UNIVERSITÀ DEGLI STUDI DI MESSINA

DIPARTIMENTO DI SCIENZE MATEMATICHE E INFORMATICHE,
SCIENZE FISICHE E SCIENZE DELLA TERRA

Corso di Laurea Triennale in Informatica

Identificazione e verifica degli UBO

Docente:
Prof. Antonio Celesti

Studenti:
Andrea Amante
Francesco Mondo

ANNO ACCADEMICO 2023/24

Indice

1 Introduzione **3**

2 Database **3**
2.1 Oracle SQL Developer 3
2.2 Neo4j 4

3 Progettazione **5**
3.1 Caso di studio 5
3.2 Datamodel 6

4 Implementazione e inserimento **7**
4.1 Oracle SQL Developer 7
4.2 Neo4j 7

5 Test **8**
5.1 Query 1 10
5.2 Query 2 12
5.3 Query 3 14
5.4 Query 4 16

6 Conclusioni **18**

1 Introduzione

I database di tipo **Not Only SQL** (abbreviato "**NoSQL**") sono sistemi di gestione dei dati sviluppati per rispondere all'esigenza di memorizzare ed elaborare grandi quantità di informazioni, offrendo soluzioni scalabili e flessibili. Questi, a differenza dei tradizionali *Relational DBMS*, consentono l'interazione con dati privi di uno schema rigido. L'avvento dei Big Data ha portato con sé lo sviluppo di nuove categorie di database, differenti per volume e complessità dei dati generati. Queste nuove categorie, ognuna con le proprie caratteristiche e peculiarità, includono:

- Database Key-Value
- Database Column-Oriented
- Database Document-Oriented
- Graph Database

I DBMS in questione rispettano il **Teorema CAP** (o **Teorema di Brewer**), il quale stabilisce che in un database distribuito non è possibile garantire simultaneamente le seguenti proprietà:

- **Consistency** (Consistenza)
- **Availability** (Disponibilità)
- **Partitioning** (Tolleranza al partizionamento)

Alla luce di queste considerazioni, i DBMS scelti per la comparazione sono **Oracle SQL Developer** e **Neo4j**.

2 Database

La scelta di Oracle SQL Developer e Neo4j come DBMS per la comparazione è dovuta alla loro adozione di paradigmi distinti e di facile implementazione. Entrambi i database permettono infatti un *deployment* agevole e forniscono sia un'interfaccia via riga di comando sia un'interfaccia grafica.

2.1 Oracle SQL Developer

Oracle SQL Developer rappresenta un *sistema di gestione di database relazionale* (**RDBMS**) altamente avanzato, caratterizzato da una struttura rigida di schema. É tipicamente collocato nella *combinazione CP*, se configurato per la coerenza forte, oppure nella *combinazione AP*, se configurato per alta disponibilità con una coerenza eventuale; la scelta dipende dalle specifiche esigenze applicative e dalle configurazioni del sistema.

La sua principale caratteristica risiede nell'implementazione del **modello relazionale**, dove i dati sono organizzati in tabelle composte da righe e colonne, garantendo una coerenza rigorosa dei dati attraverso l'uso di chiavi primarie e relazioni tra tabelle. Ogni tabella in Oracle SQL Developer deve rispettare uno schema predefinito,

che determina la *struttura* dei dati e i *tipi* di dati che possono essere memorizzati. Il DBMS supporta anche le transazioni **ACID** (Atomicità, Coerenza, Isolamento, Durabilità), che assicurano affidabilità nel completamento delle operazioni sul database. Inoltre offre funzionalità avanzate di gestione dei dati, come i **trigger**, le **stored procedure** e le **viste**, che permettono una gestione e un'elaborazione dei dati sofisticata.

Oracle SQL Developer consente la distribuzione fisica dei dati su più macchine attraverso la tecnologia **Real Application Clusters** (**RAC**), che permette a un database di essere eseguito su più server simultaneamente, garantendo alta disponibilità e scalabilità. Tuttavia, bisogna notare che la gestione di Oracle RAC può risultare complessa e richiede competenze tecniche elevate.

