Data Management Project

Master's Degree in Engineering in Computer Science

A.Y. 2023/2024

Fosci Giuseppe - 1855832
Cardinali Marco - 1797403

Prof. Maurizio Lenzerini
Prof. Roberto Maria Delfino

SAPIENZA
UNIVERSITÀ DI ROMA

A Comparative Analysis of Efficiency and Simplicity in Managing complex data:

Graph Database (NOSQL) vs Relational Database (SQL)

Table of content

Introduction

This presentation talk about the performance and ease of query writing when comparing Neo4j, a graph database, to PostgreSQL, an SQL-based relational database.

The goal is to run queries on both systems to compare response times, query complexity, and overall efficiency in data processing.

Table of content

Dataset description

The comparison uses a real-world dataset on traffic incidents in Rome, over two years (about 5.000 tuples/month).

This dataset provides information on road accidents, including environmental conditions, precise location, accident dynamics, involved vehicles, and the status of individuals.

Each record is linked to a specific accident and outlines the surrounding environment, the nature of the collision, and its outcomes, such as injuries, uninjured individuals, or fatalities, and other relevant attributes for road safety analysis.

Go back on table of content

Load CSV and query the BDs using the scripts

We created a fully automated system to load datasets and model them using Python scripts. The script can take all the datasets in the folder and import them into the correct databases.

We created a runner.py that is an interactive menù that allows you to choose which script to run.

We also created two scripts query_with_connection and query_static :
The first one is able to connect to database and execute in autonomous way the connection to DB, perform query and draw the performance chart using matplot library.
The second one draw the performance chart using plotty library.

All scripts are available in this link:
https://github.com/GiuseppeFosci/Data-Management-Project

Load csv files in the system

Execution of loading_csv.py

```
Processing dataset: ./Datasets/2021\csv_incidentiGennaio.csv
Database 'version2' already exists.
Processing incidents: 100%|                    | 4610/4610 [02:47<00:00, 27.52incident/s]
Processing dataset: ./Datasets/2021\csv_incidentiFebbraio.csv
Database 'version2' already exists.
Processing incidents: 100%|                    | 5073/5073 [04:40<00:00, 18.07incident/s]
Processing dataset: ./Datasets/2021\csv_incidentiMarzo.csv
Database 'version2' already exists.
Processing incidents: 100%|                    | 4538/4538 [05:32<00:00, 13.64incident/s]
Processing dataset: ./Datasets/2021\csv_incidentiAprile.csv
Database 'version2' already exists.
Processing incidents: 100%|                    | 5274/5274 [07:58<00:00, 11.02incident/s]
Processing dataset: ./Datasets/2021\csv_incidentiMaggio.csv
Database 'version2' already exists.
Processing incidents: 100%|                    | 6467/6467 [12:02<00:00,  8.96incident/s]
Processing dataset: ./Datasets/2021\csv_incidentiGiugno.csv
Database 'version2' already exists.
Processing incidents: 100%|                    | 6514/6514 [14:14<00:00,  7.63incident/s]
Processing dataset: ./Datasets/2021\csv_incidentiLuglio.csv
Database 'version2' already exists.
Processing incidents: 100%|                    | 6392/6392 [16:11<00:00,  6.58incident/s]
Processing dataset: ./Datasets/2021\csv_incidentiAgosto.csv
Database 'version2' already exists.
Processing incidents: 100%|                    | 4468/4468 [12:33<00:00,  5.93incident/s]
Processing dataset: ./Datasets/2021\csv_incidentiSettembre.csv
Database 'version2' already exists.
Processing incidents:  81%|                    | 5418/6691 [16:42<04:33,  4.65incident/s]
```

```
[?] Seleziona gli script da eseguire:
   [ ] Scripts/Scripts_PgAdmin/v1.py
   [ ] Scripts/Scripts_Neo4j/csv-to-neo4j-incidenti_v1.py
 > [X] Scripts/Scripts_Neo4j/csv-to-neo4j-incidenti_v2.py
   [ ] Scripts/Scripts_Neo4j/csv-to-neo4j-incidenti_v3.py
   [ ] Scripts/Scripts_Neo4j/csv-to-neo4j-incidenti_v4.py
   [ ] Esegui tutti gli script

Database 'version2' cancellato.
Database 'version2' ricreato.
Esecuzione dello script: Scripts/Scripts_Neo4j/csv-to-neo4j-incidenti_v2.pyProcessing dataset: ./Datasets/2021\csv_incidentiGennaio.csv
Database 'version2' already exists.
Processing incidents: 100%|    | 4610/4610 [03:22<00:00, 22.73incident/s]
Processing dataset: ./Datasets/2021\csv_incidentiFebbraio.csv
Database 'version2' already exists.
Processing incidents:  50%|    | 2557/5073 [02:38<03:03, 13.75incident/s]
```

Execution of runner.py:
you can choose which script to run

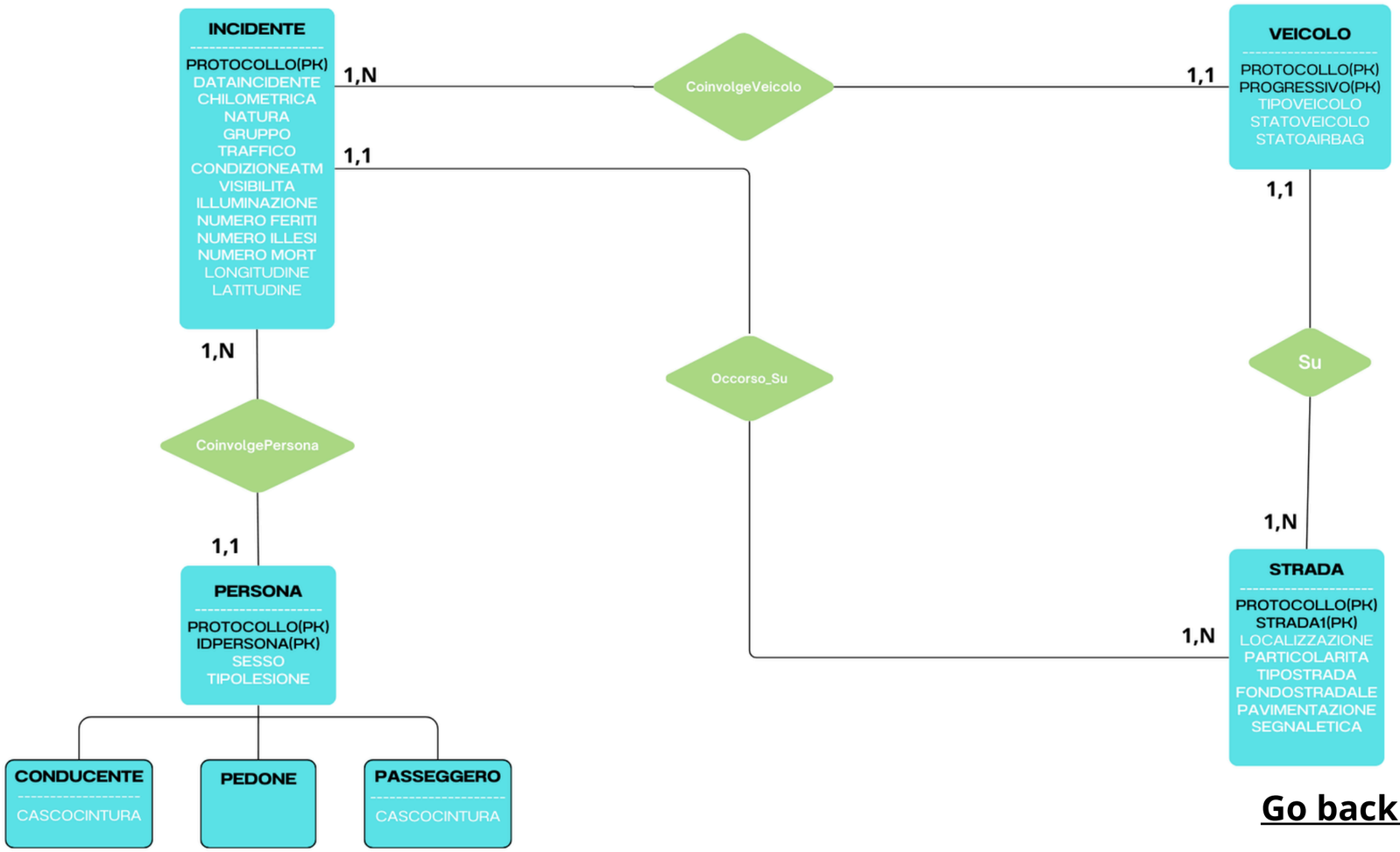
Go back on table of content

SQL Modeling

This ER diagram illustrates the relationships between the main entities involved in traffic incidents.

It shows how incidents connect to vehicles, people, and roads, providing a clear view of the data structure. Understanding these relationships is essential for effective data management and analysis.

