

*Università degli Studi di Messina*

*Dipartimento di Scienze Matematiche, Informatiche, Fisiche e Scienze della Terra*

*Relazione di Basi di Dati 2*

A. A. 2021/2022

***“Casi d’uso di antiriciclaggio di denaro”***

*Studenti:*

*Buemi Emanuele – Giambò Filippo*

*Docente:*

*Prof. Antonio Celesti*

Sommario

[Problematica affrontata 3](#_Toc94888189)

[Soluzioni DBMS considerate 4](#_Toc94888190)

[**Neo4j** 4](#_Toc94888191)

[**Hbase** 5](#_Toc94888192)

[Progettazione 6](#_Toc94888193)

[**Data model Neo4j** 9](#_Toc94888194)

[**Data model Hbase** 10](#_Toc94888195)

[Implementazione 11](#_Toc94888196)

[**Implementazione in Neo4J** 12](#_Toc94888197)

[**Implementazione in Hbase** 15](#_Toc94888198)

[**Implementazione in query** 16](#_Toc94888199)

[**QUERY 1** 17](#_Toc94888200)

[**QUERY 2** 17](#_Toc94888201)

[**QUERY 3** 18](#_Toc94888202)

[**QUERY 4** 18](#_Toc94888203)

[**QUERY 5** 19](#_Toc94888204)

[Esperimenti 20](#_Toc94888205)

[**Query 1** 21](#_Toc94888206)

[**Query 2** 22](#_Toc94888207)

[**Query 3** 23](#_Toc94888208)

[**Query 4** 24](#_Toc94888209)

[**Query 5** 25](#_Toc94888210)

[Conclusioni 26](#_Toc94888211)

# 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.

I due DBMS presi in considerazione sono:

* **Neo4j**
* **Hbase**

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.

La problematica che abbiamo affrontato riguarda l’identificazione di tutte le persone coinvolte nell’**anti-riciclaggio di denaro** attraverso l’analisi delle transazioni da loro effettuate.

Il riciclaggio di denaro *“sporco”*, noto internazionalmente come money

laundering, secondo le definizioni giuridiche consiste nella sostituzione o trasferimento di denaro, beni o altre utilità provenienti da attività criminose con lo scopo di ostacolare la possibilità di identificare la loro provenienza illecita.

In particolare i casi di studio in cui è possibile intravedere dei casi sospetti di anti-riciclaggio di denaro riguardano:

* l’identificazione degli UBO, cioè un individuo che detiene più del 25% del capitale (quote) riguardante una determinata entità (società).
* il problema dello smurfing che consiste nell’identificare tutte le transazioni che vengono effettuate su uno stesso conto bancario da parte di più persone.
* il problema del round trip in cui i fondi vengono restituiti dopo essere stati trasferiti a delle entità collegate tra di loro.

Il contrasto al fenomeno permette di combattere efficacemente la criminalità organizzata in quanto si toglie alla stessa la possibilità di utilizzare effettivamente il denaro proveniente da attività illecite. Seguendo le tracce lasciate dalle transazioni per riciclare il denaro è possibile risalire ai reati e quindi ai colpevoli.

# Soluzioni DBMS considerate

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

* Neo4j
* Apache HBase

## **Neo4j**

Neo4j è un database NoSQL che appartiene alla categoria dei “***Graph database***”.

Il data model è un grafo orientato con proprietà nome-valore, che prevede:

* **nodi**, che rappresentano le entità,
* **archi** (relazioni), che esprimono le associazioni tra le entità (tra nodi).

Poiché Neo4j è progettato attorno a questa semplice e potente ottimizzazione (fatta da nodi e archi), esegue delle query con connessioni molto complesse più velocemente rispetto ad altri database (ma con un volume di dati più basso).

Grazie a questa caratteristica, Neo4j viene ottimamente utilizzato nel campo

dell’intelligenza artificiale (IA) o nel machine learning, perché ci consente di

analizzare i dati direttamente a livello database senza l’overhead di doverli

trasferire e gestire a livello applicativo.

Neo4j è stato sviluppato in Java e presenta le seguenti caratteristiche:

* È robusto e scalabile.
* Fornisce alte prestazioni.
* Fornisce un’alta velocità di interrogazione tramite gli attraversamenti.
* Dispone di numerosi driver che gli permettono di interfacciarsi con numerosi linguaggi di programmazione.
* Supporta un linguaggio di querying dichiarativo chiamato **Cypher,** creato dal team di Neo4j allo scopo di interagire con il database in modo semplice e veloce.
* Cypher è ottimizzato per lo studio dei grafi e ci consente di indicare cosa vogliamo selezionare, inserire, aggiornare o eliminare dai nostri dati senza il bisogno di fornire una descrizione su come farlo.

## **Hbase**

HBase è un database NoSQL, distribuito, di tipo “***Column-Oriented***”, open

source e scritto in linguaggio Java.

