Università degli Studi di Messina

Dipartimento MIFT

Tesina Basi di Dati 2

"Indagine sui contratti pubblici concessi in Europa nel 2015"

Studente
Dario Famà

Docente
Prof. Antonio Celesti

Indice

2

Problematica affrontata

Lo scopo di questo progetto è effettuare un confronto in termini di prestazioni tra due diversi DBMS NoSQL, mettendo a confronto una problematica reale comune. Andremo a valutare i tempi di risposta di ciascun DBMS, al variare della dimensione del dataset e andando a considerare delle query di difficoltà crescente,al fine di stabilire quale delle due soluzioni prese in esame ci permette di avere delle migliori performance.
Il dataset preso in esame, contiene delle informazioni molto importanti, che riguardano l'assegnazione dei contratti pubblici in Europa nel 2015.
Ogni anno, più di 1,3 trilioni di euro di contratti sono aggiudicati da enti pubblici in Europa. Nel tentativo di rendere più trasparenti questi appalti pubblici, l'Unione europea ha deciso di rendere pubbliche le offerte. Le informazioni possono essere trovate online attraverso il portale di dati aperti dell'UE.

OpenTED, un'organizzazione no profit, ha compiuto un ulteriore passo avanti e ha reso i dati disponibili in un formato **CSV**.
L'Unione europea dà libero accesso a informazioni dettagliate sui contratti di acquisto pubblici. Questi dati descrivono quali istituzioni europee stanno spendendo denaro, per cosa e chi ne beneficia.

Come vedremo nel dettaglio in seguito, le entità coinvolte in questa realtà sono:

- **Authority**,rappresentano tutte le informazioni riguardanti l'ente che si occupa di gestire al meglio l'aggiudicazione di ogni singolo contratto
- **Contract**, rappresentano tutte le informazioni riguardanti ogni singolo contratto.
- **Contract Award Notice**(Avviso di aggiudicazione dei contratti), rappresentano le informazioni relative all'avviso di aggiudicazione del contratto, tra cui la tipologia del contratto in questione.
- **Contract Award**(Aggiudicazione del contratto) rappresenta tutte le informazioni relative riguardanti l'aggiudicazione vera e propria, tra cui il nome, il paese e la città del vincitore,il prezzo d'aggiudicazione e cosi via.
- **Lots** (Lotti) rappresenta le informazioni riguardanti i lotti che possono essere coinvolti nell'aggiudicazione di un contratto.

Cosa significa NoSQL?

Prima di andare a vedere nel dettaglio le due soluzioni prese in esame,
analizziamo rapidamente cosa si intende con database NOSQL.
NoSQL è l'acronimo di "Not only SQL" e viene utilizzato generalmente per indicare i database che non si basano sul tradizionale modello di dati relazionale e che quindi potrebbero non avere SQL come linguaggio di interrogazione. Questa generica definizione può essere ampliata dalle caratteristiche che hanno in comune:

● **Memorizzano i dati in formati diversi**. I RDBMS memorizzano i dati in tabelle, formate da righe e colonne. I database NoSQL possono utilizzare diversi formati come archivi di documenti, grafi, archivi chiave-valore e cosi via
● **Non utilizzano le join**. I database NoSQL sono in grado di estrarre i dati utilizzando semplici interfacce orientate ai documenti senza utilizzare join SQL.
● **Rappresentazione dei dati senza uno schema fisso** (schemaless).
Le implementazioni NoSQL si basano su una rappresentazione di dati schemaless (ad eccezione del database Cassandra). Con questo approccio non è necessario definire i dati in anticipo, e questi possono quindi continuare a cambiare nel tempo.

Classificazione dei sistemi NoSQL

I sistemi NoSql si suddividono:
● **Key-Values stores :** Il modello a chiave-valore si basa su una API analoga ad una mappa, accessibile tramite la chiave. Il valore può contenere sia dati elementari, che dati avanzati. Può convenire utilizzarli quando non è possibile definire uno schema sui dati ed è necessario un accesso rapido alle singole informazioni. Le interrogazioni si effettuano sulle chiavi (che possono essere indicizzate) e si ottiene il valore.
Viene spesso utilizzato per memorizzare informazioni che non presentano correlazioni, ad esempio per il salvataggio delle sessioni degli utenti in ambito web.
● **Column-oriented** : Vengono chiamati in questo modo perché organizzano memorizzano i dati per colonne. Ogni riga può avere un insieme diverso di *colonne, poiché possono essere aggiunte quelle necessarie e tolte quelle inutilizzate, evitando co*sì la presenza di valori null. Consente la compressione delle informazioni e il versioning. Un utilizzo tipico è l'indicizzazione di pagine web: possiedono un testo, che può essere compresso, e cambiano nel tempo, beneficiando così del versioning.
Document database: I database orientati ai documenti sono caratterizzati da una struttura fondamentale, detta document, di solito scritta in JSON, costituita da un identificatore univoco e da un qualsiasi altro numero di attributi di qualunque tipo (purché esprimibili come un documento), anche nidificato. Sono utili quando i dati variano nel tempo, e possono mappare correttamente gli oggetti nel modello OOP.
● **Graph database** : I database a grafo memorizzano grafi (nodi e collegamenti tra nodi) e sono adatti a rappresentare dati fortemente interconnessi tra loro e possono effettuare interrogazioni mediante un attraversamento efficiente della struttura. Rispetto ad una normale query di altri tipi di database, si può velocizzare il cammino da un nodo ad un altro aggiungendo un collegamento diretto tra i due (con costo unitario dell'operazione).

Soluzioni DB considerate

Le soluzioni di database NOSQL che sono state considerate sono le seguenti:

- Neo4j
- Cassandra

Neo4j

Neo4j appartiene alla categoria dei graph database, è open source ed prodotto dalla software house Neo Technology.
Il data model è un grafo orientato con proprietà chiave-valore, che prevede:

- **nodi**, che rappresentano le entità
- **relazioni** dette anche archi,che esprimono le associazioni tra entità

Per quanto riguarda le **caratteristiche dei nodi**, il modello prevede che ognuno di essi:

- Abbia un ID univoco assegnato automaticamente da Neo4j al momento della creazione.
- Possa avere una o più Label (etichette) che servono a classificare i nodi e indicizzarli.