SQL Modeling - ER diagram



Go back on table of content

Graph Modeling

Neo4J - First Version

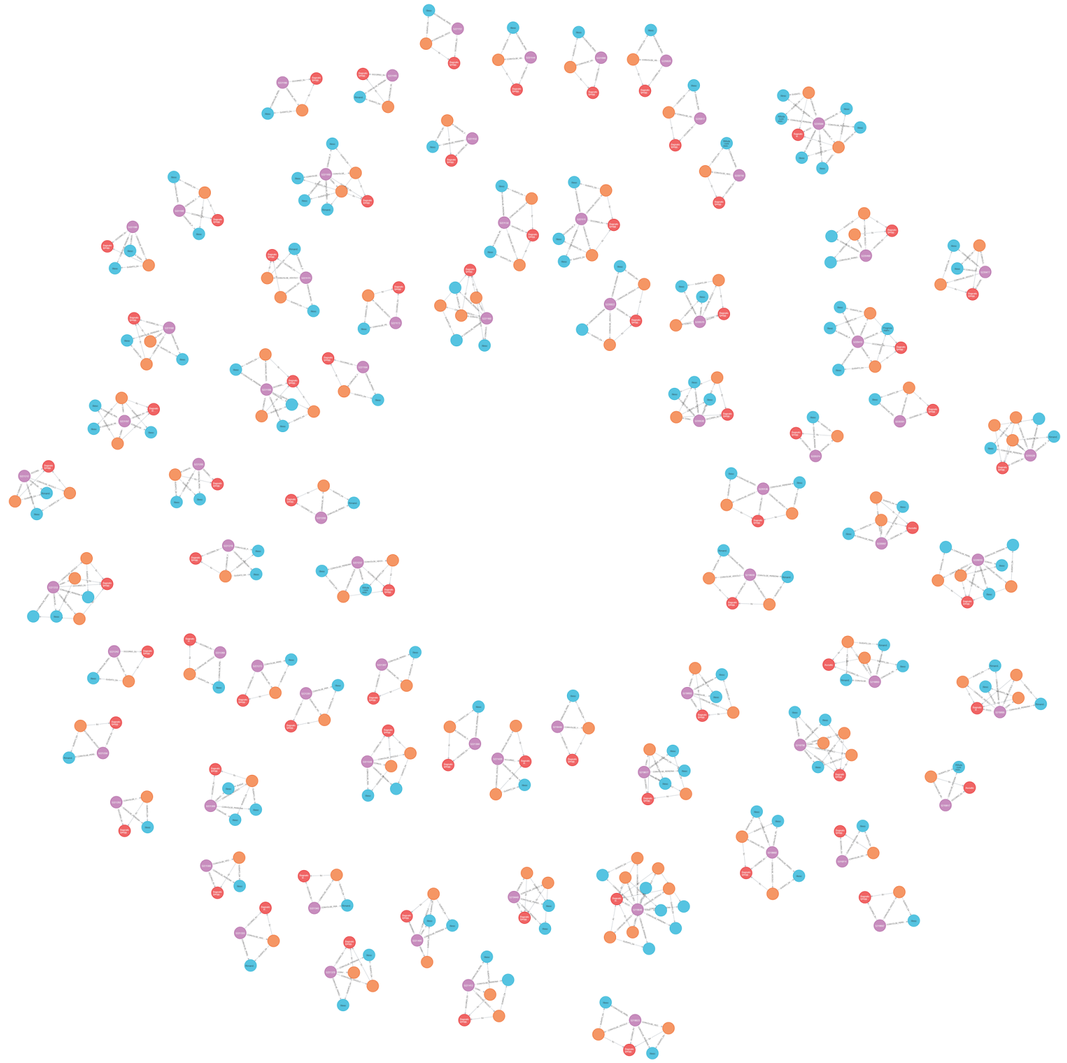
The data was represented by several small, disconnected subgraphs.
Each subgraph contained vehicles, people, incidents, and roads.

However, these entities were isolated from one another, so in many case we don’t take advantages using a Graph DB.

This disconnection limited the potential for relational queries and analysis, preventing the effective exploitation of the graph database’s.

Graph Modeling

Neo4J - First Version



Go back on table of content

Graph Modeling

Neo4J - Second Version

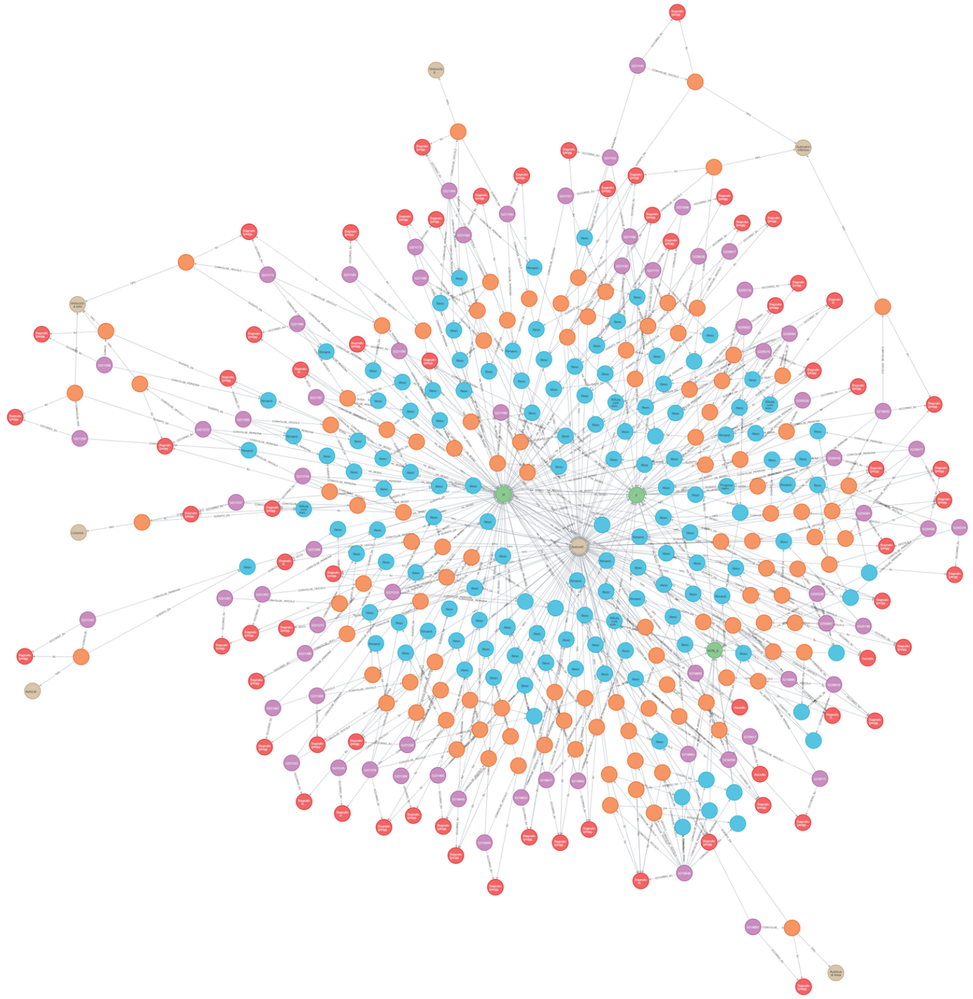
In this version, we improved the connections between different parts of the graph.

We added two new nodes: TipoVeicolo, which was previously just a property of the vehicle, making it easier to group and connect vehicles across incidents.

We also added a Sesso node, which now represents the gender of individuals involved, helping to better analyze the difference of performance.

Graph Modeling

Neo4J - Second Version



Go back on table of content

Graph Modeling

Neo4J - Third Version

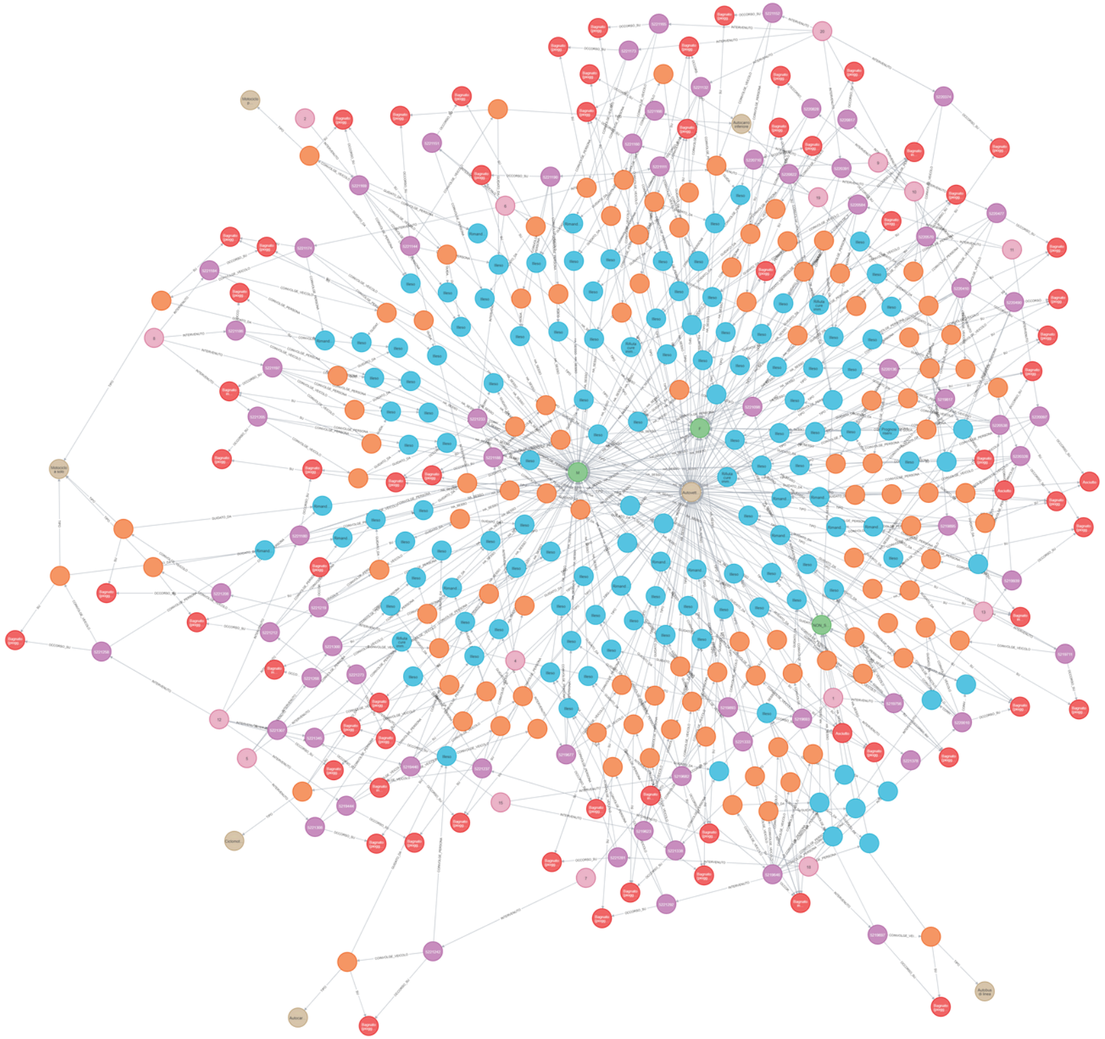
Connectivity is further improved with Group representing the number of police involved in an incident.