Memorizza i dati in tabelle che mantengono il formato chiave-valore.

Ogni riga contiene:

* una chiave univoca detta “***row key***”
* una o più “***column families***”, ovvero famiglie di colonne che contengono colonne figlie con tipi di dato simili tra loro.

Ad ogni colonna invece corrisponde un value (valore), a cui viene associato un timestamp che servirà per gestire il versioning dei dati (utile per le repliche).

HBase è una soluzione di database NoSQL che fa parte del framework di tool di Hadoop.

L’architettura di Hbase al livello più basso si basa su ***HDFS*** (*Hadoop Distributed File System*), ovvero il file system di Hadoop che permette di memorizzare i nostri dati su un sistema distribuito e parallelo mentre al livello più alto si basa su **HBaseMaster**, che comunica con altri server slave di nome **HRegionServer**. Inoltre utilizza un middleware di comunicazione chiamato **Zookeeper** per coordinare la comunicazione tra il client e il nodo master.

HBase risulta più efficiente quando si ha a che fare con dei progetti di applicazioni molto complesse, fornisce API che consentono lo sviluppo in qualsiasi linguaggio di programmazione e inoltre non necessità di occuparsi in prima persona della replicazione e manutenzione dei dati, in quanto è HDFS ad occuparsene.

# Progettazione

Per lo studio delle problematiche descritte nella sezione precedente sono stati generati dei datasets fittizi che simulano delle situazioni legate all’anti-riciclaggio di denaro.

Gli attributi che abbiamo assegnato alle varie entità sono:

* **id**

Identificativo univoco assegnato ad ogni riga del dataset.

* **nome**

Nome della persona che effettua e/o riceve la transazione.

* **cognome**

Cognome della persona che effettua e/o riceve la transazione.

* **nome\_cognomeP1**

Nome completo della persona che effettua la transazione.

* **nome\_cognomeP2**

Nome completo della persona che riceve la transazione.

* **id\_conto\_P1**

Identificativo della persona che effettua la transazione.

* **id\_conto\_P2**

Identificativo della persona che riceve la transazione.

* **tel**

Numero di telefono della persona.

* **email**

Indirizzo email della persona.

* **indirizzo**

Indirizzo civico della persona.

* **quote**

Specifica la percentuale di possesso della società da parte della persona.

* **professione**

Titolo di lavoro della persona.

* **id\_societa1**

Identificativo della società che effettua la transazione.

* **id\_societa2**

Identificativo della società della quale la persona possiede delle quote.

* **nome\_societa**

Nome della società riferito sia alla società che effettua la transazione e sia a quella in possesso della persona.

* **paese**

Prefisso della nazione dove si trova la società.

* **citta**

Città riferita della sede dove si trova la società.

* **id\_credito\_banca**

Identificativo del conto bancario appartenente alla persona ricevente la transazione.

* **tipo\_tessera**

Tipo di carta di credito.

* **data\_apertura**

Data di apertura del conto bancario.

* **data\_transazione**

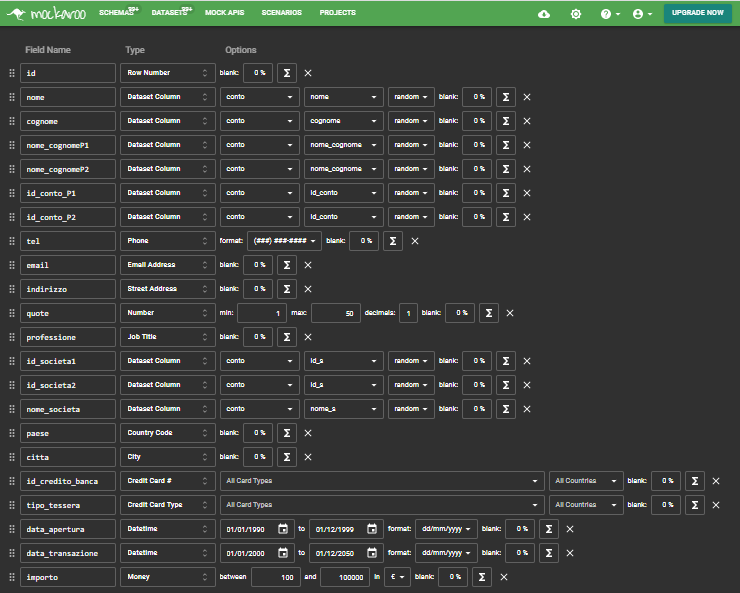
Data di quado è stata effettuata la transazione.

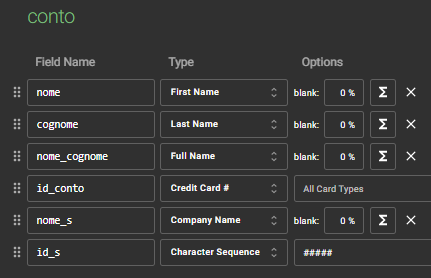
* **Importo**

Somma di denaro trasferita durante la transazione.