Ogni relazione invece :
-Ha necessariamente un tipo(etichetta), che deve essere specificata dall'utente al momento della creazione.
-Ha necessariamente una direzione, ossia una relazione va da un nodo ad un altro e non è possibile creare relazioni senza verso. È possibile però fare query su relazioni indipendentemente dalla direzione delle relazioni, ad esempio per trovare i nodi che hanno la relazione "AMICO_DI" un certo nodo, indipendentemente dal fatto che le relazioni arrivano o partono da tale nodo. Il modello si presta a molti scenari di utilizzo.

Generalmente, le Label si usano per raggruppare entità dello stesso tipo: ad esempio, se uso Neo4j come database di un social network, potrei avere dei nodi con Label "Utente", altri con label "Gruppi", "Messaggi", etc. Un nodo può avere più Label, aprendo la strada ad applicazioni interessanti, implementando una sorta di polimorfismo. I nodi con una certa Label possono essere indicizzati su certe proprietà, per velocizzarne la ricerca.

E' stato sviluppato interamente in java ed presenta le seguenti caratteristiche:

- robusto
- scalabile
- alte prestazioni
- dotato di transazioni ACID
- alta disponibilità
- tecniche di memorizzazione per miliardi di nodi e relazioni,
- alta velocità di interrogazione tramite attraversamenti

Inoltre è dotato linguaggio di interrogazione dichiarativo e grafico (**Cypher**), dispone di numerosi driver che gli permettono di interfacciarsi con numerosi linguaggi di programmazione.
E' un DBMS schemaless, ciò sta a significare che i suoi dati non devono attenersi ad alcuna struttura di riferimento prefissata.
Innanzitutto, esso si presta a modellare situazioni che hanno intrinsecamente un modello a grafo, come ad esempio un'infrastruttura di una rete aziendale. Inoltre, grazie alla facilità di navigazione all'interno del grafo, questo modello è adatto a casi in cui siano necessarie ricerche semantiche, ad esempio nei sistemi ***di rilevazione di frodi.***

Cassandra è un database NoSQL, originariamente sviluppato da Facebook per potenziare la ricerca all'interno del sistema di posta e tuttora utilizzato. Nel 2008 sono stati resi disponibili i sorgenti su Google Code, mentre dal 2009 il progetto ha cominciato ad essere distribuito sotto la Apache Licence 2 (una licenza di software libero della Apache Software Foundation (ASF) che obbliga gli utenti a preservare l'informativa di diritto d'autore e d'esclusione di responsabilità nelle versioni modificate).
Dal punto di vista del teorema CAP, Cassandra è caratterizzato da due proprietà:

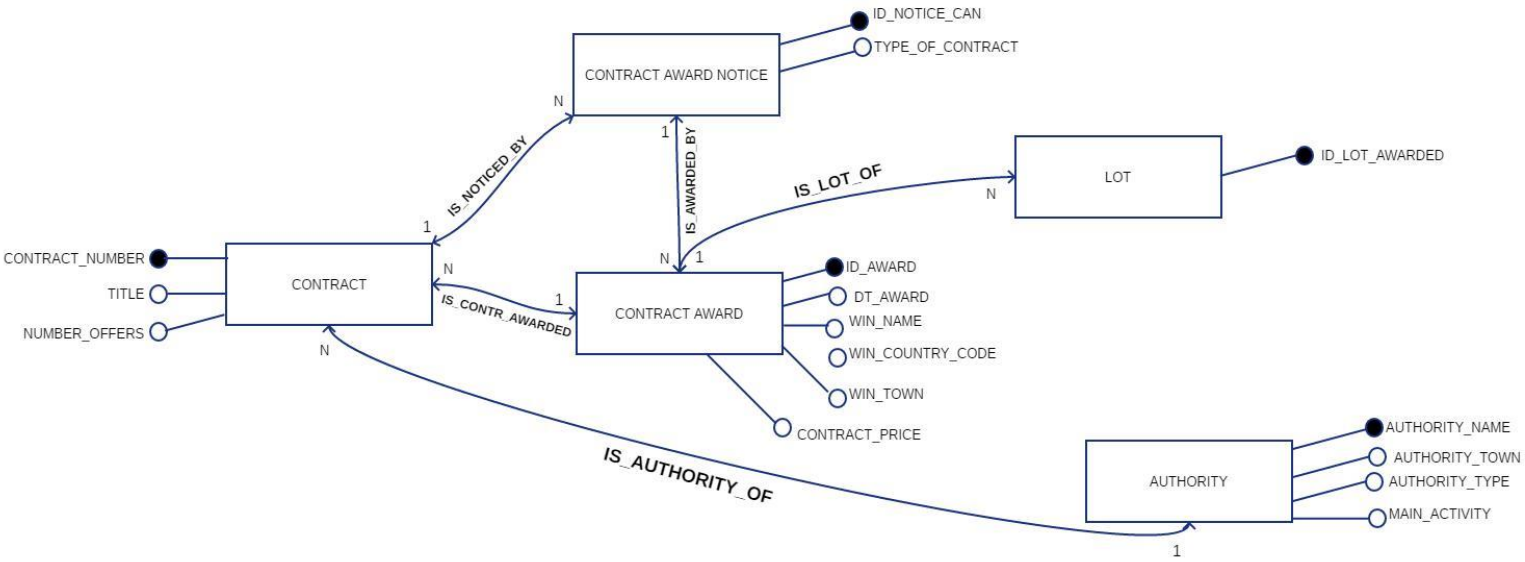
- **Availability (disponibilità):** ad una richiesta, il sistema è sempre in grado di dare una risposta, in altre parole, il sistema è sempre disponibile.
- **Partition Tolerance (tolleranza di partizione):** se le comunicazioni si interrompono tra due punti del sistema, il sistema non fallisce ma continua ad essere disponibile.