Now, we also have a relationship between the Group nodes and the incidents in which this group is involved.

Queries that require exploring relationships can be executed faster when nodes are well connected.

Graph Modeling

Neo4J - Third Version



Go back on table of content

Graph Modeling

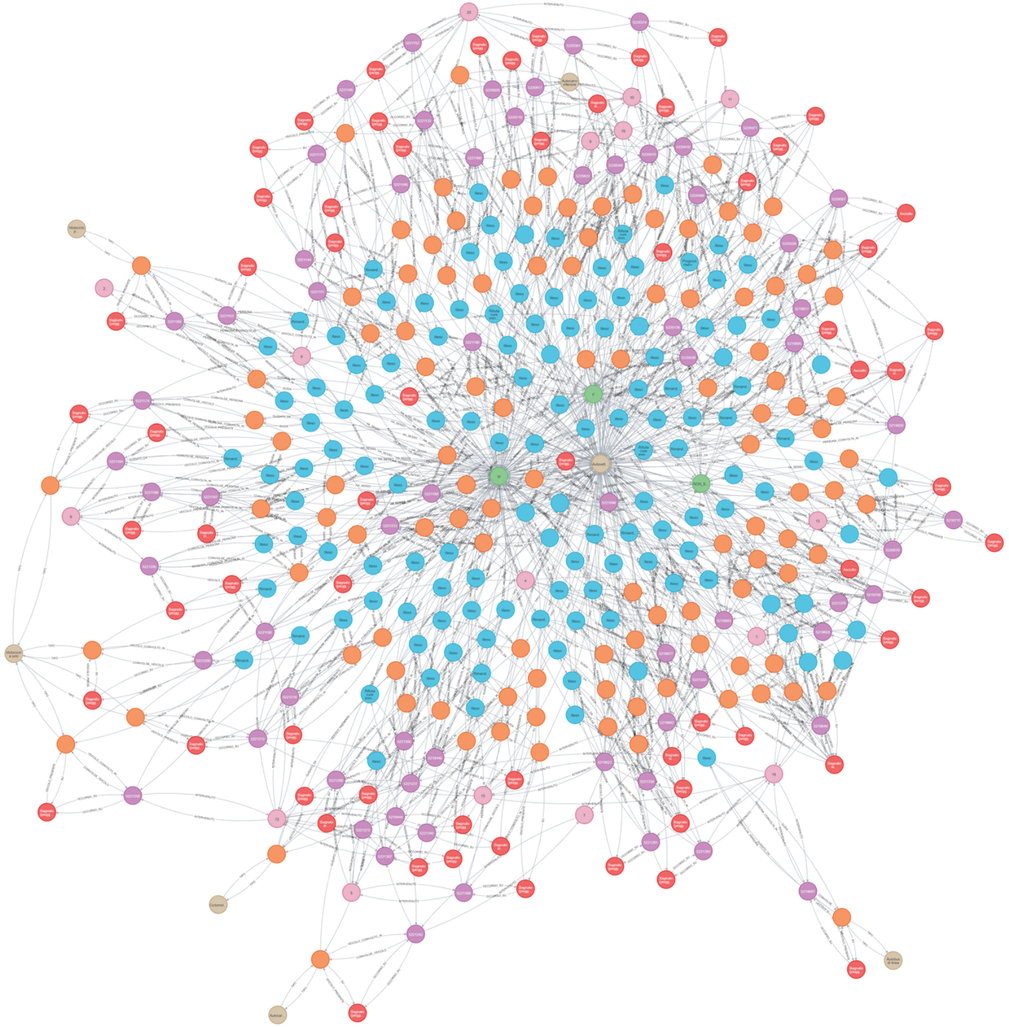
Neo4J - Fourth Version

In this version, we have introduced edges in both directions between the nodes, changing the graph's structure.

The Group node, representing the number of police involved in an incident, is now connected to the incidents in which this group is involved.

Graph Modeling

Neo4J - Fourth Version



Go back on table of content

Comparison between
Graph Databases and Relational Databases

GraphDatabase (NOSQL)

- Uses a graph structure
- Schemaless, you can accumulate data incrementally, providing flexibility
- Provide good performance, because they avoid classical joins

Relational Database (SQL)

- The schema of a relational database is static
- Do not well behave in the presence of high variety in the data
- Query execution times increase as the size of relational tables

Go back on table of content

Comparison between Neo4J and PostgreSQL

Go back on table of content

Writing Queries in Neo4J

- Cypher uses a much more graph-oriented syntax, with **MATCH**, **RETURN**, and parentheses to represent nodes **(n)** and relations **[r]**
- Cypher query language focuses on **pattern matching**
- Complex queries can be written concisely due to inherent graph structure.
- Example: Find all incidents on a specific road with injured passengers.

Writing Queries in PostgreSQL

- SQL has a traditional syntax structured with SELECT, FROM, WHERE
- SQL emphasizes joins and table relations.
- Queries for complex relationships may require multiple joins.
- SQL is optimized for aggregate functions like COUNT, SUM, MAX, etc
- Example: Query involving multiple tables for incidents, roads, and vehicles.

Go back on table of content

Query examples and comparison

Go back on table of content

Query 1

View informations about all incidents

Neo4J

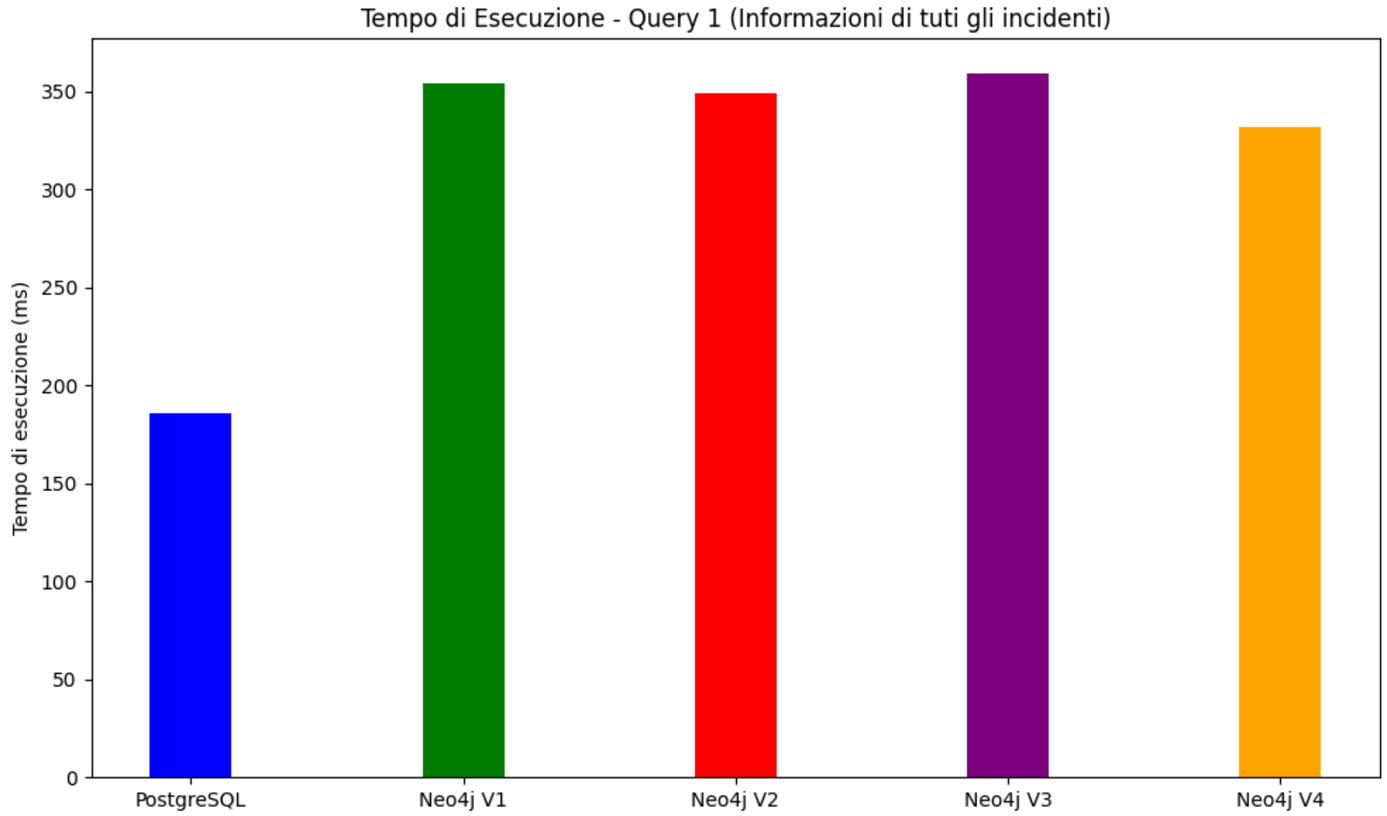
```
MATCH (i:Incidente)
  RETURN i
```

PostgreSQL

```
SELECT *
   FROM Incidente
```

Query 1

Performance comparison



Query 1

Result analysis

SQL databases are generally more optimized for queries involving structured tables, especially when working with queries that require you to select entire tables.

Furthermore, in this case I don't have to do any type of join.