Per questo progetto, sono stati preparati dei sei di dati fittizi utilizzando **Mockaroo**. I dataset sono stati salvati in formato csv (comma-separated values).

Qui sotto è ripoto lo screen della struttura dati in Mockaroo:





Per fare in modo che una stessa persona possa effettuare e ricevere una transazione, e fare in modo che una persona o una società effettui più transazioni su più conti bancari (creando così delle relazioni 1 – N) è stato utilizzato un dataset contenente i soli attributi: nome, cognome, nome\_cognome, id\_conto, nome\_s e id\_s

e abbiamo importato questo dataset come ripetizione random all’interno del dataset precedente tramite le funzioni di Mockaroo.

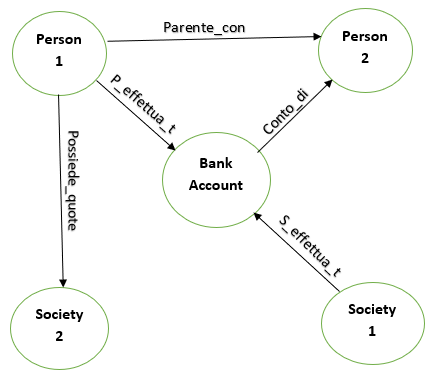
## **Data model Neo4j**

Il data model di Neo4j si focalizza su 5 entità che sono:

* 2 entità **Person**,
* 1 entità **BankAccount**,
* 2 entità **Society**,

e 5 relazioni che sono:

* 1 relazione “**P\_effettua\_t**” 🡪 che fa riferimento alla persona che effettua la transazione,
* 1 relazione “**Conto\_di**” 🡪 che fa riferimento al conto bancario posseduto da una determinata persona.
* 1 relazione “**S\_effettua\_t**” 🡪 che fa riferimento alla società che effettua la transazione,
* 1 relazione “**Possiede\_quote**” 🡪 che fa riferimento alla persona che possiede la società,
* 1 relazione “**Parente\_con**” 🡪 che fa riferimento alla relazione di parentela tra due persone.



Secondo questo modello a grafo da noi costruito una persona può effettuare una o più transazioni (Person 1) sul conto bancario (Bank Account) di una seconda persona e nello stesso tempo ricevere una o più transazioni (Person 2) su un proprio conto bancario.

Allo stesso modo una società può effettuare una o piu transazioni (Society 1) e può essere posseduta da diverse persone (Society 2).

## **Data model Hbase**

Il data model di Hbase è costituito da una tabella costituita da **3 Column Family** che sono:

* **Person** 🡪 composta da 7 colonne (id\_conto\_P1, nome, cognome, nome\_cognomeP1, tel, email, indirizzo);
* **Society** 🡪 composta da 5 colonne (id\_societa2, nome\_societa, paese, citta, quote);
* **BankAccount** 🡪 composta da 8 colonne (id\_conto\_P2, id\_credito\_banca, nome\_cognomeP2, professione, tipo\_tessera, data\_transazione, data\_apertura, importo).

**DB**

**FAMILY: Person: Columns:** id\_conto\_P1, nome, cognome,

nome\_cognomeP1, tel, email, indirizzo

**Society: Columns:** id\_societa2, nome\_societa, paese,

citta, quote

**BankAccount Columns:** id\_conto\_P2, id\_credito\_banca,

nome\_cognomeP2, professione, tipo\_tessera,

data\_transazione, data\_apertura, importo

# Implementazione

In questa sezione verranno analizzati i metodi più importanti usati nel programma. Per l’implementazione di questo progetto abbiamo utilizzato un sistema operativo Ubuntu, in quanto l’installazione di Hbase su ambiente Windows risulta più problematica.

Il linguaggio di programmazione usato è Java e per il collegamento con i due database sono stati aggiunti al file “***pom.xml***” i seguenti **driver**:

* Per Hbase:

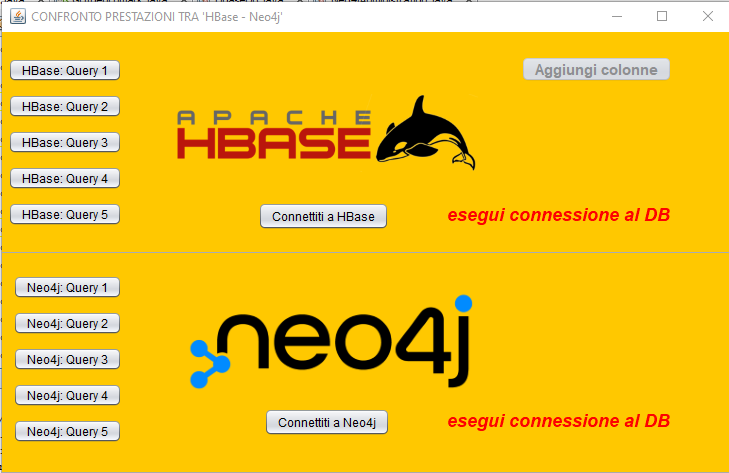
[Maven Repository: org.apache.hbase » hbase-client » 2.4.8 (mvnrepository.com)](https://mvnrepository.com/artifact/org.apache.hbase/hbase-client/2.4.8)