2.2 Neo4j

Neo4j è un DBMS NoSQL basato su *strutture a grafo* per l'organizzazione dei dati. La più piccola unità di dati è rappresentata da un nodo del grafo, il quale contiene informazioni sotto forma di **proprietà**. Le relazioni tra i vari nodi sono rappresentate come archi orientati, a cui è possibile associare a loro volta altre proprietà. Questa configurazione permette di sfruttare tutti i vantaggi caratteristici della rappresentazione a grafo, facilitando attraversamenti e ricerche su tutto il grafo o su sottosezioni di esso.

Le proprietà dei nodi e delle relazioni sono gestite come **coppie key-value**, permettendo un accesso rapido e diretto ai dati. I database a grafo seguono un approccio *scheme-free*, consentendo ai nodi di avere proprietà diverse anche all'interno della stessa categoria. Questa flessibilità li rende altamente adattabili a vari tipi di applicazioni. Tuttavia, una delle principali limitazioni dei database a grafo come Neo4j è l'impossibilità di partizionare efficacemente il grafo su più nodi fisici.

3 Progettazione

3.1 Caso di studio

Il caso di studio presentato riguarda la verifica e l'identificazione dei beneficiari effettivi di una persona giuridica o una transazione (**Ultimate Beneficial Owner** o **UBO**), fondamentali per garantire trasparenza in conformità alle normative antiriciclaggio (**AML**) e di contrasto al finanziamento del terrorismo (**CFT**).

Gli UBO sono le persone fisiche che possiedono o controllano un'entità legale, come una società, un trust, o un'altra organizzazione. Questi individui spesso operano attraverso strutture complesse e stratificate, rendendo difficile la loro identificazione. Tradizionalmente, la verifica degli UBO richiede notevoli risorse e può essere soggetta a errori umani. Il progetto presentato si concentra sull'adozione di tecniche avanzate di visualizzazione grafica e analisi dei dati per migliorare l'efficienza e l'accessibilità di tali processi, sfruttando la potenza delle moderne tecnologie di visualizzazione grafica per rappresentare le relazioni tra entità aziendali in modo chiaro e intuitivo.

In particolare, utilizziamo una rappresentazione a grafo per mappare e analizzare le connessioni tra diverse entità; ciò semplifica la visualizzazione delle complesse interconnessioni tra i vari stakeholder e permette di identificare rapidamente i beneficiari effettivi e di rilevare eventuali anomalie o schemi sospetti. I principali vantaggi riscontrabili includono:

- **Maggiore trasparenza**: La visualizzazione grafica facilita l'identificazione delle relazioni tra entità e beneficiari.
- **Efficienza operativa**: L'automazione dei processi riduce il tempo necessario per la verifica degli UBO.
- **Affidabilità**: L'uso di analisi avanzate riduce il rischio di errori e garantisce una maggiore precisione.

3.2 Datamodel

Le entità coinvolte nella creazione della base di dati sono sintetizzabili nel seguente **Datamodel**:

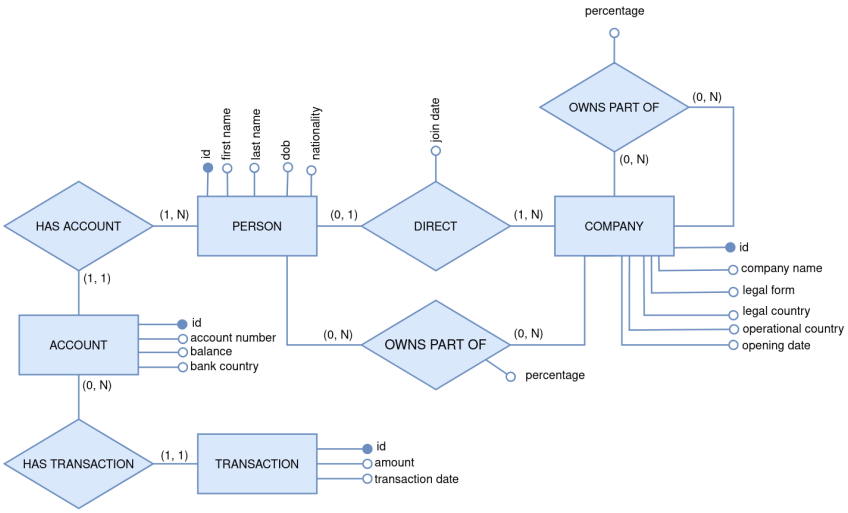


Figura 1: Datamodel

Si evincono le seguenti entità:

- *Person*, caratterizzata da un ID univoco, un nome, un cognome, una data di nascita e una nazionalità;
- *Company*, caratterizzata da un ID univoco, dal nome della compagnia, da una forma giuridica, da una sede legale, da una sede operativa e da una data di apertura;
- *Account*, caratterizzato da un ID univoco, da un numero IBAN, da un bilancio e dalla nazionalità della banca;
- *Transaction*, caratterizzata da un ID univoco, da una somma trasferita e dalla data della transazione.

Si evincono le seguenti relazioni:

- *Direct*, che lega una persona ad una compagnia. In questo modo, la persona sarà direttore di quella compagnia;
- *Owns part of*, che lega una persona o una compagnia ad un'altra compagnia. In questo modo, la persona o la compagnia possiederà una percentuale delle azioni dell'altra compagnia.
- *Has account*, che lega una persona ad un conto bancario. In questo modo, la persona possiederà uno o più account bancari;
- *Has transaction*, che lega un conto bancario ad una transazione.

4 Implementazione e inserimento

I dati presenti nei database in esame sono stati generati da uno script in Python appositamente realizzato. Lo script è progettato per generare dati realistici in modo pseudo-casuale, utilizzando il modulo *Faker*, e inserirli nei file CSV con cui sono alimentati i database di Oracle SQL Developer e Neo4j. I dataset presentano la seguente struttura:

	25%	50%	75%	100%
Persons	7500	15000	22500	30000
Companies	1250	2500	3750	5000
Accounts	7500	15000	22500	30000
Transactions	20000	40000	60000	80000

Tabella 1: Dataset generati

4.1 Oracle SQL Developer

In Oracle SQL Developer, le entità sono organizzate in **tabelle**, le quali contengono attributi specifici che descrivono le entità. Le relazioni, che non possono essere espresse direttamente in modo naturale, vengono gestite dal progettista del database che definisce una struttura in grado di implementare relazioni concettuali tra le entità.

Per importare i dati è stata utilizzata l'impostazione *import* da interfaccia grafica che permette la creazione di tabelle e l'importazione dei dati da file CSV.

4.2 Neo4j

In Neo4j, le entità vengono rappresentate come etichette (**labels**). Per gestire i database Neo4j e caricare i file CSV generati, è stato utilizzato **Neo4j Desktop**, il quale offre un'interfaccia intuitiva per la creazione dei database.

Nell'applicazione desktop sono stati creati quattro database Neo4j, ciascuno corrispondente a una diversa dimensione del dataset (25%, 50%, 75%, 100%). Per ogni database, è necessario inserire i file CSV nella corrispettiva cartella *import*. Di seguito viene riportato l'esempio di query che crea la relazione tra le persone e i direttori.

```
LOAD CSV WITH HEADERS
FROM 'file:///directors_25.csv'AS row
MATCH (p:Person {id: row.person_id})
MATCH (c:Company {id: row.company_id})
CREATE (p)-[:DIRECTS {join_date: date(row.join_date)}]->(c);
```

5 Test

Le query eseguite sui due database sono quattro, progettate per mostrare il comportamento dei due DBMS al crescere della complessità delle query e della dimensione del dataset, mettendo in evidenza i loro punti di forza e debolezza. Le query vengono eseguite su entrambi i database tramite script Python. Al termine dell'esecuzione dello script, vengono registrati i tempi di esecuzione, salvati in un file CSV con il seguente formato:

- Tempo della prima esecuzione
- Tempi delle successive 30 esecuzioni
- Media dei tempi delle 30 esecuzioni

Dopo la prima esecuzione, effettuata in un ambiente appena configurato, vengono eseguite ulteriori 30 misurazioni per confrontare i meccanismi di **caching** dei due DBMS, che rendono l'esecuzione della stessa query più veloce nel tempo. Le misurazioni raccolte sono state successivamente importate in un foglio di calcolo, da cui è stato possibile creare due tipologie di istogrammi per sintetizzare i risultati di ogni query. Il primo istogramma mostra i **tempi della prima esecuzione** della query, mentre il secondo mostra la **media delle 30 esecuzioni successive**, includendo l'intervallo di confidenza al 95%.

```
# Query Neo4j

def execute_query(query, query_num):
  filename = f"times_query_{query_num}.csv"
  times = []

  with driver.session() as session:
    for i in range(31):
      start_time = time.perf_counter()
      try:
        session.run(query)
      except Exception as e:
        times.append(float('nan'))
        continue
      end_time = time.perf_counter()
      execution_time = end_time - start_time
      times.append(execution_time)
```

```
# Query Oracle SQL Developer

def execute_query(query, query_num):
  filename = f"times_query_{query_num}.csv"
  times = []

  for i in range(31):
    start_time = time.perf_counter()
    cursor = connection.cursor()
    cursor.execute(query)
    cursor.close()

    end_time = time.perf_counter()
    execution_time = end_time - start_time
    times.append(execution_time)
```

5.1 Query 1

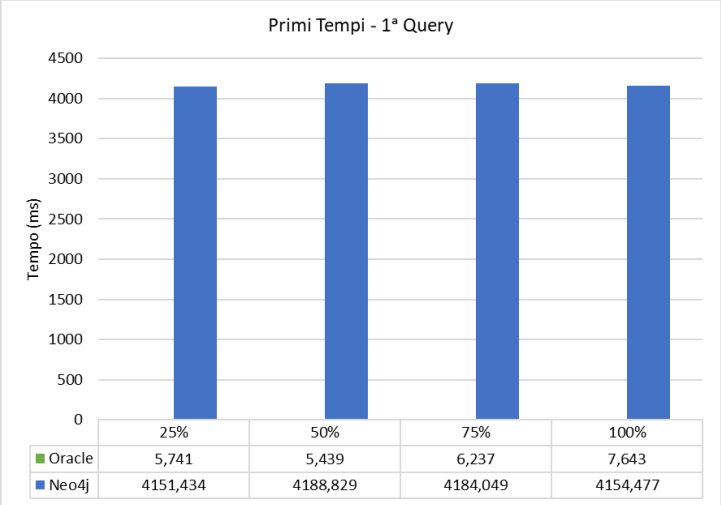
Questa query recupera tutti gli account italiani ordinati per *balance*.

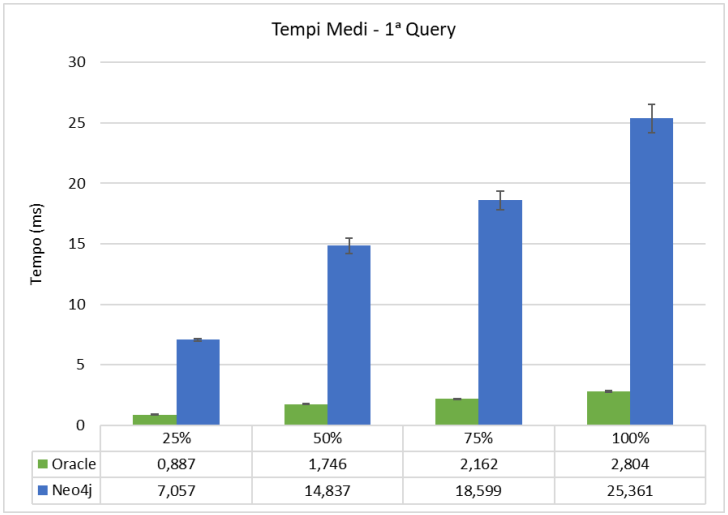
```
// Query Neo4j

MATCH (a:Account)
WHERE a.bank_country = "IT"
RETURN a.balance AS Balance
ORDER BY a.balance DESC
LIMIT 5;
```

```
// Query Oracle SQL Developer

SELECT balance AS Balance
FROM Accounts
WHERE bank_country = 'IT'
ORDER BY balance DESC
FETCH FIRST 5 ROWS ONLY;
```