Go back on table of content

Query 2

View all accidents and vehicles involved

Neo4J

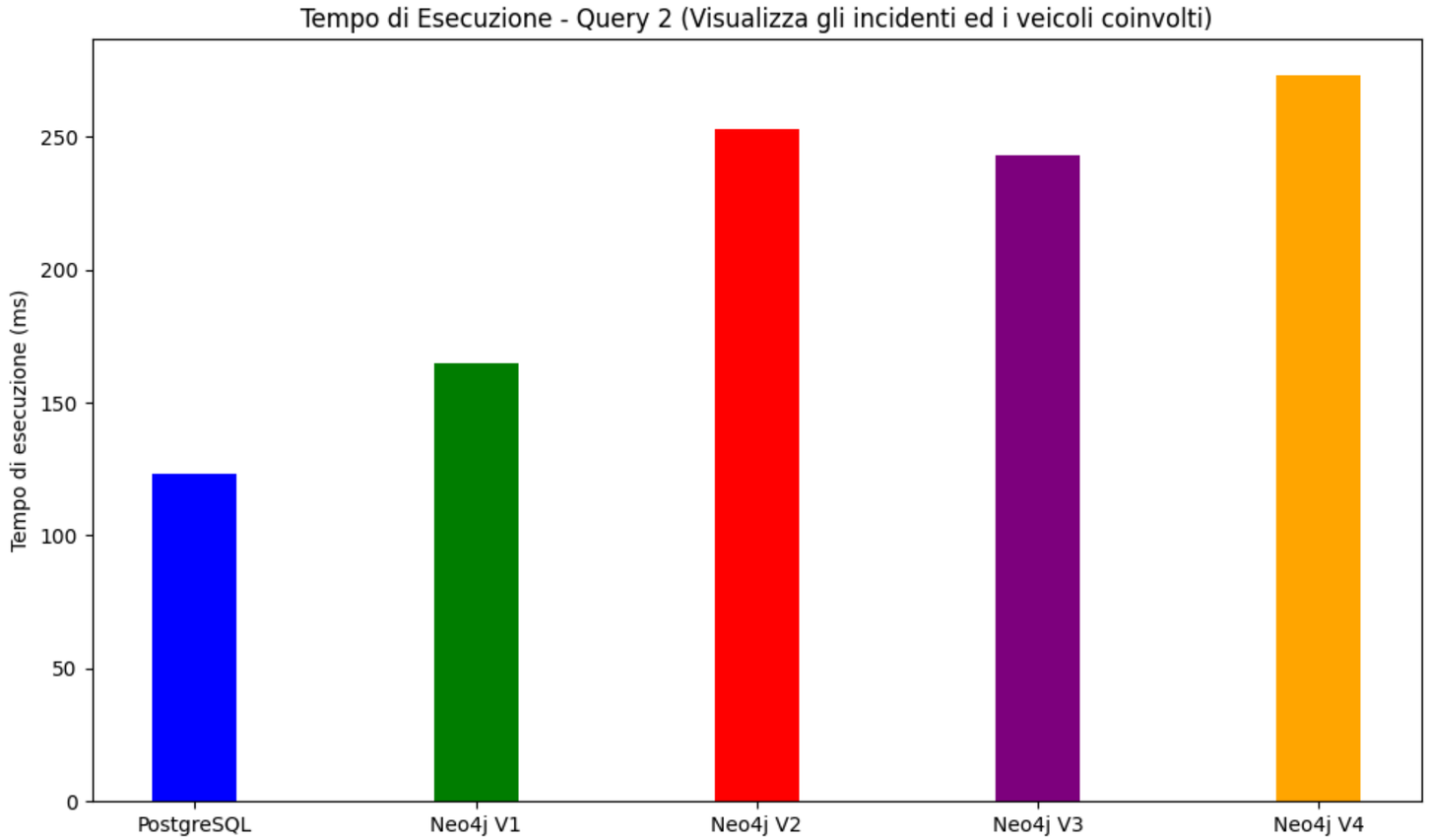
```
MATCH(i:Incidente)-[:COINVOLGE_VEICOLO]->(v:Veicolo)
  RETURN i,v
```

PostgreSQL

```
SELECT i.protocollo
  FROM Incidente i join veicolo v on i.protocollo=v.protocollo
```

Query 2

Performance comparison



Query 2

Result analysis

In this case, Neo4j explores all the Incident nodes without knowing a priori what the starting node is (for example, a specific protocol value), while in SQL, you can be more precise and filter the data through the join key, the which makes the search more targeted.

Go back on table of content

Query 3

Count number of accidents for given group

Neo4J

V1 - V2

```
MATCH (i:Incidente)
WITH toInteger(i.gruppo) AS gruppo, COUNT(i) AS numero_incidenti
RETURN gruppo, numero_incidenti
ORDER BY gruppo;
```

V3 - V4

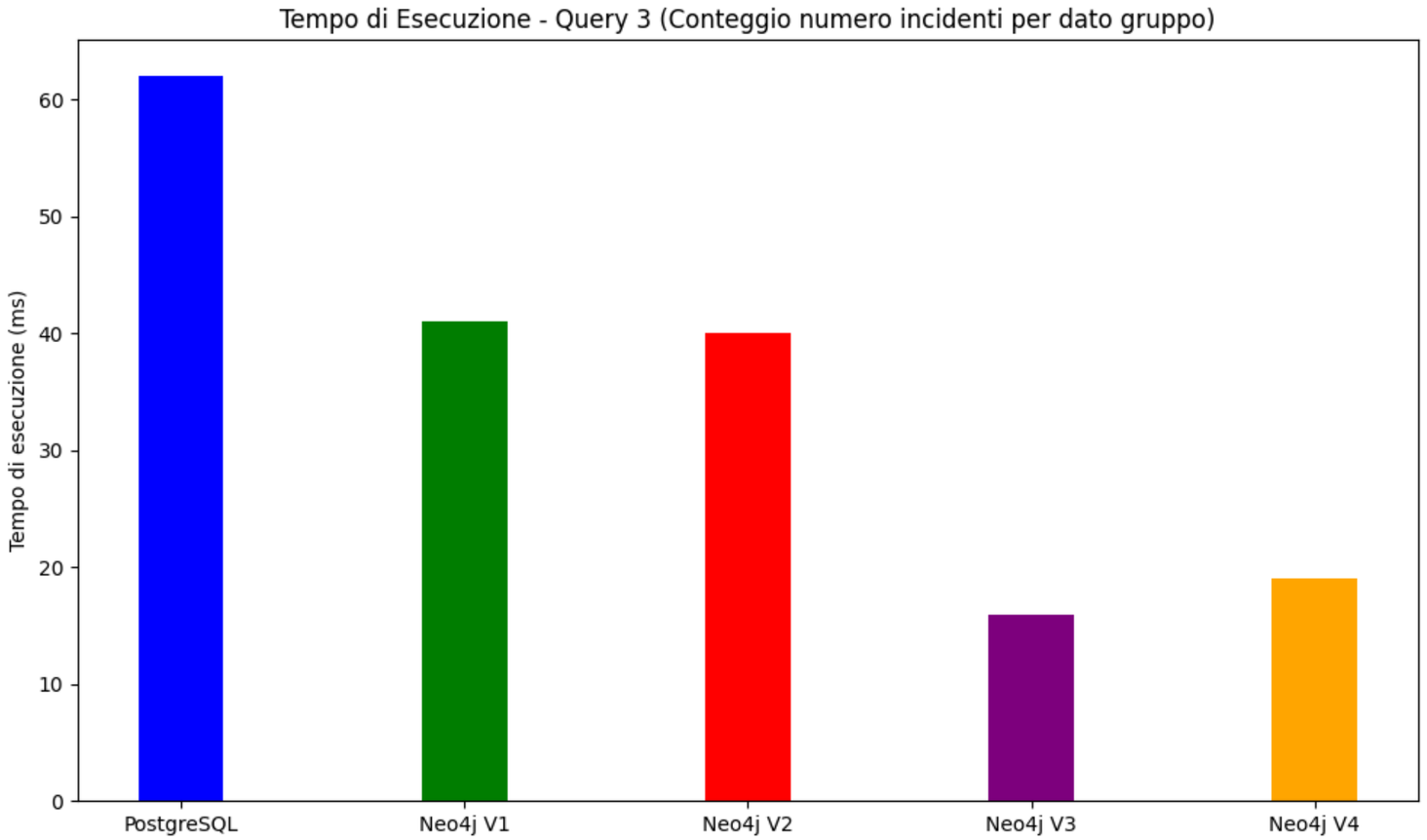
```
MATCH (i:Incidente)<-[:INTERVENUTO]-(g:Gruppo)
WITH g, COUNT(i) AS numero_incidenti
RETURN g.nome AS gruppo, numero_incidenti
ORDER BY toInteger(g.nome);
```

PostgreSQL

```
SELECT gruppo, COUNT(*) AS numero_incidenti
  FROM incidente
  GROUP BY gruppo
  ORDER BY gruppo;
```

Query 3

Performance comparison



Query 3

Result analysis

We can notice a notable improvement in the second version of Neo4J since, in this version we have a Group node which is directly connected to the nodes in which it is involved.

Regarding PostgreSQL, one might expect better performance compared to Neo4j Version 1. However, being a relational database, it works with rows and tables and must perform a full table scan for each GROUP BY operation. If the table is very large, the aggregation operation can be slow.