Per questi aspetti è simile ad altri database NoSQL, come CouchDB. Ma ci sono anche diverse differenze o specificità:

- L'interrogazione in Cassandra non viene eseguita su HTTP.
- Ciascun linguaggio ha un proprio driver nativo per la connessione diretta al database;
- Cassandra è decentralizzato: i nodi nel cluster sono identici e non esiste alcun single point of failure (una componente che in caso di malfunzionamento o anomalia causa la disfunzione dell'intero sistema).
- La modalità di storage di Cassandra è un incrocio tra un archivio di chiavi/valori e un database a tabelle; ogni chiave si associa a una o più colonne e tali colonne possono essere raggruppate in famiglie di colonne.
- Cassandra è Fault-tolerant, i dati vengono replicati automaticamente su più nodi. È supportata la replica mediante diversi data center, e la sostituzione dei nodi può essere effettuata senza alcun downtime.
- Il livello di consistenza (sia in scrittura che in lettura) può essere modificato, ma a scapito della disponibilità dei dati.
- Scalabilità: I nodi hardware possono essere aggiunti a Cassandra senza disservizi e quindi senza perdita di tempi aggiuntivi. Cassandra è quindi facilmente scalabile: man mano che l'applicazione cresce, è possibile aggiungere altro hardware e Cassandra continuerà a funzionare.
- Per effettuare delle query, Cassandra fornisce il proprio linguaggio di query (specificamente progettato per i gruppi di colonne), che è simile a SQL.

Progettazione del data model relativo al progetto

Partendo dalla pagina ufficiale di **LINKURIOUS** , si è cercato di comprendere in profondità la natura dei dati presi in esame.
Tuttavia, il dataset disponibile per il download aveva subito numerose variazioni rispetto al contesto originale , quindi consultando la documentazione ufficiale, si è rivelata molto utile, la stesura di una nuovo **diagramma entità-relazioni** che permettesse di avere una **linea guida** per rappresentare efficacemente le **entità** e le **relazioni** presenti tra di esse, al fine di rappresentare al meglio la realtà presa in esame. Come già detto questo schema rappresenta solo una **linea guida;** lavorando infatti con database di tipo NoSQL, questo grafico non risulterà vincolante per la definizione del modello logico di ciascun DB utilizzato.



Entità

- **Authority** (**Authority_name**, Authority_town, Authority_type, Main_activity)
- **Contract** (**Contract_number**, Title, Number_offers)
- **Contract_award_notice** (**Id_notice_can**, type_of_contract)
- **Contract_award (Id_award,** win_name, win_country_code, win_town, contract_price)
- **Lot** (**Id_lot_awarded**)

Relazioni

- **Is_Authority_Of** (Associazione tra Authority e Contract, 1 a molti)
- **Is_Noticed_By** (Associazione tra Contract e Contract Award Notice, 1 a molti)
- **Is_Awarded_By** (Associazione tra Contract Award Notice e Contract_Award, 1 a molti)
- **Is_Contract_Awarded** (Associazione Contract Award e Contract, 1 a molti)
- **Is_Lot_Of (**Associazione tra Contract Award e Lot, 1 a molti)

NEO4J

In Neo4j, per quanto riguarda la creazione del data model e l'importazione dei dati stessi , non è stato possibile utilizzare lo script presente nel link di riferimento al progetto, poiché il dataset disponibile per il download nel corso degli anni ha subito numerose modifiche.
E' stato dunque necessario riscrivere degli script nel linguaggio nativo di Neo4j(Cypher).

//Il comando **CONSTRAINT** ci permette di definire dei vincoli di unicità sulle chiavi primarie
//Il comando **LOAD CSV** ci permette di importare il dataset in formato csv.
//Il comando **MERGE** può essere immaginato come un tentativo di effettuare una corrispondenza sulla struttura dati, e se questa corrispondenza non esiste la crea;quindi può essere considerata una combinazione dei comandi **MATCH** e **CREATE**.
//Il comando **ON CREATE SET** ci permette di stabilire i campi da estrarre dal file csv.
//Il comando **MATCH** utilizzato nella creazione delle relazioni,ci permette di stabilire un legame tra le chiavi delle entità coinvolte nella relazione.

//Creazione Entità Contract

```
CREATE CONSTRAINT ON (a: CONTRACT) ASSERT a.contract_number IS UNIQUE;

USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
WITH line
WHERE line.CONTRACT_NUMBER IS NOT NULL
MERGE (a:CONTRACT {contract_number: line.CONTRACT_NUMBER})

ON CREATE SET a.contract_number =
                line.CONTRACT_NUMBER, a.title= line.TITLE,
                a.number_offers= line.NUMBER_OFFERS
```

//Creazione Entità Authority

```
CREATE CONSTRAINT ON (a:AUTHORITY) ASSERT a.authority_name IS UNIQUE;

USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
WITH line
WHERE line.CAE_NAME IS NOT NULL
MERGE (a:AUTHORITY {authority_name: line.CAE_NAME})

ON CREATE SET a.authority_name = line.CAE_NAME,
                a.autority_town= line.CAE_TOWN,
                a.autority_type= line.CAE_TYPE,
                a.main_activity= line.MAIN_ACTIVITY
```

//Creazione Entità CONTRACT_AWARD_NOTICE

```
CREATE CONSTRAINT ON (a:CONTRACT_AWARD_NOTICE) ASSERT a.id_notice_can
IS UNIQUE;

USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
WITH line
WHERE line.ID_NOTICE_CAN IS NOT NULL
MERGE (a:CONTRACT_AWARD_NOTICE {id_notice_can: line.ID_NOTICE_CAN})

ON CREATE SET a.id_notice_can = line.ID_NOTICE_CAN,
                    a.type_of_contract= line.TYPE_OF_CONTRACT
```

//Creazione Entità CONTRACT_AWARD

```
CREATE CONSTRAINT ON (a:CONTRACT_AWARD) ASSERT a.id_award IS UNIQUE;

USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
WITH line
WHERE line.ID_AWARD IS NOT NULL
MERGE (a:CONTRACT_AWARD {id_award: line.ID_AWARD})

ON CREATE SET a.id_award = line.ID_AWARD,
                    a.win_name= line.WIN_NAME,
                    a.dt_award= line.DT_AWARD,
                    a.win_country_code= line.WIN_COUNTRY_CODE,
                    a.win_town= line.WIN_TOWN,
                    a.contract_price=
```

---

line.AWARD_VALUE_EURO_FIN_1 **//Creazione Entità LOT**

```
CREATE CONSTRAINT ON (a:LOT) ASSERT a.id_lot_awarded IS
UNIQUE; USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
WITH line
WHERE line.ID_LOT_AWARDED IS NOT NULL
MERGE (a:LOT {id_lot_awarded: line.ID_LOT_AWARDED})

ON CREATE SET a.id_lot_awarded = line.ID_LOT_AWARDED
```

//Relazione tra contract e authority

```
USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
MATCH (b:CONTRACT {contract_number: line.CONTRACT_NUMBER})
MATCH (a:AUTHORITY {authority_name: line.CAE_NAME})
MERGE (a)-[r:IS_AUTHORITY_OF]->(b);
```