Tempi Medi - 1ª Query
Tempo (ms)
30
25
20
15
10
5
0
25%
50%
75%
100%
Oracle
0,887
1,746
2,162
2,804
Neo4j
7,057
14,837
18,599
25,361

5.2 Query 2

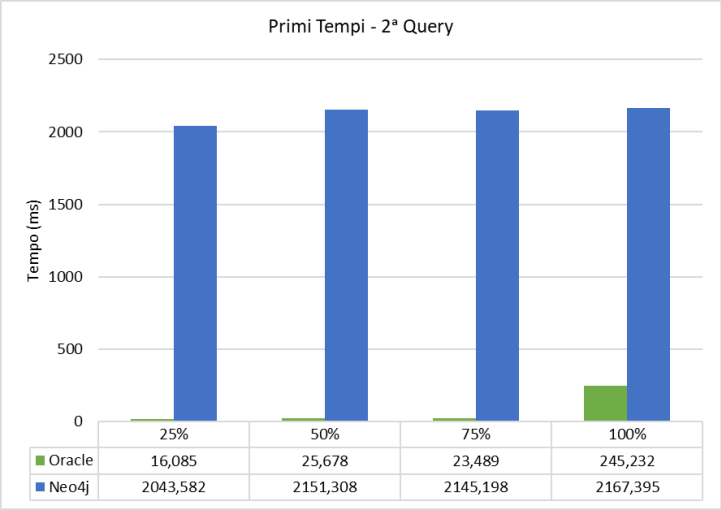
Ottiene nome, cognome e IBAN delle persone che dirigono un'azienda (in seguito: *direttore*) e li mostra in ordine di bilancio.

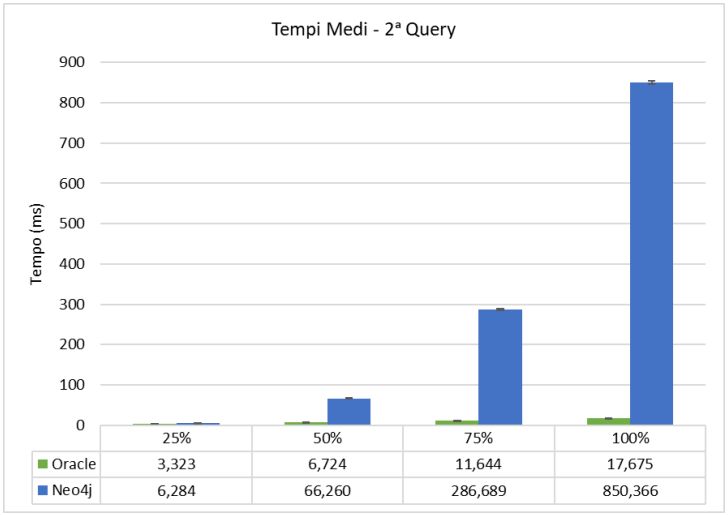
```
// Query Neo4j

MATCH (:Company)<-[:DIRECTS]-(d:Person)-[:HAS_ACCOUNT]->(a
    :Account)
RETURN d.first_name AS Name,
d.last_name AS Surname,
a.account_number AS IBAN,
a.balance AS Balance
ORDER BY Balance DESC;
```

```
// Query Oracle SQL Developer

SELECT p.first_name AS Name,
  p.last_name AS Surname,
  a.account_number AS IBAN,
  a.balance AS Balance
FROM Persons p
JOIN Directors d ON p.id = d.person_id
JOIN Accounts a ON p.id = a.owner_id
ORDER BY a.balance DESC;
```