Go back on table of content

Query 4

View information on road accidents involving a specific type of vehicle

Neo4J

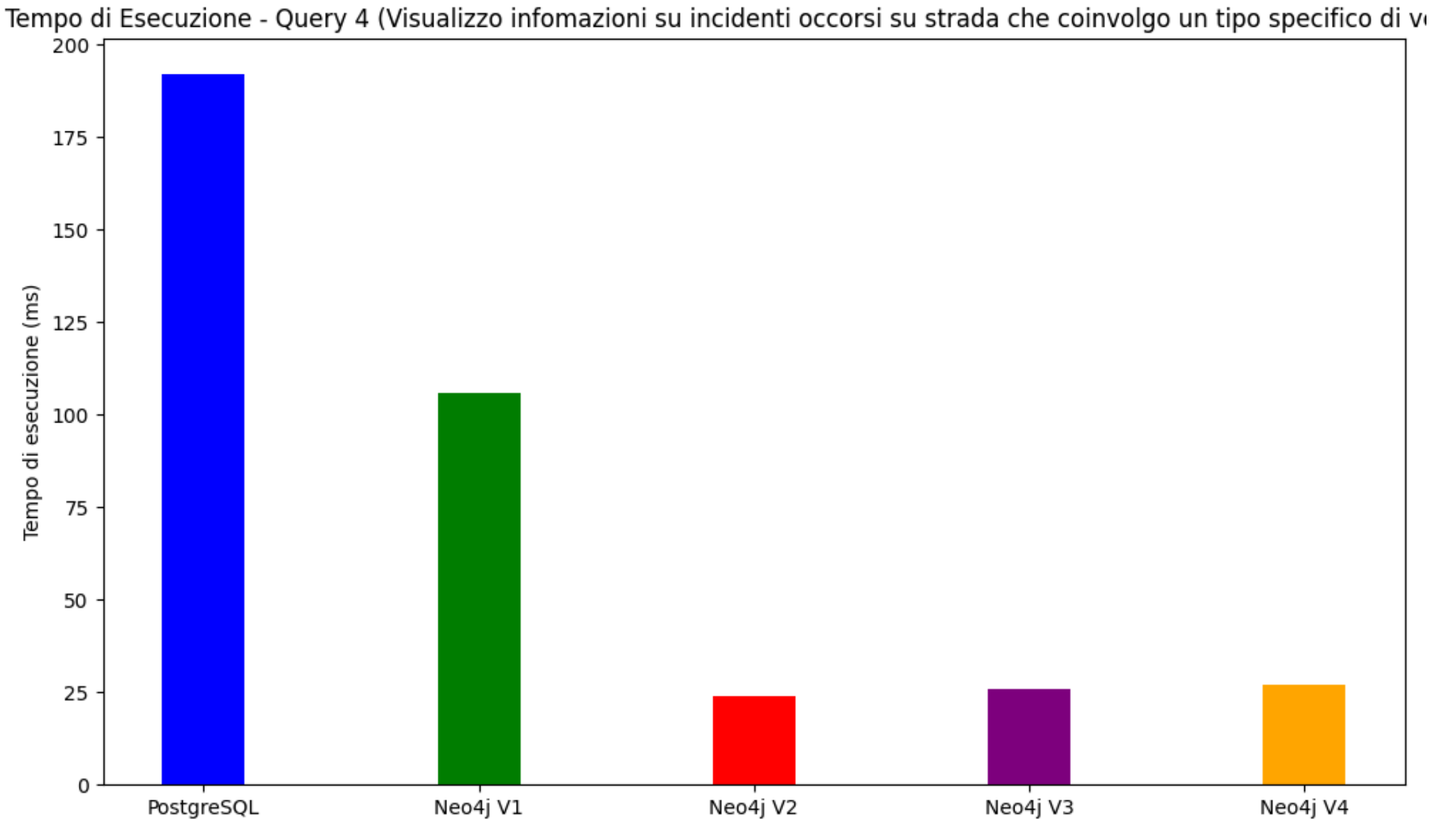
V1
```
MATCH (v:Veicolo {tipoveicolo: "Velocipede"})<-[:COINVOLGE_VEICOLO]-(i:Incidente)-[:OCCORSO_SU]->(s:Strada)
RETURN v, i, s
```
V2,V3
```
MATCH (tv:TipoVeicolo {nome:"Velocipede"})<-[:TIPO]-(v:Veicolo)<-[:COINVOLGE_VEICOLO]-(i:Incidente)-[:OCCORSO_SU]->(s:Strada)
RETURN v, i, s
```
V4
```
MATCH (tv:TipoVeicolo {nome:"Velocipede"})-[:TIPO]->(v:Veicolo)-[:VEICOLO_COINVOLTO_IN]->(i:Incidente)-[:OCCORSO_SU]->(s:Strada)
RETURN v, i, s
```

PostgreSQL

```
SELECT v., i., s.*
  FROM Veicolo v
  JOIN Incidente i ON i.protocollo = v.protocollo
  JOIN Strada s ON s.protocollo = i.protocollo
  WHERE v.tipo_veicolo = 'Velocipede';
```

Query 4

Performance comparison



Query 4

Result analysis

Neo4J's graph traversal allows it to directly reach the relevant data (especially with the second query that uses the TipoVeicolo node), making it faster than PostgreSQL, which needs to scan through the tables and perform join operations for each part of data.

Go back on table of content

Query 5

Identify all accidents involving a private passenger car and find all people involved in related accidents up to a depth of 3.

Neo4J

V1

```
MATCH (i:Incidente)-[:COINVOLGE_VEICOLO]->(v:Veicolo)
  WHERE v.tipoveicolo = "Autovettura privata"
  MATCH (i)-[:COINVOLGE_PERSONA*1..3]->(p:Persona)
  RETURN DISTINCT i.protocollo, p.idpersona,
    v.tipoveicolo AS tipoVeicolo
  ORDER BY i.protocollo
```

V2,V3,V4

```
MATCH (i:Incidente)-[:COINVOLGE_VEICOLO]->(v:Veicolo)-[:TIPO]->(t:TipoVeicolo
{nome: "Autovettura privata"}),
  (i)-[:COINVOLGE_PERSONA*1..3]->(p:Persona)
  RETURN DISTINCT i.protocollo, p.idpersona, t.nome
  ORDER BY i.protocollo
```

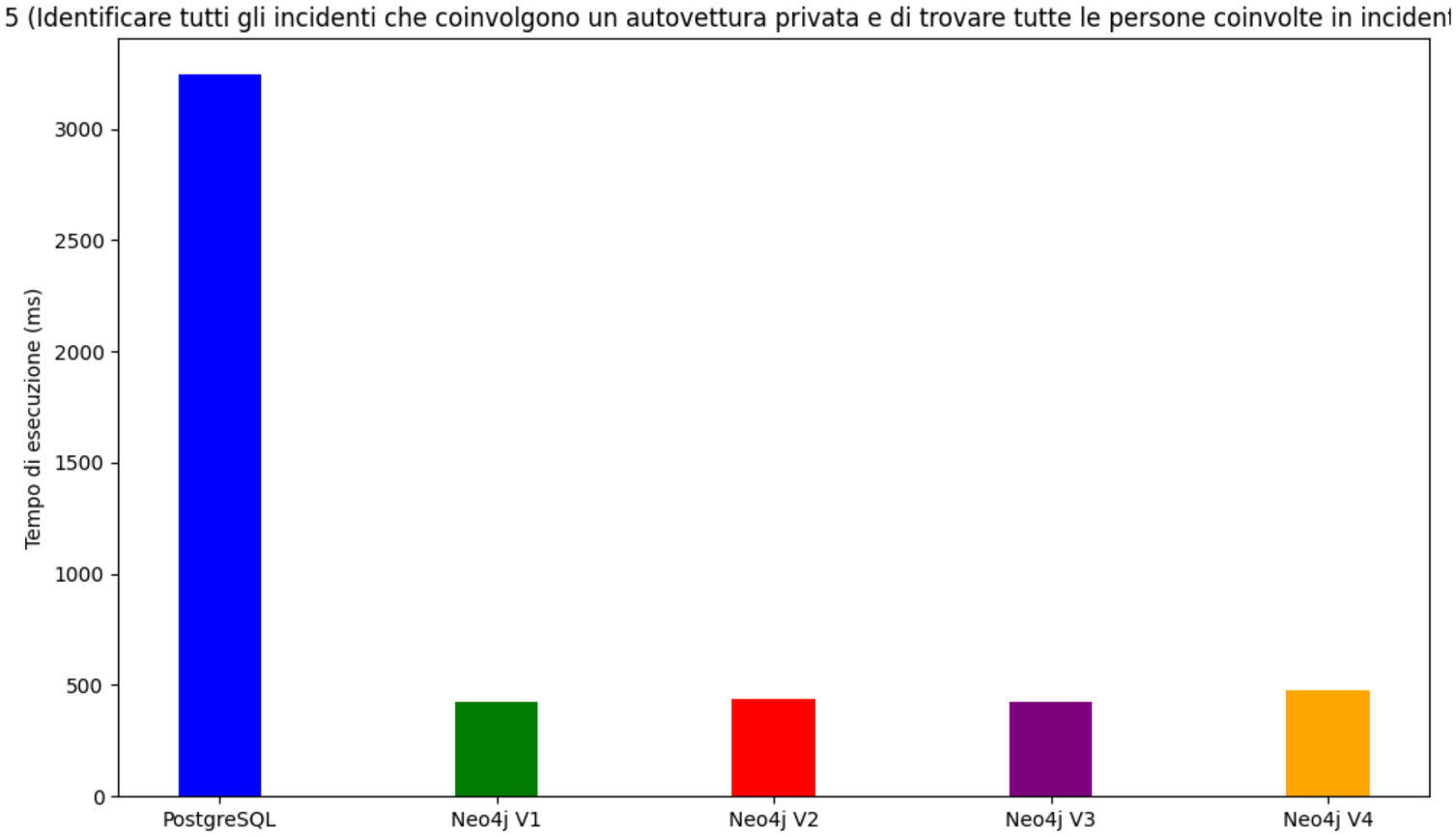
PostgreSQL

```
SELECT i.protocollo, p1.idpersona, v.tipo_veicolo
  FROM incidente AS i
  JOIN veicolo AS v ON i.protocollo = v.protocollo
  JOIN persona AS p1 ON i.protocollo = p1.protocollo
  WHERE v.tipo_veicolo = 'Autovettura privata'
UNION
SELECT i.protocollo, p2.idpersona, v.tipo_veicolo
  FROM incidente AS i
  JOIN veicolo AS v ON i.protocollo = v.protocollo
  JOIN persona AS p1 ON i.protocollo = p1.protocollo
  JOIN persona AS p2 ON p1.protocollo = p2.protocollo
  AND p1.idpersona <> p2.idpersona
  WHERE v.tipo_veicolo = 'Autovettura privata'
```

```
UNION
SELECT i.protocollo, p3.idpersona, v.tipo_veicolo
  FROM incidente AS i
  JOIN veicolo AS v ON i.protocollo = v.protocollo
  JOIN persona AS p1 ON i.protocollo = p1.protocollo
  JOIN persona AS p2 ON p1.protocollo = p2.protocollo AND p1.idpersona <> p2.idpersona
  JOIN persona AS p3 ON p2.protocollo = p3.protocollo AND p2.idpersona <> p3.idpersona
  WHERE v.tipo_veicolo = 'Autovettura privata'
  ORDER BY protocollo;
```

Query 5

Performance comparison



Query 5

Result analysis

The PostgreSQL query performs several JOIN operations, then it performs multiple unions to account for cases where multiple people (p1, p2, p3) are involved in the same incident, and we know how join operations are expensive.