//Relazione tra contract award notice e contract award

```
USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
MATCH (b:CONTRACT_AWARD {id_award: line.ID_AWARD})
MATCH (a:CONTRACT_AWARD_NOTICE {id_notice_can:
line.ID_NOTICE_CAN}) MERGE (a)-[r:IS_AWARDED_BY]->(b);
```

//Relazione tra lotto e contract award

```
USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
MATCH (b:LOT {id_lot_awarded: line.ID_LOT_AWARDED})
MATCH (a:CONTRACT_AWARD {id_award: line.ID_AWARD})
MERGE (a)-[r:IS_LOT_OF]->(b);
```

//Relazione tra contract award notice e contract

```
USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
MATCH (b:CONTRACT_AWARD_NOTICE {id_notice_can: line.ID_NOTICE_CAN})
MATCH (a:CONTRACT {contract_number: line.CONTRACT_NUMBER})
MERGE (a)-[r:IS_NOTICED_BY]->(b);
```

//Relazione tra contract e contract_award

```
USING PERIODIC COMMIT 2000
LOAD CSV WITH HEADERS FROM "file:///50k.csv" AS line
FIELDTERMINATOR ';'
MATCH (b:CONTRACT {contract_number: line.CONTRACT_NUMBER})
MATCH (a:CONTRACT_AWARD {id_award: line.ID_AWARD})
MERGE (a)-[r:IS_CONTRACT_AWARDED]->(b);
```

//Set del contract price ad intero

```
MATCH (n:CONTRACT_AWARD)
SET n.contract_price = toInt(n.contract_price;
```

Cassandra: Modello column-oriented

Il modello dati prevede un'organizzazione che parte dal cluster come entità di più alto livello e scende fino alla singola colonna.
Gli elementi che ne fanno parte sono i seguenti:

- **Cluster:** l'insieme dei server che costituiscono l'istanza di cassandra.
- **Keyspace:** un namespace per le column family.
- **Column family:** è un contenitore di colonne (o supercolumn) è l'equivalente della tabella in un RDBMS.
- **Column:** è definita attraverso un nome e, per ciascuna riga, contiene un valore e un timestamp. Quest'ultimo è utilizzato per la risoluzione dei conflitti.
- **Supercolumn:** la supercolumn è una colonna che a sua volta contiene altre colonne.
- **Row:** la riga è identificata da una chiave, la **row key** (che non può essere più grande di 64 KB), e contiene valori appartenenti a più colonne. Ciascuna riga di una colonna family può contenere valori di tutte le colonne o solo di alcune.

RDBMS	**Cassandra**
RDBMS si occupa di dati strutturati	Cassandra si occupa di dati non strutturati
Ha uno schema fisso	Non ha uno schema fisso
Negli RDBMS una tabella è un array di array (Numero di righe * Numero di colonne)	In Cassandra, una tabella è una lista di "coppie chiave-valore nidificate". (N righe * N rowkey * N RowValue)
Le tabelle sono le entità di un Database.	Le tabelle o Column Family sono l'entità di un keyspace
La row è un generico record in RDBMS.	La row è un unità di replicazione in Cassandra.
La colonna è rappresentata da un attributo di una relazione	La colonna è un unità di memorizzazione su Cassandra
RDBMS supporta join e chiavi esterne	Le relazioni sono tradotte come collezioni di dati

Codice per la costruzione del DB su Cassandra

//Creazione del keyspace con fattore di replicazione 1

```
CREATE KEYSPACE Database2 WITH replication = {'class': 'SimpleStrategy',
'replication_factor': 1};
```

//Creazione della tabella dei contratti

```
CREATE TABLE Database2.contract (
CONTRACT_NUMBER text,
TITLE text,
NUMBER_OFFERS text,
PRIMARY KEY (CONTRACT_NUMBER)
);
```

//Creazione della tabella Authority

```
CREATE TABLE Database2.authority (
MAIN_ACTIVITY text,
CAE_TOWN text,
CAE_NAME text,
CAE_TYPE text,
PRIMARY KEY (CAE_NAME)
);
```

//Creazione della tabella Lot

```
CREATE TABLE Database2.lot (
ID_LOT_AWARDED text,
PRIMARY KEY (ID_LOT_AWARDED)
);
```

//Creazione della tabella Contract Award

```
CREATE TABLE Database2.contract_award (
ID_AWARD text,
WIN_NAME text,
WIN_COUNTRY_CODE text,
WIN_TOWN text,
CONTRACT_PRICE int,
PRIMARY KEY (ID_AWARD)
);
```

// Creazione della tabella Contract Award Notice

```
CREATE TABLE Database2.contract_award_notice(
ID_NOTICE_CAN text,
TYPE_OF_CONTRACT text,
PRIMARY KEY (ID_NOTICE_CAN)
);
```

// Creazione della tabella IsNoticedBy, tabella che rappresenta la relazione tra Contract e Contract Award Notice

```
CREATE TABLE Database2.isNoticedBy(
ID_NOTICE_CAN text,
TYPE_OF_CONTRACT text,
CONTRACT_NUMBER text,
TITLE text,
NUMBER_OFFERS text,
PRIMARY KEY (ID_NOTICE_CAN, CONTRACT_NUMBER)
);
```

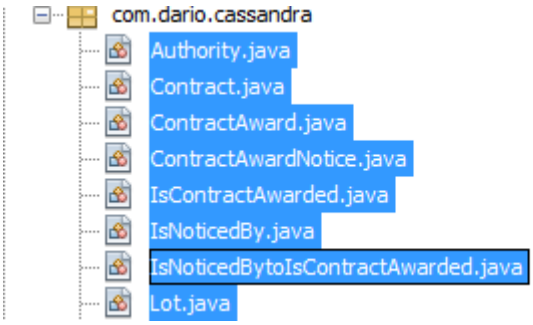
// Creazione della tabella IsContractAwarded, tabella che rappresenta la relazione tra Contract Award e Contract

```
CREATE TABLE Database2.isContractAwarded (
CONTRACT_NUMBER text,
TITLE text,
NUMBER_OFFERS text,
ID_AWARD text,
WIN_NAME text,
WIN_COUNTRY_CODE text,
WIN_TOWN text,
CONTRACT_PRICE float,
PRIMARY KEY (ID_AWARD, CONTRACT_NUMBER)
);
```