Tempi Medi - 2ª Query
Tempo (ms)
900
800
700
600
500
400
300
200
100
0
25%
50%
75%
100%
Oracle
3,323
6,724
11,644
17,675
Neo4j
6,284
66,260
286,689
850,366

5.3 Query 3

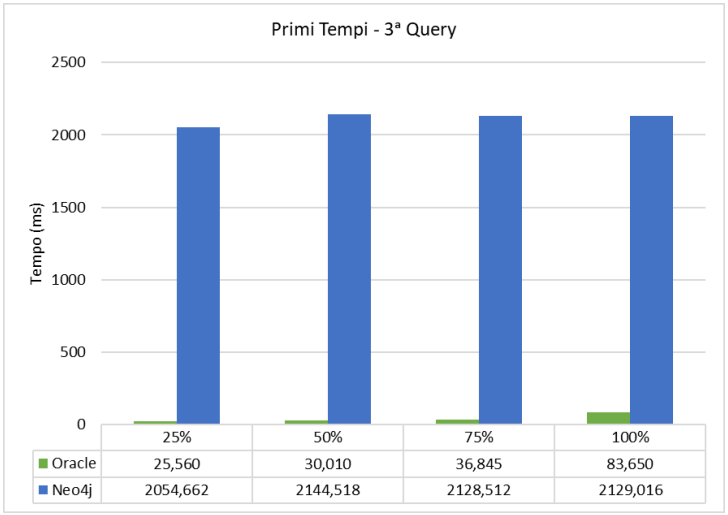
Ottiene IBAN del mittente, IBAN del destinatario, somma inviata e data dell'operazione delle transazioni avvenute nello stesso Paese.

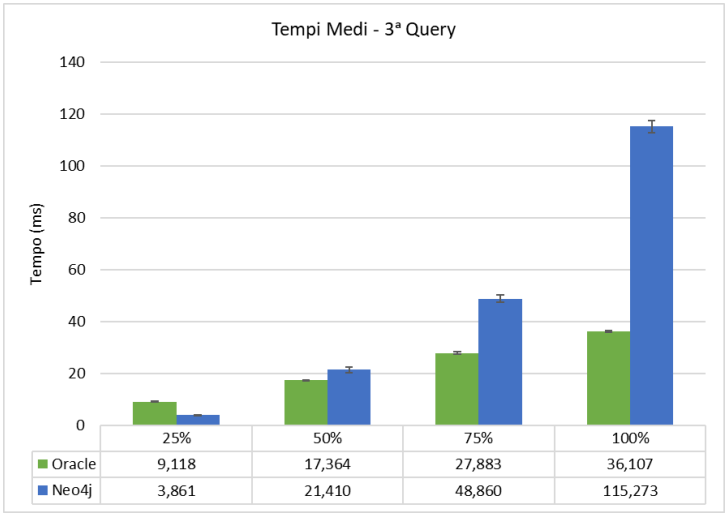
```
// Query Neo4j

MATCH (s:Account)-[t:HAS_TRANSACTION]>(r:Account)
WHERE s.bank_country = r.bank_country
AND t.amount > 600000
RETURN s.account_number AS Sender,
r.account_number AS Receiver,
t.amount AS Amount,
t.transaction_date AS Date
ORDER BY Date DESC;
```

```
// Query Oracle SQL Developer

SELECT
  s.account_number AS Sender,
  r.account_number AS Receiver,
  t.amount AS Amount,
  t.transaction_date AS Transaction_date
FROM Transactions t
JOIN Accounts s ON t.sender_id = s.id
JOIN Accounts r ON t.receiver_id = r.id
WHERE s.bank_country = r.bank_country
AND t.amount > 600000
ORDER BY t.transaction_date DESC;
```