Moreover the where clause (tipo_veicolo = 'Autovettura privata') may not significantly reduce the number of rows in PostgreSQL if there are many "Autovettura privata" vehicles, and this contributes to the slowness of the query.

Go back on table of content

Query 6

Detailed information on specific road accidents

Neo4J

V1

```
MATCH (i:Incidente)-[:COINVOLGE_VEICOLO]->(v:Veicolo)
  WHERE v.tipoveicolo = "Autovettura privata"
  MATCH (i)-[:COINVOLGE_PERSONA*1..3]->(p:Persona)
  MATCH (i)-[:OCCORSO_SU]->(s:Strada)
  RETURN i.protocollo, p.idpersona, v.tipoveicolo AS tipoVeicolo,
s.nome AS nomestrada
  ORDER BY i.protocollo;
```

V2, V3, V4

```
MATCH (i:Incidente)-[:COINVOLGE_VEICOLO]->(v:Veicolo)-[:TIPO]->(t:TipoVeicolo {nome:
"Autovettura privata"}),
  (i)-[:COINVOLGE_PERSONA*1..3]->(p:Persona),
  (i)-[:OCCORSO_SU]->(s:Strada)
 RETURN i.protocollo, p.idpersona, t.nome AS tipo_veicolo, s.nome AS nome_strada
 ORDER BY i.protocollo;
```

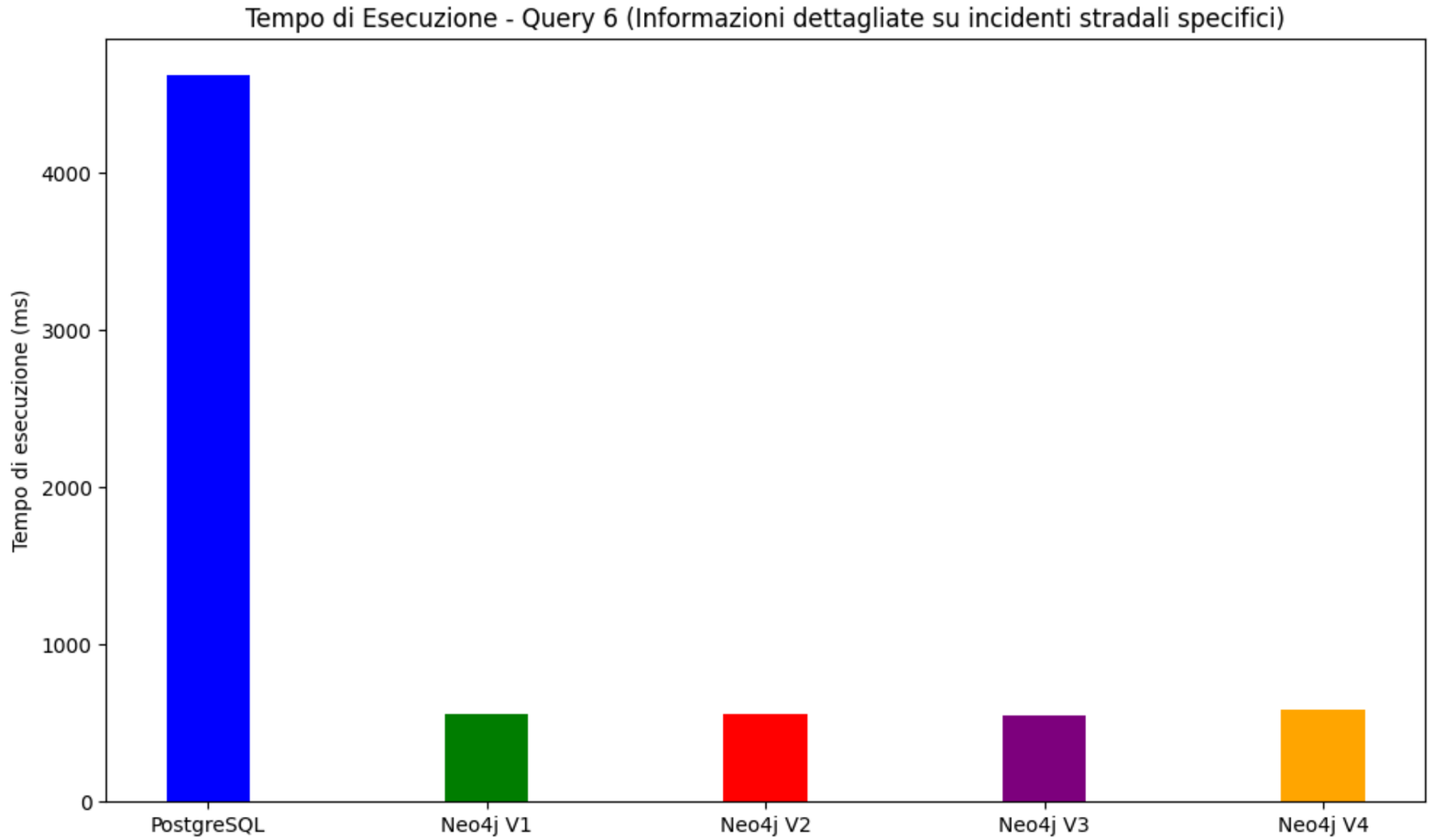
PostgreSQL

```
SELECT i.protocollo, p1.idpersona, v.tipo_veicolo, s.strada1 AS nome_strada
FROM incidente AS i
JOIN veicolo AS v ON i.protocollo = v.protocollo
JOIN persona AS p1 ON i.protocollo = p1.protocollo
JOIN strada AS s ON i.protocollo = s.protocollo
WHERE v.tipo_veicolo = 'Autovettura privata'
    UNION
SELECT i.protocollo, p2.idpersona, v.tipo_veicolo, s.strada1 AS nome_strada
FROM incidente AS i
JOIN veicolo AS v ON i.protocollo = v.protocollo
JOIN persona AS p1 ON i.protocollo = p1.protocollo
JOIN persona AS p2 ON p1.protocollo = p2.protocollo AND p1.idpersona <> p2.idpersona
JOIN strada AS s ON i.protocollo = s.protocollo
WHERE v.tipo_veicolo = 'Autovettura privata'
```

```
    UNION
SELECT i.protocollo, p3.idpersona, v.tipo_veicolo, s.strada1 AS nome_strada
FROM incidente AS i
JOIN veicolo AS v ON i.protocollo = v.protocollo
JOIN persona AS p1 ON i.protocollo = p1.protocollo
JOIN persona AS p2 ON p1.protocollo = p2.protocollo AND p1.idpersona <>
p2.idpersona
JOIN persona AS p3 ON p2.protocollo = p3.protocollo AND p2.idpersona <>
p3.idpersona
JOIN strada AS s ON i.protocollo = s.protocollo
WHERE v.tipo_veicolo = 'Autovettura privata'
ORDER BY protocollo;
```

Query 6

Performance comparison



Query 6

Result analysis

This query is very similar to the previous one, except that we add an additional JOIN. As expected, the execution time on PostgreSQL increases further.

The time on Neo4j also increases, as it is necessary to use the "SU" edge to reach the strada nodes.

Go back on table of content

Query 7

Look for specific accidents involving men with injury type injury = Prognosi Riservata and there are at least two different vehicles involved in the same accident where the road has a dry surface

Neo4J

V1

```
MATCH (i:Incidente)-[:COINVOLGE_PERSONA]->
    (p:Persona {sesso: 'M', tipolesione: 'Prognosi riservata'}),
    (i)-[:COINVOLGE_VEICOLO]->(v1:Veicolo)-[:SU]->
    (s:Strada {fondostradale: 'Asciutto'}),
    (i)-[:COINVOLGE_VEICOLO]->(v2:Veicolo)-[:SU]->(s)
  WHERE v1.tipoveicolo <> v2.tipoveicolo
  RETURN i.protocollo, p.idpersona, p.tipolesione, v1.tipoveicolo, v2.tipoveicolo
```