// Creazione della tabella IsNoticedBytoIsContractAwarded, tabella che rappresenta la relazione tra Contract Award e Contract insieme ai campi della relazione IsNoticedBy

```
CREATE TABLE Database2.isNoticedBytoIsContractAwarded(
ID_NOTICE_CAN text,
TYPE_OF_CONTRACT text,
CONTRACT_NUMBER text,
TITLE text,
NUMBER_OFFERS text,
ID_AWARD text,
WIN_NAME text,
WIN_COUNTRY_CODE text,
WIN_TOWN text,
CONTRACT_PRICE float,
PRIMARY KEY (ID_NOTICE_CAN, CONTRACT_NUMBER, ID_AWARD)
);
```

Upload su Cassandra



Sebbene Cassandra disponga di una propria Shell per l'invio di interrogazioni in CQL (Cassandra Query Language). L'importazione tramite Shell risulterebbe alquanto difficile se non impossibile per il nostro dataset, quindi si è scelto di creare il tutto con Java attraverso i driver messi a disposizione per l'interfacciamento con Cassandra. Nel nostro caso l'interfacciamento è stato effettuato attraverso un connettore chiamato Datastax Spark Connector, il quale richiede per funzionare altre librerie come Spark Core e Spark SQL.
Prima di importare il dataset ho dovuto settare Spark per configurarsi con il cluster di Cassandra; per fare ciò abbiamo creato un file JSON che conterrà le impostazioni di Spark in modo tale da modificarle in caso di necessità. All'avvio del programma verrà implementata una JavaSparkContex che avvierà un'istanza di Spark con le impostazioni per interfacciarsi con Cassandra. Per importare il csv ho fatto uso della libreria di appoggio Apache CSV la quale permette di eseguire il parsing di file csv in modo facile e veloce. Per caricare i dati abbiamo fatto uso dei metodi associati alla classe CassandraJavaUtil e al metodo parallelize della classe JavaSparkContex. Combinando questi due metodi è possibile implementare un istanza della classe JavaRDD per abbinargli il metodo saveToCassandra(). Un RDD (Residenti Distributed Dataset) che è la teoria alla base del sistema Spark, è essenzialmente un set di dati suddiviso in partizioni ed ha esattamente delle proprietà che gli garantiscono immutabilità e l'incapacità di creare altri RDD se non da operazioni deterministiche e ripetibili.
Riporto il metodo utilizzato per caricare i contratti nel database:

```
conf.set("spark.master", "local");
conf.set("spark.cassandra.connection.host", "localhost");
JavaSparkContext sc = new JavaSparkContext(conf);
Map<String, Contract> mappa = new HashMap<String, Contract>();
Contract contratto;
Reader reader;
CSVParser csvParser = null;
try {
    reader = Files.newBufferedReader(Paths.get(CSV_FILE_PATH), StandardCharsets.UTF_8);
    csvParser = CSVFormat.EXCEL.withFirstRecordAsHeader().withDelimiter(';').parse(reader);

    for (CSVRecord csvRecord : csvParser) {
        // Accessing values by the names assigned to each column
        String contract_number = csvRecord.get("CONTRACT_NUMBER");

        if(!contract_number.equals("")){
            contratto = mappa.get(contract_number);
            if (contratto == null) {
                contratto = new Contract();
                contratto.setContractNumber(contract_number);
                contratto.setTitle(csvRecord.get("TITLE"));
                contratto.setNumberOffers(csvRecord.get("NUMBER_OFFERS"));
                mappa.put(contract_number, contratto);
                List<Contract> listainserimento = new ArrayList<Contract>(mappa.values());
                JavaRDD<Contract> parallelize = sc.parallelize(listainserimento);
                CassandraJavaUtil.javaFunctions(parallelize)
                .writerBuilder("database2", "contract", mapToRow(Contract.class)).saveToCassandra();
            }
        }
```

Implementazione Query

Una volta inseriti i dati vengono eseguite per entrambi i DB cinque query. Riportiamo in basso il codice di Cypher per tutte le query mentre per quanto riguarda Cassandra inseriamo un piccolo esempio in quanto il codice Java di alcune query potrebbe essere troppo lungo e di difficile comprensione (alleghiamo il codice sorgente alla relazione).
E' stata realizzata un interfaccia user friendly per effettuare i benchmark anche su Neo4j:
che una volta inserite le credenziali permette di effettuare le query.

Query 1

Lista delle autorità di Beaune che hanno come attività principale la salute (Health)

Neo4j

```
MATCH (a: AUTHORITY {authority_town: 'Beaune', main_activity: 'Health'}) RETURN a
```

Cassandra

```
CassandraConnector connector = CassandraConnector.apply(sc.getConf());

try (Session session = connector.openSession()) {
        ResultSet results = session.execute("SELECT * FROM database2.authority " +
                "WHERE CAE_TOWN = 'Beaune' AND MAIN_ACTIVITY = 'Health' ALLOW FILTERING;");
}
```

Query 2

Lista dei contratti aggiudicati cn una singola offerta

Neo4j

```
MATCH (c: CONTRACT {number_offers:'1'}) RETURN c
```

Cassandra

```
CassandraConnector connector = CassandraConnector.apply(sc.getConf());


try (Session session = connector.openSession()) {

    ResultSet results = session.execute("SELECT * FROM database2.contract " +
    "WHERE NUMBER_OFFERS = '1' ALLOW FILTERING;");
}
```

Query 3

//Contratti, i cui vincitori sono residenti a Bucarest

Neo4j

MATCH (c: CONTRACT)<-[r:IS_CONTRACT_AWARDED]-(ca:CONTRACT_AWARD) WHERE ca.win_town ='Bucuresti' RETURN c,ca

Cassandra

```
the schema
n session = connector.openSession()) {
ultSet results = session.execute("SELECT CONTRACT_NUMBER, TITLE, NUMBER_OFFERS, WIN_TOWN, WIN_NAME FROM database2.iscontractawarded " +
    "WHERE WIN_TOWN = 'Bucuresti' ALLOW FILTERING;");
```

Query 4

Contratti appartenenti alla tipologia “services” aggiudicati sul territorio della

Romania

Neo4j

MATCH(c:CONTRACT)-[r:IS_NOTICED_BY]->(can:CONTRACT_AWARD_NOTICE)-[re:IS_AWARDED_BY]->(ca:CONTRACT_AWARD) WHERE can.type_of_contract='S' AND ca.win_country_code= 'RO' RETURN c,ca,can