* Per Neo4j:



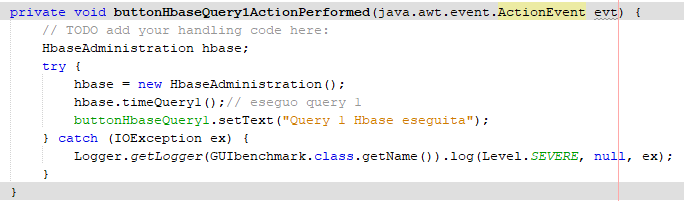
[Maven Repository: org.neo4j.driver » neo4j- java-driver » 4.4.2 (mvnrepository.com)](https://mvnrepository.com/artifact/org.neo4j.driver/neo4j-java-driver/4.4.2)

Per rendere il progetto più dinamico abbiamo creato una **GUI** che ci permette di connetterci ai due database e di eseguire le 5 query cliccando sui bottoni.



A ogni tasto è associato un evento gestito mediante un gestore di eventi **ActionListener** che implementa un metodo chiamato actionPerformed; questo metodo in base all'evento che si verifica, risponde a quel determinato evento associando il rispettivo comportamento cioè l’esecuzione delle query.

Di seguito è riportato un esempio del metodo **ActionPerformed** associato al bottone che esegue la query 1 di Hbase.

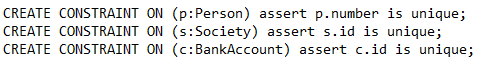


## **Implementazione in Neo4J**

Per quanto riguarda Neo4j, inizialmente abbiamo importato il dataset creato in formato CSV tramite apposita operazione di import.

Successivamente abbiamo creato il grafo utilizzando i seguenti **comandi Cypher**:

* **CREATE CONSTRAINT** per creare dei vincoli di univocità per ciascuna entità sulle chiavi primarie.

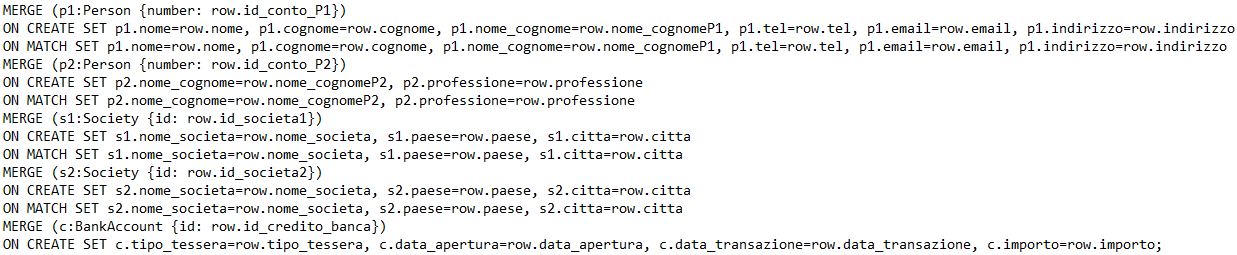


* **LOAD CSV** per importare il dataset in formato csv all’interno del database



* **MERGE** che rappresenta una combinazione tra il comando CREATE e MATCH, in particolare se il nodo esiste restituisce i risultati, altrimenti se il nodo non esiste lo crea e restituisce i risultati.

**ON CREATE SET** che ci permette di stabilire i campi da estrarre dal file csv e da passare ai vari nodi.

Nel nostro caso il comando MERGE è stato utilizzato per creare i nodi Person, Society, BankAccount mentre il comando ON CREATE SET ci ha permesso di associare gli attributi corretti ai nodi.

* **DROP CONSTRAINT** per eliminare i precedenti vincoli.



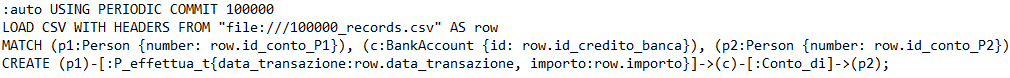
* **CREATE INDEX** per aggiungere degli indici a number di Person e a id di Society e Bank Account.



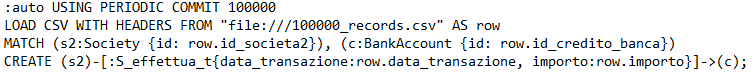
* **MATCH** utilizzato nella creazione delle relazioni e ci permette di stabilire dei legami tra le chiavi primarie delle entità coinvolte nella relazione.