V2,V3,V4

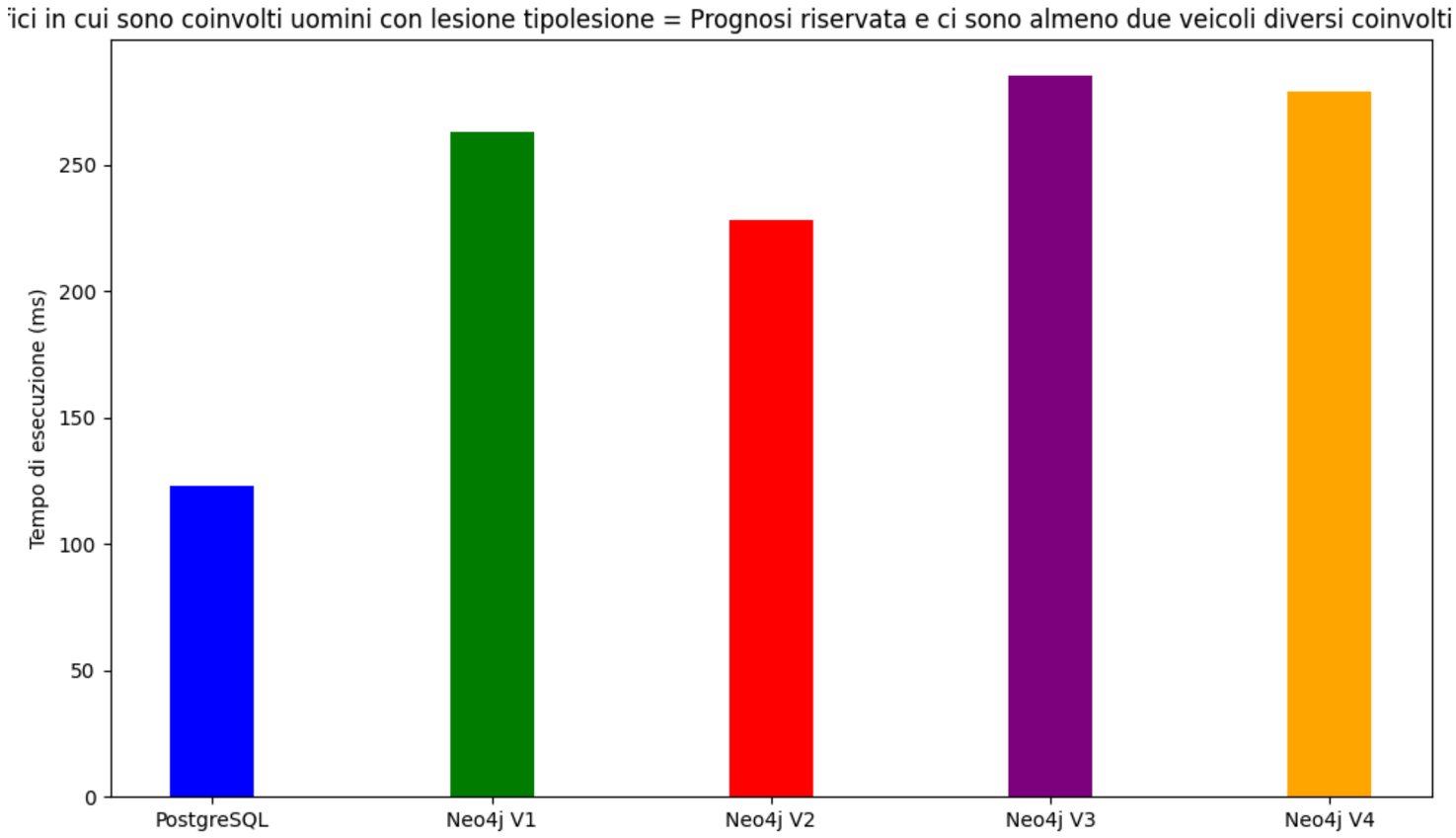
```
MATCH (i:Incidente)-[:COINVOLGE_PERSONA]->
    (p:Persona {sesso: 'M', tipolesione: 'Prognosi riservata'}),
    (i)-[:COINVOLGE_VEICOLO]->(v1:Veicolo)-[:TIPO]->
    (t1:TipoVeicolo),
    (i)-[:COINVOLGE_VEICOLO]->(v2:Veicolo)-[:TIPO]->(t2:TipoVeicolo),
    (v1)-[:SU]->(s:Strada {fondostradale: 'Asciutto'}),
    (v2)-[:SU]->(s)
  WHERE t1.nome <> t2.nome
  RETURN i.protocollo, p.idpersona, p.tipolesione, t1.nome AS tipoVeicolo_v1,
t2.nome AS tipoVeicolo_v2
```

PostgreSQL

```
SELECT i.protocollo, p.idpersona, p.tipolesione, v1.tipo_veicolo AS tipoVeicolo_v1, v2.tipo_veicolo AS tipoVeicolo_v2
  FROM incidente AS i
  JOIN persona AS p ON i.protocollo = p.protocollo AND p.sesso = 'M' AND p.tipolesione = 'Prognosi riservata'
  JOIN veicolo AS v1 ON i.protocollo = v1.protocollo
  JOIN veicolo AS v2 ON i.protocollo = v2.protocollo
  JOIN strada AS s ON v1.protocollo = s.protocollo AND s.fondostradale = 'Asciutto'
  WHERE v1.tipo_veicolo <> v2.tipo_veicolo
  ORDER BY i.protocollo
```

Query 7

Performance comparison



Query 7

Result analysis

These results may be unexpected, given the various joins, but we can see that the condition on "Prognosi riservata" applied to the persona table significantly reduces the number of rows to analyze and thus the rows to join.

Go back on table of content

Query 8

Incident information starting from a specific node

Neo4J

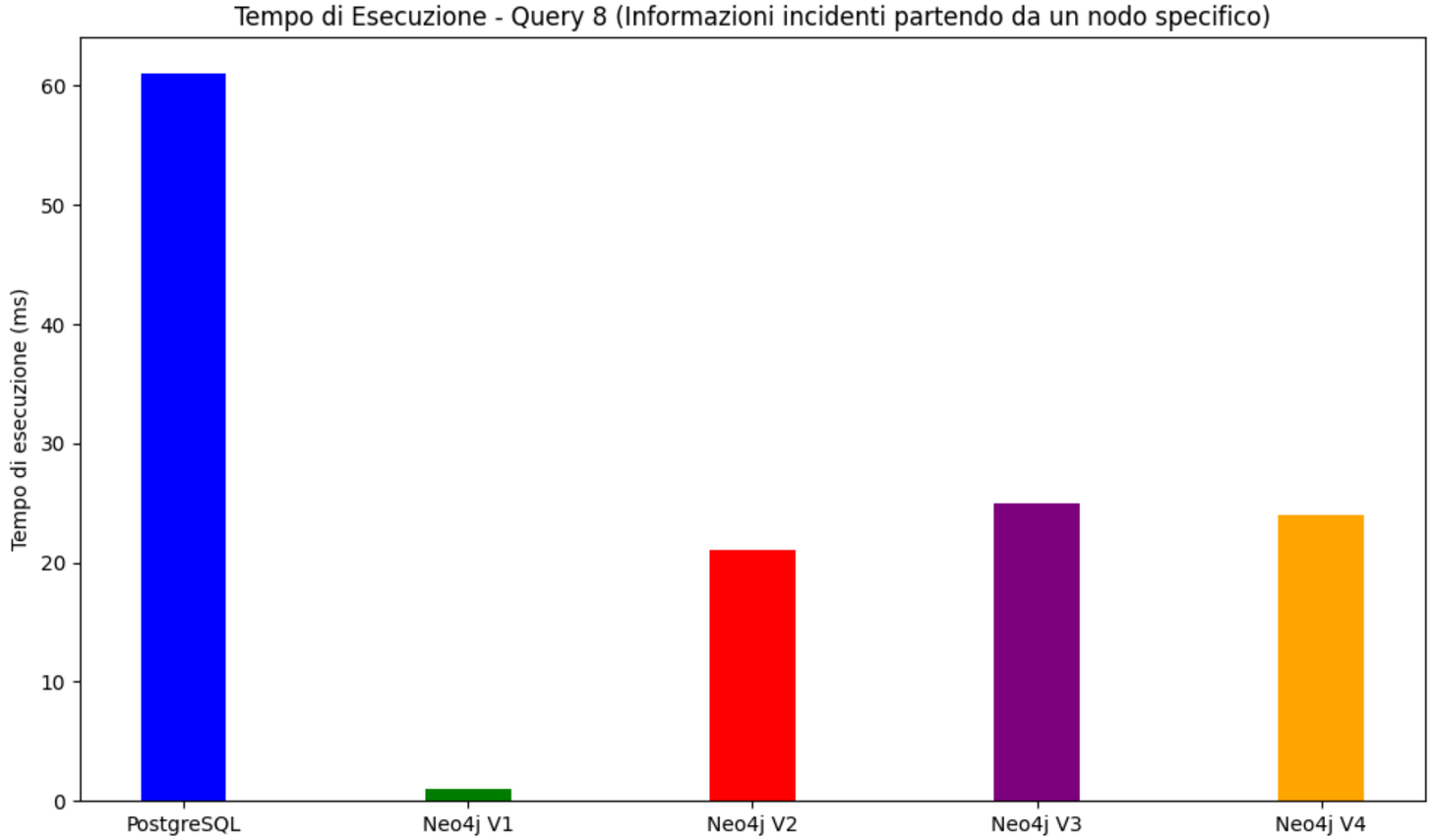
```
MATCH (i:Incidente {protocollo: '4733221'})-[:COINVOLGE_VEICOLO]->
(v:Veicolo)-[:SU]->(s:Strada)
  RETURN i,v,s
```

PostgreSQL

```
SELECT i.*, v.*, s.*
  FROM incidente AS i
  JOIN veicolo AS v ON i.protocollo = v.protocollo
  JOIN strada AS s ON v.protocollo = s.protocollo
  WHERE i.protocollo = '4733221';
```

Query 8

Performance comparison



Query 8

Result analysis

In this Cypher query we start from a specific node (Incident with protocol: '4733221'), Neo4j can directly access that node and quickly navigate its relationships.

This approach makes the most of the graph-based architecture, avoiding searches over a large number of nodes and relationships.

In contrast, in SQL, even though the WHERE clause filters by protocollo = '4733221', the database still has to perform joins with the veicolo and strada tables to retrieve the results, which can be less efficient.