Cassandra

```
CassandraConnector connector = CassandraConnector.apply(sc.getConf());


try (Session session = connector.openSession()) {

        ResultSet results = session.execute("SELECT * FROM database2.isnoticedbytoiscontractawarded " +
                "WHERE TYPE_OF_CONTRACT = 'S' AND WIN_COUNTRY_CODE = 'RO' ALLOW FILTERING;");

}
```

Query 5

Contratti appartenenti alla tipologia “work” aggiudicati per un prezzo superiore a

1.000.000€

Neo4j

MATCH(c: CONTRACT)-[r:IS_NOTICED_BY]->(can:CONTRACT_AWARD_NOTICE)-[re:IS_AWARDED_BY]->(ca:CONTRACT_AWARD) WHERE ca.contract_price > 1000000 AND can.type_of_contract='W' RETURN c,can,ca

Cassandra

```
CassandraConnector connector = CassandraConnector.apply(sc.getConf());
try (Session session = connector.openSession()) {
        ResultSet results = session.execute("SELECT * FROM database2.isnoticedbytoiscontractawarded " +
                "WHERE CONTRACT_PRICE > 1000000 AND TYPE_OF_CONTRACT = 'W' ALLOW FILTERING;");

}
```

ESPERIMENTI

Premessa:

Per effettuare gli esperimenti, sono stati valutati dataset di differenti dimensioni:

100 record

10.000 record

50.000 record

Si è scelto dataset di queste dimensioni a causa delle tempistiche di upload su Cassandra, che tramite interfaccia Spark, sono state molto elevate riguardo il caricamento dei dati.
Dal momento che molti DBMS NoSQL utilizzano meccanismi di caching, a parità di dimensione del dataset, si è considerato, a parte il primo tempo di esecuzione, **e il valor medio delle successive 30 esecuzioni per un totale di 31 tests**.

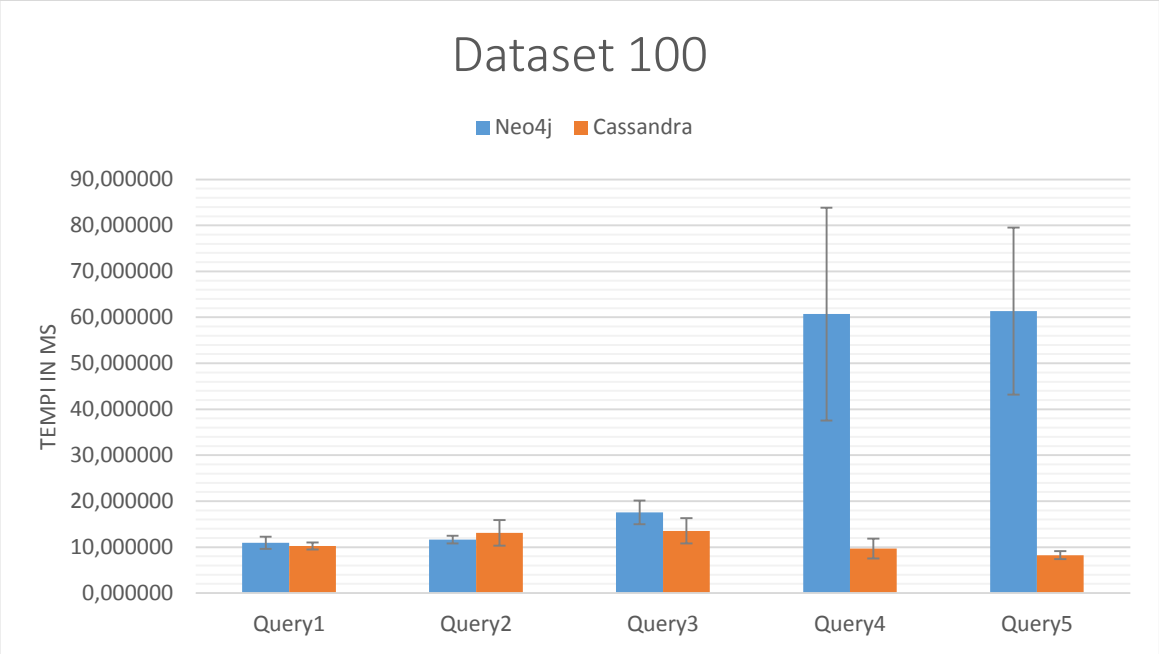
Per ogni dataset sono stati rappresentati due **istogrammi** :

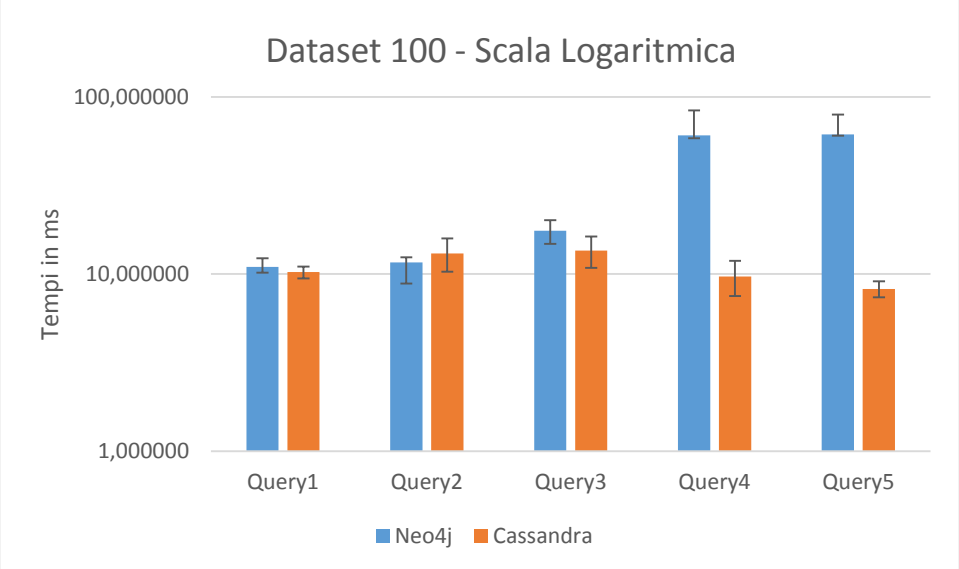
- Il primo raffigura i tempi medi di ogni query, con un intervallo di confidenza del 95%
- Il secondo raffigura i tempi medi di ogni query rappresentati su scala logaritmica e con un intervallo di confidenza del 95%