Le relazioni che abbiamo creato sono le seguenti:

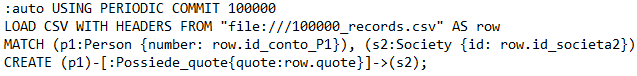
* La relazione P\_effettua\_t 🡪 tra la persona che effettua la transazione e il conto corrente della persona ricevente.

La relazione Conto\_di 🡪 tra il conto corrente e la persona posseditore del conto.

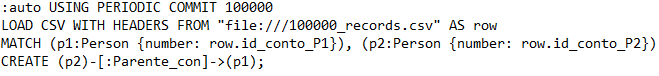
* La relazione S\_effettua\_t 🡪 tra la società che effettua la transazione e il conto corrente della persona ricevente.

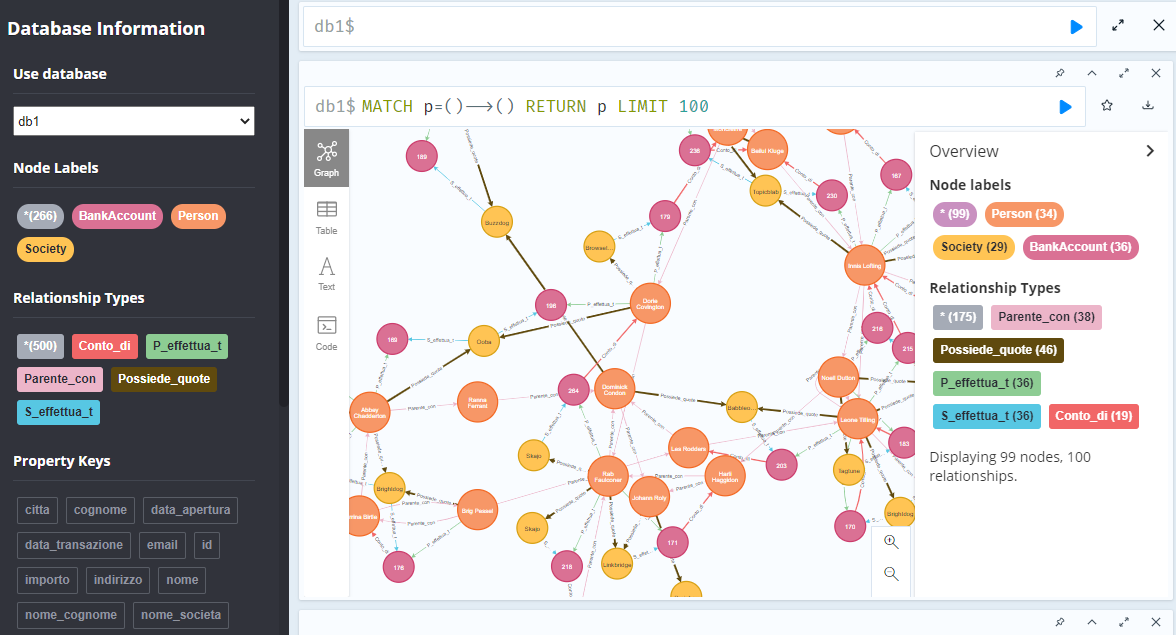


* La relazione Possiede\_quote 🡪 tra la persona e la società posseduta da quella persona.



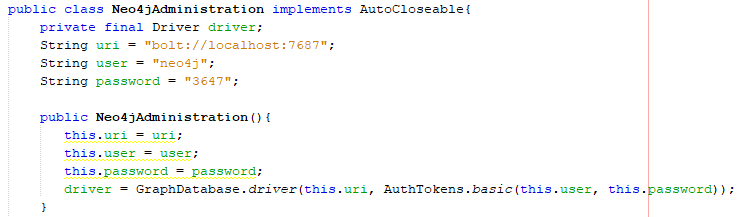
* La relazione Parente\_con 🡪 tra due entità Person che sono parenti.



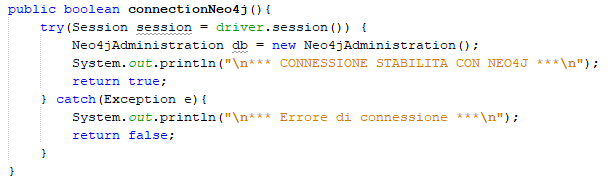


**Esempio di visualizzazione del grafo sull’interfaccia Neo4j utilizzando il linguaggio di querying Cypher.**

Dopo aver creato il grafo attraverso il linguaggio di programmazione Java abbiamo definendo all’interno del costruttore della classe Neo4jAdministration() l’uri (corrispondente alla bolt port), lo user, la password e un driver per un’istanza di neo4j.