Go back on table of content

Query 9

Incident information starting from a generic node

Neo4J

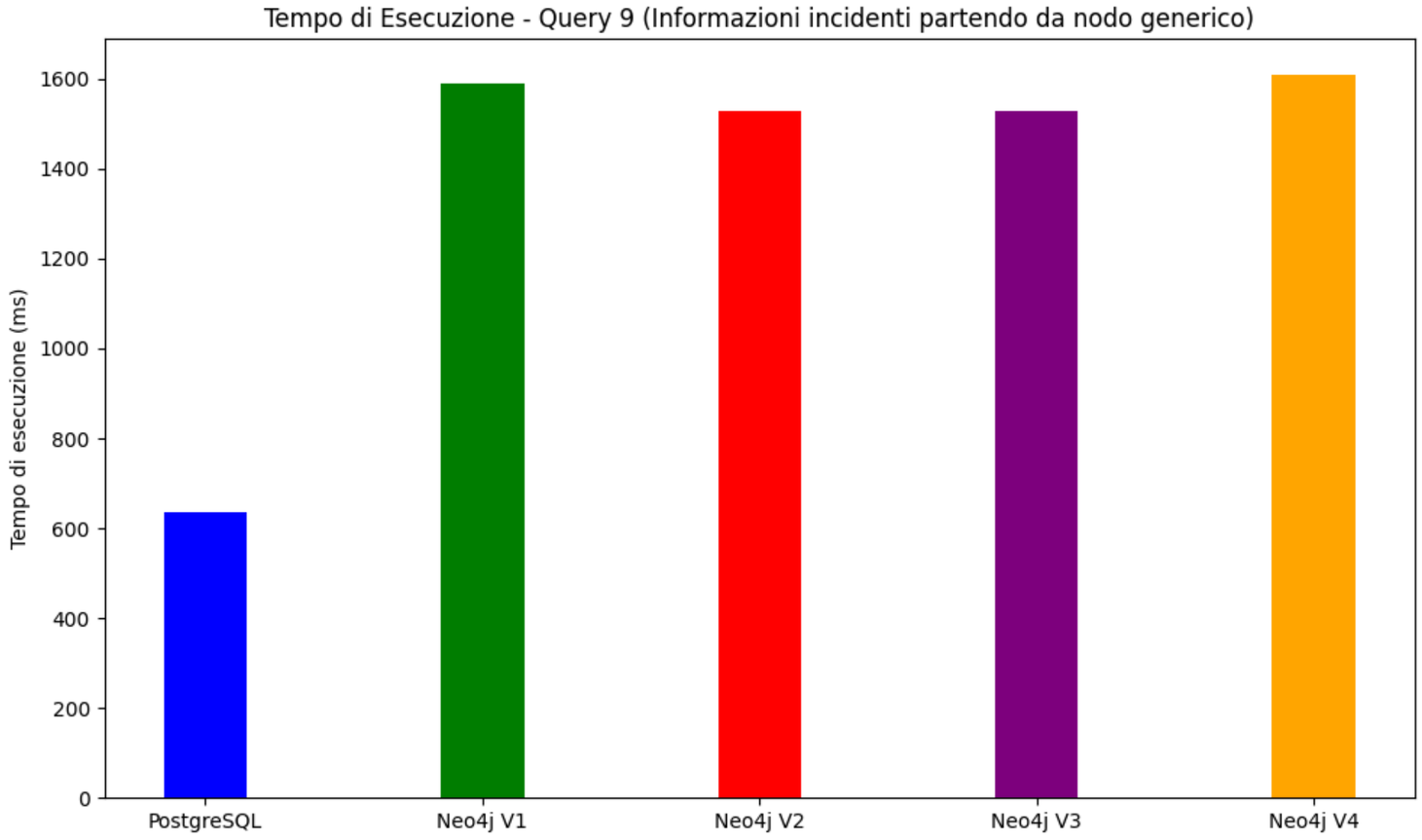
```
MATCH (i:Incidente)-[:COINVOLGE_VEICOLO]->(v:Veicolo),
(v)-[:SU]->(s:Strada)
  RETURN i, v, s
```

PostgreSQL

```
SELECT i.*, v.*, s.*
  FROM incidente AS i
  JOIN veicolo AS v ON i.protocollo = v.protocollo
  JOIN strada AS s ON v.protocollo = s.protocollo;
```

Query 9

Performance comparison



Query 9

Result analysis

This query is similar to the previous one, but instead of starting from a specific node we start from all the incident nodes.

This implies that, even if in the SQL case it is necessary to do two joins to obtain the result, we obtain better performance compared to neo4j, since in this case each incident node must be analyzed and then navigate along two edges to obtain all the information.

Go back on table of content

Query 10

Information on accidents where the 26 group intervened

Neo4J

V1-V2

```
MATCH (i:Incidente{gruppo: '26'})-[:COINVOLGE_VEICOLO]->
(v:Veicolo),(v)-[:SU]->(s:Strada)
  RETURN i, v, s
```

V3 - V4

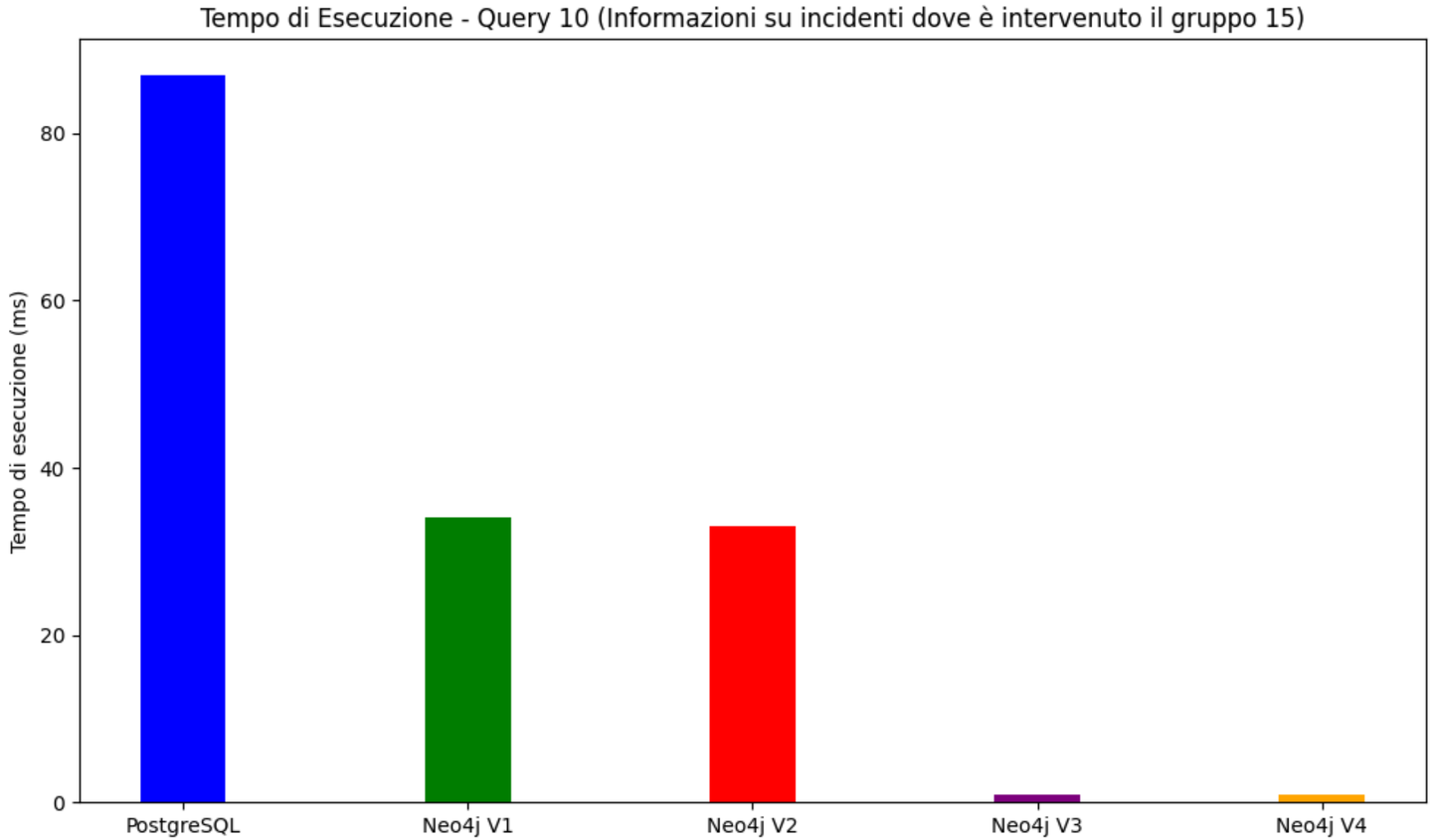
```
MATCH (g:Gruppo {nome: "26"})-[:INTERVENUTO]->(i:Incidente)-[:OCCORSO_SU]->
(s:Strada), (i)-[:COINVOLGE_VEICOLO]->(v:Veicolo)
  RETURN i, s, v
```

PostgreSQL

```
SELECT i.*, s.*, v*
  FROM incidente AS i
  JOIN strada AS s ON i.protocollo = s.protocollo
  JOIN veicolo AS v ON i.protocollo = v.protocollo
  WHERE i.gruppo = '26';
```

Query 10

Performance comparison



Query 10

Result analysis

From the chart we can see that by adding a "Group" node the performance improves drastically, as we can use the edge which connects it directly to the various incident nodes.

So starting from node 26 (this node was not chosen randomly, but the group in which it intervened least often in accidents) we navigate two edges to obtain information on vehicles and roads.

While without the "Gruppo" node it is necessary to analyze each individual node incident and filter based on the "group" property

Go back on table of content

Difficulty of Writing queries

The comparison between query 6 and 7 is particularly useful in illustrating the difference in ease of writing recursive queries.

In SQL, the query is more complicated because we need to manually repeat the JOIN and UNION clauses for each level of recursion. This makes the query longer and harder to read.

Instead, Cypher makes recursive queries easy to write with simple syntax, using patterns like *1..3.

Conclusion

In conclusion, as we expected, Neo4j performs better when starting from a specific node and traversing through relationships to reach other nodes.

This is because such an operation in SQL would involve multiple joins between tables, which can be less efficient.

However, we also observe that when we apply a filter that drastically reduces the number of tuples to join, the performance between the two systems becomes more comparable.

Conclusion

In addition, using double relationships in version 4 Neo4j does not result in better performance but simply provides another way of writing the same query, starting from a different node.

The performance does not improve because the underlying operation is essentially the same; it's just another way of structuring the query to traverse through the graph from a different perspective.

For more details, visit our GitHub repository:

SAPIENZA
UNIVERSITÀ DI ROMA

Thank for your attention!