DATASET da 100

Cassandra					
Query1	Query2	Query3	Query4	Query5	
10,25058917	13,10104	13,55333	9,692292	8,246884	MEDIA
2,127080565	7,46876	7,305449	5,861193	2,314707	DEVIAZIONE STANDARD
0,794264936	2,788881	2,727899	2,188605	0,864326	INTERVALLO DI CONFIDENZA

Neo4j					
Query1	Query2	Query3	Query4	Query5	
10,977047	11,645001	17,5453482	60,70599	61,37699	MEDIA
3,537891	2,163207	6,92915575	62,04743	48,6916	DEVIAZIONE STANDARD
1,3210703	0,807755	2,58738928	23,16889	18,18174	INTERVALLO DI CONFIDENZA

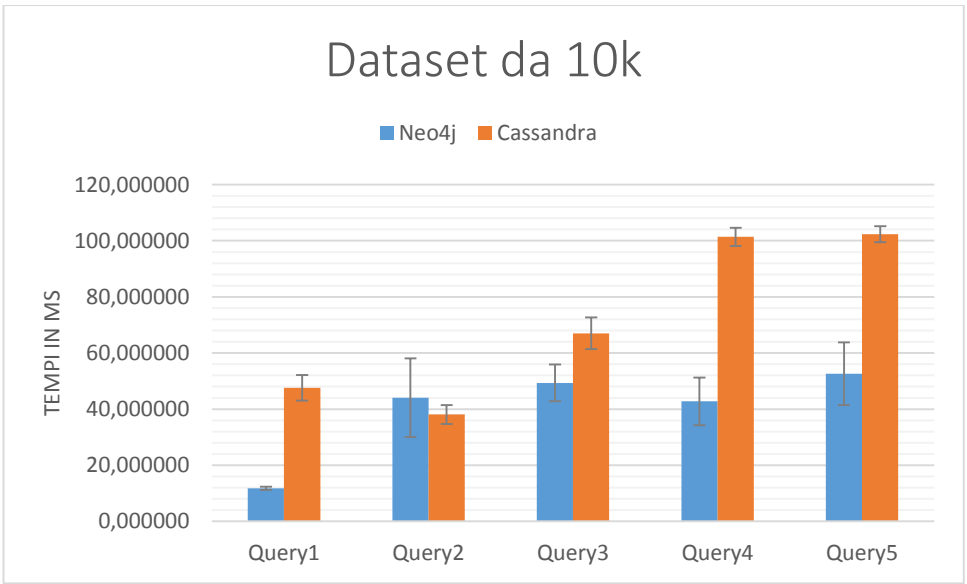


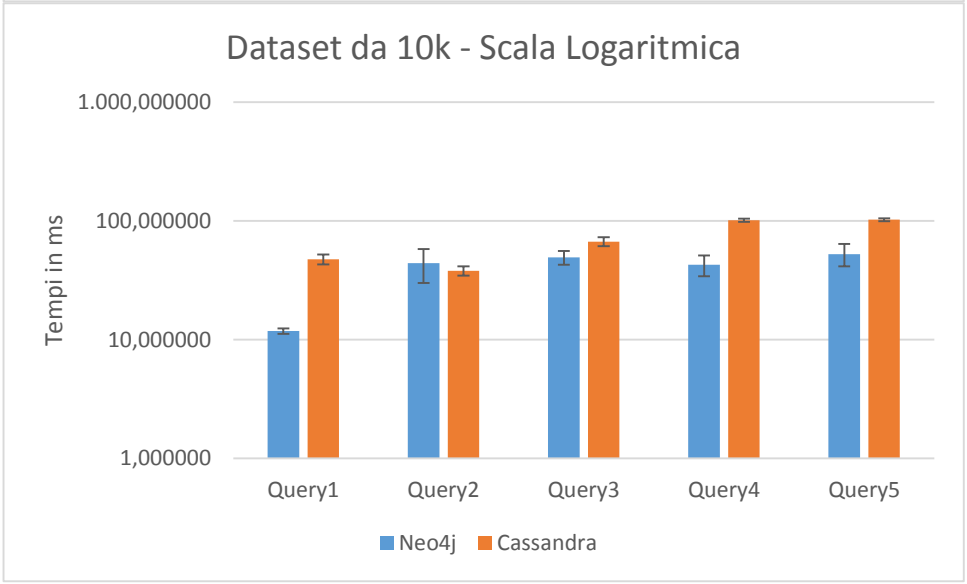


DATASET da 10k

CASSANDRA					
Query1	Query2	Query3	Query4	Query5	
47,584385	38,07555	67,01346437	101,4064	102,3523	MEDIA
12,100298	9,067358	15,181962	8,695859	7,644654	DEVIAZIONE STANDARD
4,518325559	3,385807	5,669037788	3,247087	2,854561	INTERVALLO DI CONFIDENZA

NEO4J					
Query1	Query2	Query3	Query4	Query5	
11,801396	44,068646	49,3363124	42,7604378	52,59137533	MEDIA
1,627219	37,464022	17,587490	22,781518	29,991286	DEVIAZIONE STANDARD
0,6076134	13,98929564	6,56727686	8,506758622	11,19893019	INTERVALLO DI CONFIDENZA