Successivamente con il metodo connectionNeo4j() abbiamo eseguito la connessione a Neo4j.



## **Implementazione in Hbase**

Per quanto riguardata HBase è stata scelta una **configurazione standalone** per la realizzazione di questo progetto.

Sebbene HBase disponga di una propria Shell per l'invio di interrogazioni, si

è scelto di creare il tutto con Java attraverso i driver messi a disposizione

per l'interfacciamento con HBase, in particolare sono state utilizzate delle

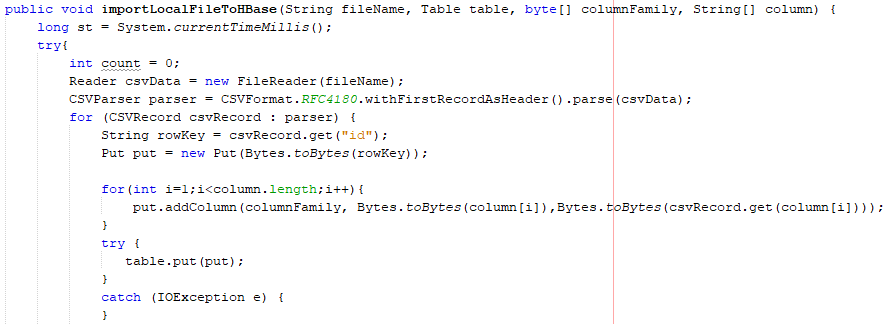
API Java.

Per prima cosa abbiamo caricato i nostri dati sperimentali utilizzando la libreria Java

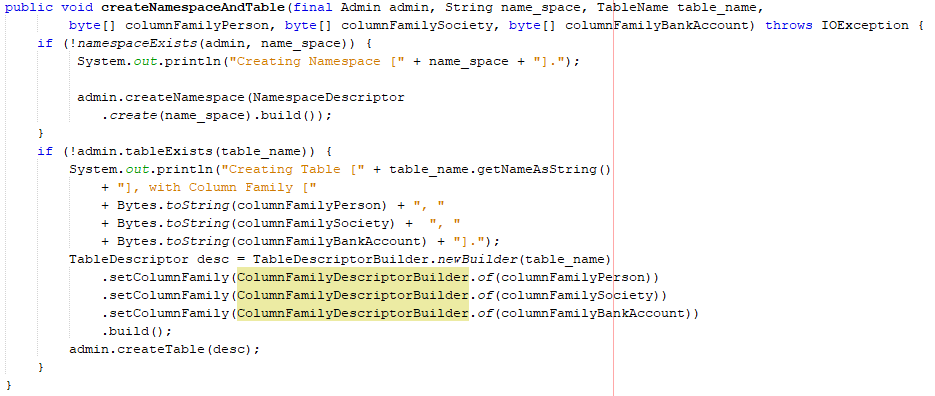
“commons.csv” per la lettura dei file in formato csv, e in seguito, tramite API Java

“hbase.client” con il comando “put”, sono stati caricati i dati nelle corrispondenti

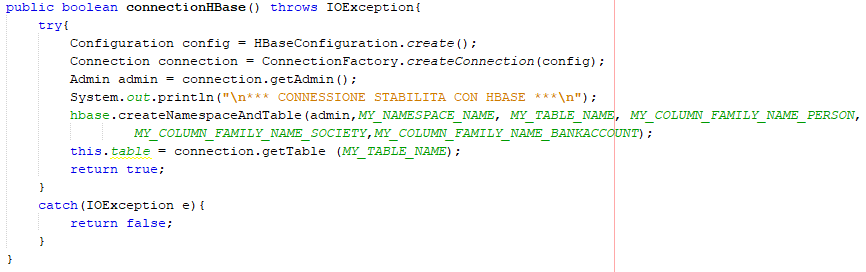
Column Family.



Per la creazione del Namespace e della tabella contenente le 3 Column family (Person, Society, BankAccount) in HBase, sono stati utilizzati i seguenti comandi:



Per creare la connessione con HBasemaster tramite API Java abbiamo definito il seguente metodo.



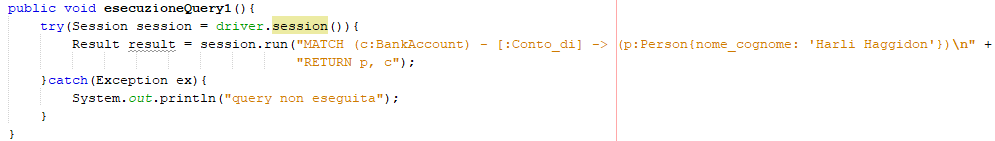
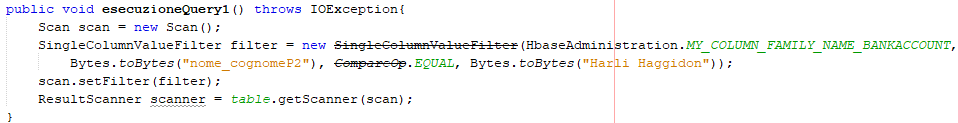
## **Implementazione in query**

Una volta inseriti i dati vengono eseguite per entrambi i DB cinque query. Per ciascuna query, inoltre, abbiamo implementato un metodo che calcola i tempi di esecuzione (della rispettiva query) 31 volte e che stampa il risultato della query.