Primi Tempi - 3ª Query
Tempo (ms)
2500
2000
1500
1000
500
0
25%
50%
75%
100%
Oracle
25,560
30,010
36,845
83,650
Neo4j
2054,662
2144,518
2128,512
2129,016


Tempi Medi - 3ª Query
Tempo (ms)
140
120
100
80
60
40
20
0
25%
50%
75%
100%
Oracle
9,118
17,364
27,883
36,107
Neo4j
3,861
21,410
48,860
115,273

5.4 Query 4

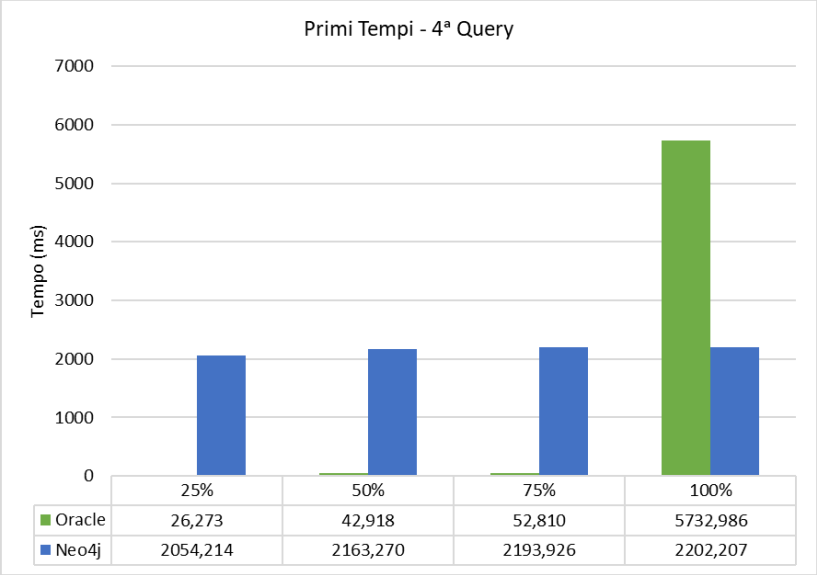
Ottiene nome e cognome dell'UBO che possiede almeno il 50% delle azioni, la percentuale di possesso e il nome, il cognome e il bilancio del direttore di quell'azienda con bilancio superiore a 250.000.

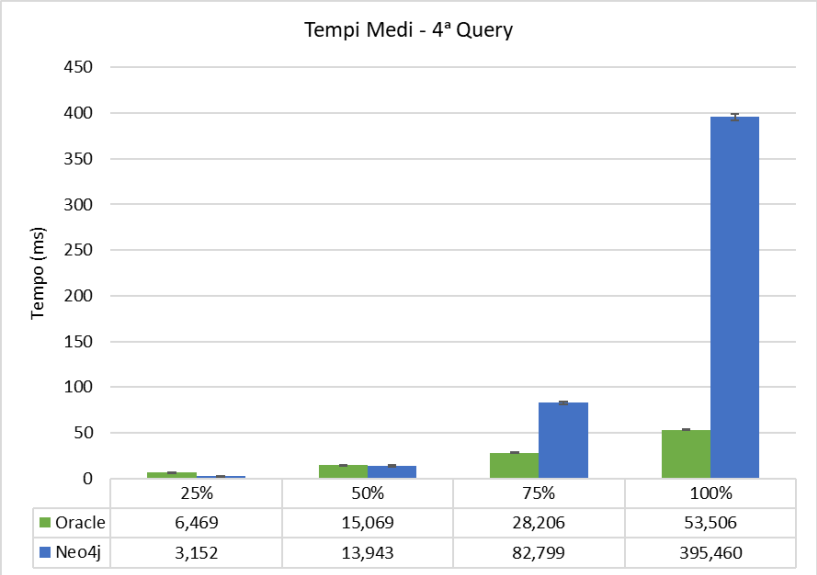
```
// Query Neo4j

MATCH (p:Person)-[owns:OWNS_PART_OF]->(c:Company)<-[:
    DIRECTS]-(d:Person)-[:HAS_ACCOUNT]->(a:Account)
WHERE owns.percentage > 0.50
AND d.nationality <> p.nationality
AND a.balance > 250000
RETURN p.first_name AS UBO_name,
p.last_name AS UBO_surname,
owns.percentage AS percentage,
d.first_name AS Director_name,
d.last_name AS Director_surname,
a.balance AS Director_balance
ORDER BY percentage DESC;
```

```
// Query Oracle SQL Developer

SELECT
  ubo.first_name AS UBO_name,
  ubo.last_name AS UBO_surname,
  sh.percentage AS percentage,
  dir.first_name AS Director_name,
  dir.last_name AS Director_surname,
  acc.balance AS Director_balance
FROM Shareholders sh
JOIN Persons ubo ON sh.entity_id = ubo.id
JOIN Companies c ON sh.company_id = c.id
JOIN Directors d ON c.id = d.company_id
JOIN Persons dir ON d.person_id = dir.id
JOIN Accounts acc ON dir.id = acc.owner_id
WHERE sh.percentage > 0.5
AND ubo.nationality <> dir.nationality
AND acc.balance > 250000
ORDER BY sh.percentage DESC;
```