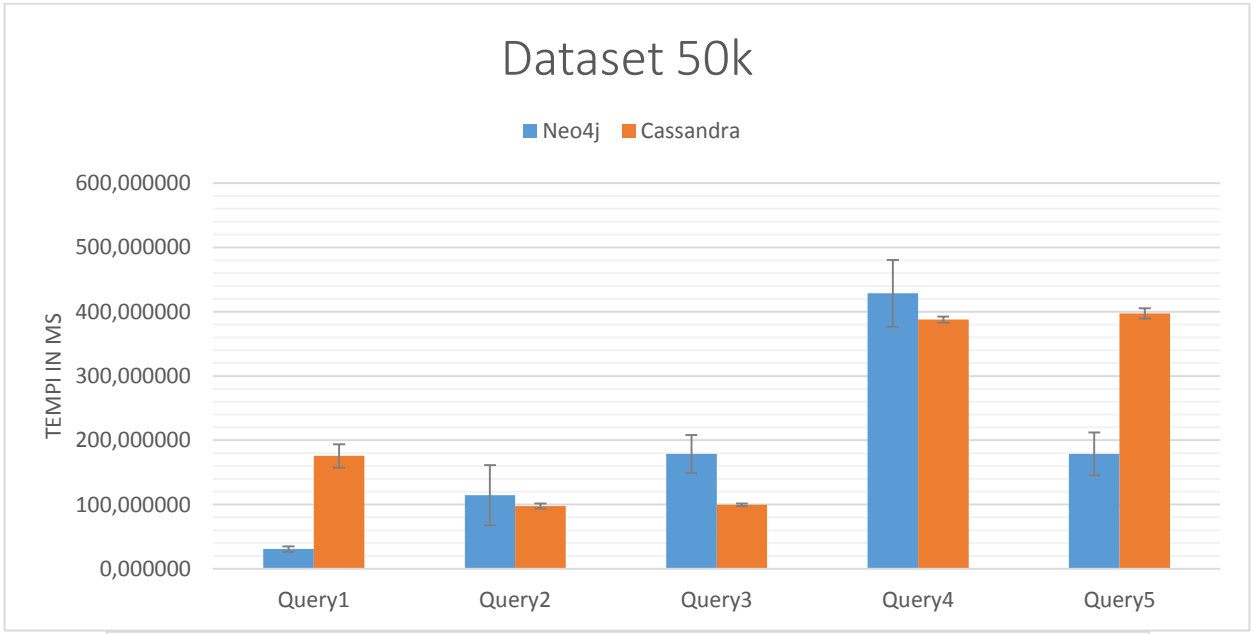
DATASET da 50k

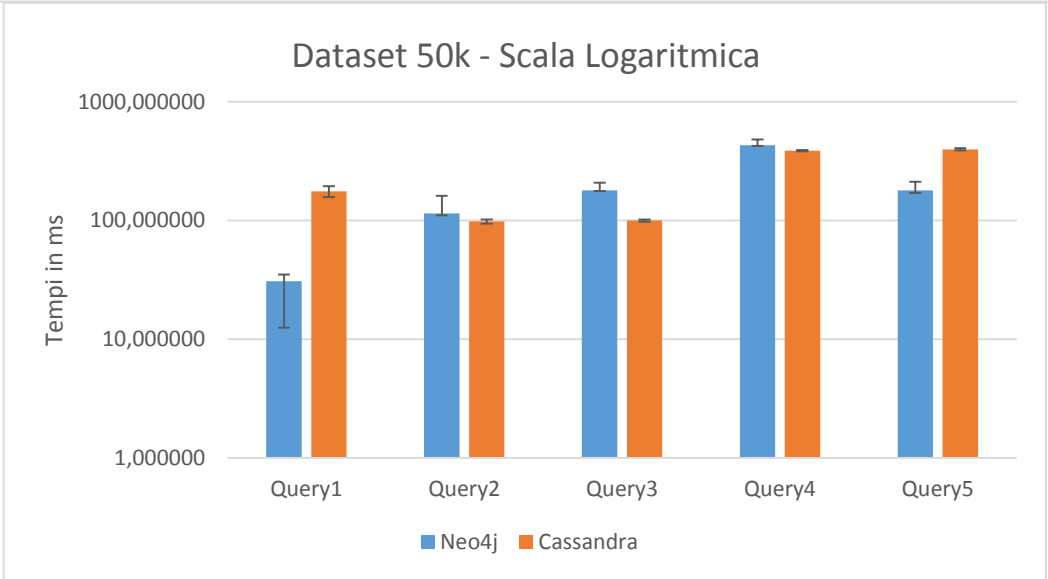
NEO4j

Query1	Query2	Query3	Query4	Query5	
30,799953	114,637361	178,653210	428,618386	178,880005	MEDIA
11,443998	125,371330	79,499139	139,443110	89,835491	DEVIAZIONE STANDARD
4,273259	46,814424	29,68546648	52,06891291	33,5451237	INTERVALLO DI CONFIDENZA

CASSANDRA

Query1	Query2	Query3	Query4	Query5	
175,670484	97,882395	99,682840	387,705601	397,376454	MEDIA
48,985622	10,704835	5,297593	12,509469	21,612915	DEVIAZIONE STANDARD
18,29153184	3,99725125	1,9781538	4,67111252	8,070394986	INTERVALLO DI CONFIDENZA





CONCLUSIONI

Come si evince dai grafici, è possibile affermare che **Neo4J si comporta senza dubbio meglio rispetto a Cassandra**.
Analizziamo adesso singolarmente i vari dataset.
Possiamo notare nel dataset da 100 record, la struttura column-oriented di cassandra risulta più efficiente rispetto a neo4j, che comunque garantisce sempre delle ottime performance.

Già dal secondo dataset da 10000 Neo4j riprende terreno e i 2 database ci forniscono prestazioni molto simili. Nell'ultimo dataset utilizzato, Neo4j si conferma più adatto a gestire query di una certa complessità, offrendoci sempre delle performance migliori rispetto a Cassandra.
Analizziamo le query:

- **Query1**: Nella prima query, specialmente in dataset più piccoli, possiamo notare come Cassandra risulta essere più efficiente rispetto a Neo4J, nonostante entrambi mantengano comunque delle ottime performance;
- **Query2**: Nella seconda query, Neo4j dimostra quel valore in più relativamente alle performance sottolineando il fatto che comunque entrambi i database offrono prestazioni simili;
- **Query3**: Relativamente alla terza query, in dataset piccoli neo4j riesce ancora a reggere il confronto con cassandra, ma viene evidenziato il fatto che con l'aumentare della dimensione del dataset le prestazioni di neo4j vanno man mano ad essere più efficienti a differenza di Cassandra nella quale degradano molto più velocemente;

Il comportamento da evidenziare è che in generale, **all'aumentare della dimensione del dataset le performance di Cassandra degradano molto più velocemente rispetto a Neo4j**

L'inserimento dello stesso dataset sui i due ambienti è risultato più rapido su Neo4j rispetto a Cassandra che mi ha costretto a limitare i molto i dataset da analizzare. Difatti Neo4j ha impiegato circa 2 minuti e 26 secondi per caricare correttamente i record dal file csv con 50k record, a differenza di Cassandra che per effettuare la medesima operazione ha impiegato 3 giorni e 18 ore.