Qui possiamo vedere un esempio di tale metodo per calcolare i tempi sulla query 1 di Neo4j.

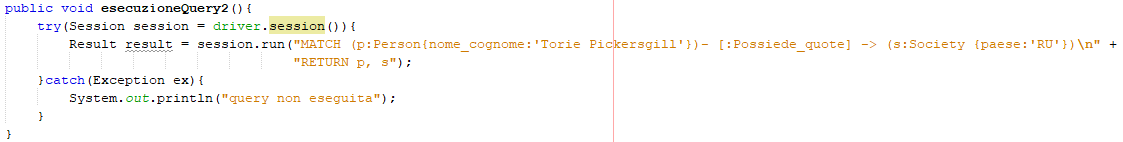
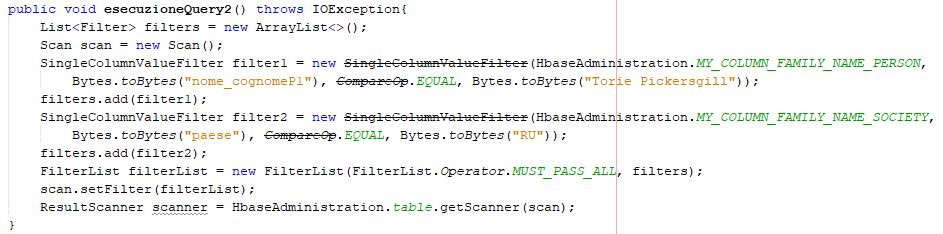
### **QUERY 1**

Trovare tutti i conti bancari posseduti da "Harli Haggidon".

* NEO4J
* HBASE

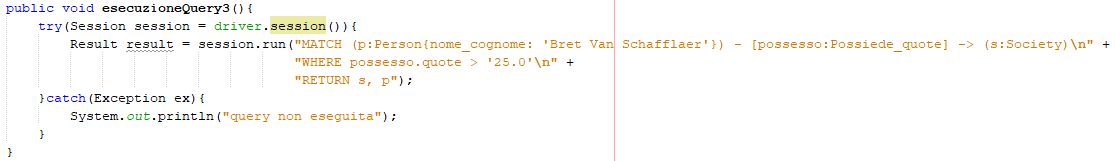
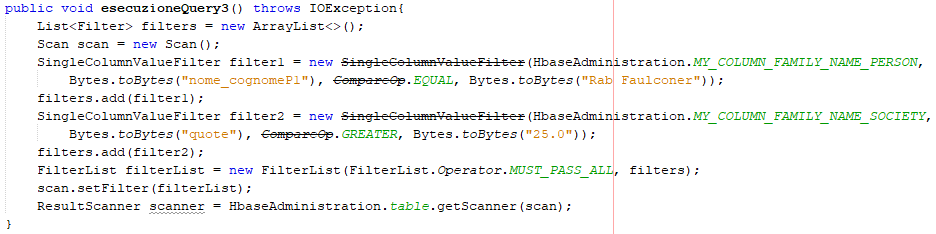
### **QUERY 2**

Trovare tutte le società possedute da "Torie Pickersgill" con sede in Russia (RU).

* NEO4J
* HBASE

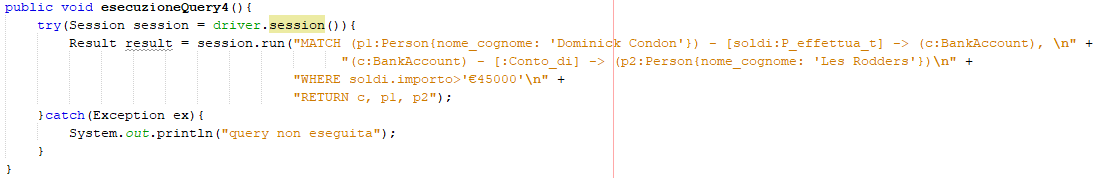
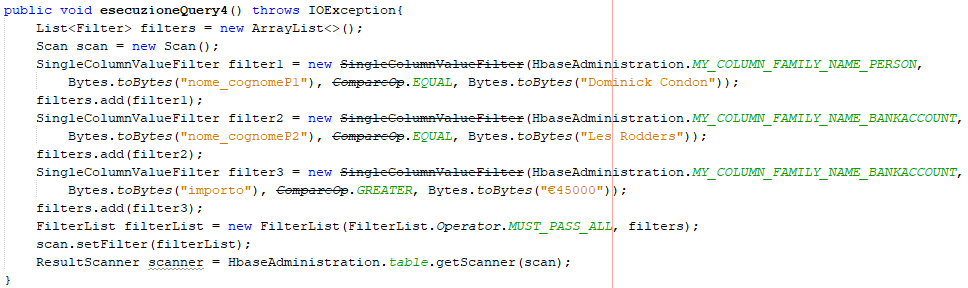
### **QUERY 3**

Trovare tutte le società in cui "Rab Faulconer" possiede delle quote sociali superiori al 25%.

