinesBetweenABNotBetweenCD AS airlineName
RETURN DISTINCT airlineName
```

The query returned 4 rows.

PostgreSQL
The query execution time with clean cache: 3699 ms
The avg time to execute the query: 3504 ms

Neo4j
The query execution time with clean cache: 3580 ms
The avg time to execute the query: 1368 ms

Query 6

For each route it prints the number of airlines operating on that route and which ones they are.

SQL

```
SELECT
   a1.airport,
   a2.airport,
   COUNT(DISTINCT f.airline) AS NumAirlines,
                  STRING_AGG(DISTINCT f.airline::text, ', ') AS Airlines
FROM flights AS f, airports a1, airports a2
WHERE f.origin_airport = a1."IATA_code" AND f.destination_airport = a2."IATA_code"
GROUP BY a1.airport, a2.airport;
```

Cipher

```
MATCH (f:Flight)
MATCH (a1:Airport)←[:LEAVES_FROM_AIRPORT]-(f)-[:ARRIVES_IN_AIRPORT]-
>(a2:Airport)
RETURN a1.airportName AS originAirport, a2.airportName AS destinationAirport,
COUNT(DISTINCT f.airline) AS NumAirlines, COLLECT(DISTINCT f.airline) AS
Airlines;
```

The query returned 4693 rows.

PostgreSQL
The query execution time with clean cache: 25680 ms
The avg time to execute the query: 25493 ms

Neo4j
The query execution time with clean cache: 40 ms
The avg time to execute the query: 5 ms

14

Query 7

For each airlines find the number of departure delay flights and the number of flights with departure delay that arrive in time.

SQL

```
WITH delayflightcount AS (
        SELECT al.airline, COUNT(*) AS tot_departude_delay_flights
        FROM flights f, airlines al
        WHERE f.airline = al."IATA_code" AND f.departure_delay > 0
        GROUP BY al.airline
),
delayRflightscount AS (
        SELECT al.airline, COUNT(*) AS
tot_departude_delay_flights_that_arrive_intime
        FROM flights f, airlines al
        WHERE f.airline = al."IATA_code" AND f.departure_delay > 0 AND
f.arrival_time = f.scheduled_arrival
        GROUP BY al.airline
)
SELECT dfc.airline, dfc.tot_departude_delay_flights,
dRfc.tot_departude_delay_flights_that_arrive_intime
FROM delayflightcount AS dfc JOIN delayRflightscount AS dRfc ON dfc.airline =
dRfc.airline
ORDER BY dfc.tot_departude_delay_flights DESC;
```

Cipher

```
MATCH (a:Airline)<-[:WITH_AIRLINE]-(f:Flight)
WHERE f.departureDelay > 0
WITH a, COUNT(f) AS tot_departude_delay_flights
MATCH (a)<-[:WITH_AIRLINE]-(f)
WHERE f.departureDelay > 0 AND f.arrivalTime = f.scheduledArrival
WITH a, COUNT(f) AS tot_departude_delay_flights_that_arrive_intime,
tot_departude_delay_flights
RETURN a.airlineName, tot_departude_delay_flights,
tot_departude_delay_flights_that_arrive_intime
ORDER BY tot_departude_delay_flights DESC;
```

The query returned 14 rows.

PostgreSQL
The query execution time with clean cache: 2858 ms
The avg time to execute the query: 2613 ms

Neo4j
The query execution time with clean cache: 249525 ms
The avg time to execute the query: 64783 ms

15

PostgreSQL

VS

neo4j

Final results and Conclusions

Final results and Conclusions

Query performance summary (avg execution time)

	PostgreSQL	Neo4j
Query 1	3063 ms	35556 ms
Query 2	1279 ms	52688 ms
Query 3	1264 ms	1449 ms
Query 4	7205 ms	5273 ms
Query 5	3504 ms	1368 ms
Query 6	25493 ms	5 ms
Query 7	2613 ms	64783 ms

When you need to execute queries that need to filter data based on flight attributes, PostgreSQL has significantly better performance than Neo4j; this is because the flights have 5.8 million rows so the search for flights that respects a certain condition is much more efficient on the tables than on the nodes, also because each flight represents an independent node.
However, performance was better in Neo4j than in PostgreSQL when queries based on relationships were executed, this is because in SQL these types of queries must use many JOINs which slow down execution.

There have been pros and cons to both PostgreSQL and Neo4j, and these depend on the type of use we want to make of the data and what types of queries are performed. For the dataset used for this analysis, it is more efficient to use a relational database because this dataset adapts very well to the tabular structure, there are very few relationships between the entities and the typical queries are filtering between flights, analyzing the various delays and cancellations, and these types of queries perform better in a relational database.

Therefore, the use of a graph database compared to a relational database strictly depends on the type of data and the use you want to make.

17

THANK YOU FOR THE ATTENTION

PostgreSQL

VS

neo4j