Primi Tempi - 4ª Query
Tempo (ms)
7000
6000
5000
4000
3000
2000
1000
0
25%
50%
75%
100%
Oracle
26,273
42,918
52,810
5732,986
Neo4j
2054,214
2163,270
2193,926
2202,207


Tempi Medi - 4ª Query
Tempo (ms)
450
400
350
300
250
200
150
100
50
0
25%
50%
75%
100%
Oracle
6,469
15,069
28,206
53,506
Neo4j
3,152
13,943
82,799
395,460

6 Conclusioni

Dai dati presentati emerge un'evidente differenza dei tempi di esecuzione delle query effettuate in un ambiente privo di dati nella memoria cache e successivamente quando i suddetti dati sono presenti.

Si può evincere che Neo4j risulti sempre in **svantaggio** in termini di tempi di esecuzione rispetto ad Oracle SQL Developer, soprattutto all'aumentare della mole di dati.

Nel caso della prima query, la differenza tra i due DBMS è marcata in mancanza di cache; nelle successive iterazioni (con utilizzo della cache) i tempi di risposta su Neo4j si riducono notevolmente ma non raggiungono i risultati di Oracle SQL Developer, con tempi di risposta **quasi nulli**. Anche nella seconda query Oracle SQL Developer mostra tempi di esecuzione inferiori.

Nella terza query Oracle SQL Developer presenta nuovamente una netta superiorità nei test not cached, ma nei test cached Neo4j risponde meglio di Oracle SQL Developer sul dataset più piccolo, ma ottiene risultati peggiori all'aumentare della mole di dati, riducendo però il distacco rispetto ai risultati precedenti.

Infine, nella quarta query, nei test not cached, l'efficienza di Neo4j **migliora** all'aumentare dei dati, con risultati migliori di Oracle SQL Developer; Nei test con cache entrambi i database presentano inizialmente pari prestazioni , ma all'aumentare del carico computazionale Neo4j ottiene risultati **peggiori** rispetto ad Oracle.

Questi risultati sono perfettamente compatibili con la natura dei due DBMS. Oracle SQL Developer è infatti ottimizzato per operazioni su dati tabellari e relazionali (join, aggregazioni, ecc.) come quelli in esame. Neo4j, invece, è ottimizzato per dati di tipo grafico, dove le relazioni e le connessioni tra i nodi sono predominanti. Alla luce di ciò, si può concludere che nel caso di studio presentato, Oracle SQL Developer risulta **notevolmente efficiente** rispetto a Neo4j quando si hanno delle relazioni, anche su grossi dataset.