* NEO4J
* HBASE

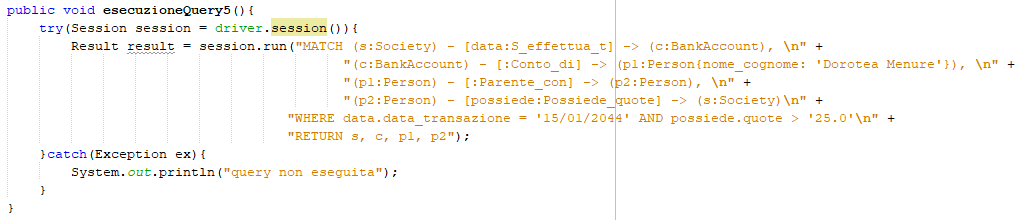
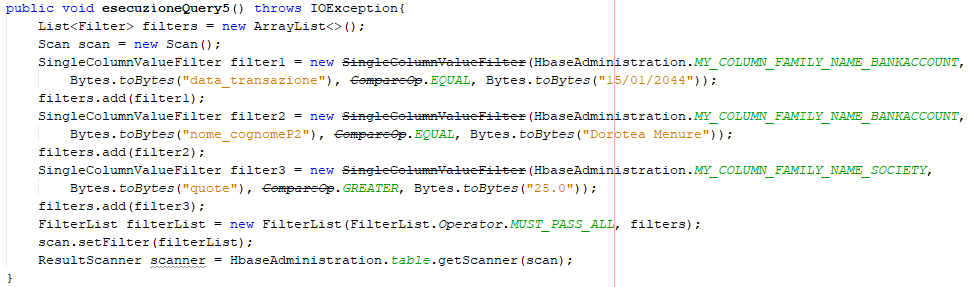
### **QUERY 4**

Trovare la persona di nome "Dominick Condon" che ha effettuato delle transazioni con un importo > di 45000 euro sul conto corrente di "Les Rodders".

* NEO4J
* HBASE

### **QUERY 5**

Trovare la società che ha effettuato delle transazioni il 15 Gennaio del 2044 sul conto corrente posseduto da "Dorotea Menure" che è parente con delle persone che possiedono quote sociali maggiori al 25% riguardanti la società stessa.

* NEO4J
* ****HBASE

# Esperimenti

Gli esperimenti effettuati mirano a confrontare le prestazioni riguardanti i tempi di esecuzione delle query (con gradi di complessità crescenti) al crescere del dataset per **Neo4j** e **Hbase**.

Per effettuare gli esperimenti, sono stati valutati dataset crescenti considerando una dimensione variabile del 25%, 50%, 75% e 100%.

In particolare sono stati considerati 4 dataset con:

* **100 record**,
* **1000 record**,
* **10000 record**,
* **100000 record**.

Per ogni query è stata inserita una tabella contenente:

* il 1° tempo di esecuzione che viene preso a parte in quanto la prima esecuzione di una query impiega tendenzialmente più tempo rispetto a quelle successive,
* la media dei tempi delle 30 esecuzioni successive,
* la deviazione standard,
* l’intervallo di confidenza al 95%.

Inoltre, per ogni query sono anche presenti due istogrammi:

* il primo che mostra l’andamento del **1° tempo di esecuzione** al crescere del dataset,
* il secondo che mostra l’andamento della **media dei tempi**, con il relativo intervallo di confidenza al crescere del dataset.

## **Query 1**

## **Query 2**

## **Query 3**

## **Query 4**

## **Query 5**

# Conclusioni

Dagli esperimenti effettuati possiamo dire di avere un vincitore in quanto risulta che in tutti i casi, **HBase risponde più velocemente rispetto a Neo4j per tutte le query esaminate**.

* HBase è un database di tipo Column Oriented e riesce a gestire meglio ampi volumi di dati con complessità relativamente bassa.
* Neo4J, invece, è un database di tipo Graph e riesce a gestire meglio dati con complessità elevata ma con un volume di dati minore.

In conclusione da questa indagine possiamo concludere che nonostante la complessità delle query aumenta il DBMS Hbase è senza dubbio migliore per modellare delle situazioni come quelle prese in carico secondo questo studio sull’antiriciclaggio di denaro.