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 UBO
BaseX vs Neo4j

Docente:
Prof. Massimo Villari
Prof. Antonio Celesti

Realizzato da:
Davide Ferrara

Introduzione

Obiettivo del Progetto

L'obiettivo principale di questo progetto è esplorare le capacità di gestione e interrogazione di due diversi sistemi di gestione dei database: Neo4j e BaseX, nel contesto dell'identificazione e verifica del Beneficiario Effettivo Ultimo (UBO). Sebbene l'obiettivo non sia la determinazione diretta degli UBO come descritto nel sito ufficiale, questo studio si concentra sull'implementazione di un dataset che simula scenari tipici di strutture di proprietà aziendale complessa. Utilizzando tale dataset, il progetto mira a eseguire cinque query di complessità crescente su entrambi i database, valutando le loro performance in termini di velocità di esecuzione e capacità di gestione dei dati complessi.

Tecnologie Utilizzate

Per affrontare il problema della gestione e interrogazione dei dati relativi agli UBO, sono stati scelti due database rappresentativi di approcci diversi alla gestione dei dati:

- **Neo4j**: Neo4j è un database a grafo che rappresenta i dati sotto forma di nodi, relazioni e proprietà. Questo tipo di modello è particolarmente adatto per visualizzare e analizzare reti complesse di entità interconnesse, come le strutture di proprietà multilivello tipiche dell'identificazione degli UBO.
- **BaseX**: BaseX è un database XML nativo e un motore di interrogazione XQuery. Memorizza e gestisce dati in formato XML, un linguaggio altamente strutturato e adatto alla rappresentazione di informazioni gerarchiche complesse. Nel contesto del progetto sull'identificazione degli UBO, BaseX può essere utilizzato per gestire strutture di proprietà multilivello tramite la modellazione XML, permettendo interrogazioni efficaci grazie all'uso di XQuery per esplorare le relazioni tra le entità.

Per l'interazione con entrambi i database e l'automazione delle operazioni di inserimento dati e query, è stato scelto il linguaggio Python. Python offre una vasta gamma di librerie e strumenti, come py2neo per Neo4j e BaseXClient.py per BaseX, che facilitano la connessione ai database, l'esecuzione di operazioni di lettura e scrittura e l'analisi dei risultati.

Importanza dello Studio

L'identificazione e verifica del Beneficiario Effettivo Ultimo (UBO) è diventata una componente fondamentale per la conformità normativa nelle istituzioni finanziarie e in altri settori regolamentati a livello globale. La trasparenza nella proprietà aziendale è cruciale per combattere efficacemente il riciclaggio di denaro, la corruzione e altre forme di crimine finanziario.
Sebbene la determinazione degli UBO sia spesso complessa, comprendere chi controlla effettivamente un'entità giuridica consente alle organizzazioni di mitigare i rischi associati alle transazioni e alle partnership commerciali.

Negli ultimi anni, legislazioni internazionali come il Corporate Transparency Act negli Stati Uniti e le direttive europee contro il riciclaggio di denaro (4AMLD e 5AMLD) hanno intensificato l'attenzione sulla necessità di disporre di informazioni aggiornate e facilmente accessibili sugli UBO. Tuttavia, l'implementazione pratica di queste normative richiede strumenti che possano gestire in modo efficace le complessità legate alle strutture di proprietà multilivello.

In questo contesto, il presente studio si propone di confrontare Neo4j e BaseX per valutare quale dei due database possa offrire una maggiore efficienza nella gestione di dati complessi simili a quelli necessari per l'identificazione degli UBO. Sebbene Neo4j sia naturalmente adatto a questo compito grazie al suo modello a grafo, BaseX rappresenta un'opzione flessibile e scalabile che potrebbe essere altrettanto efficace grazie alla sua capacità di gestire documenti XML e interrogazioni XQuery. Attraverso l'esecuzione di cinque query di complessità crescente, questo progetto intende fornire una valutazione oggettiva delle capacità dei due database, considerando aspetti come la velocità di esecuzione, la scalabilità e la facilità di utilizzo.

Descrizione del Processo

Il progetto inizia con la creazione di un dataset simulato che rappresenta strutture di proprietà aziendale complesse, tipiche dello scenario UBO. Questo dataset include nodi che rappresentano entità aziendali e relazioni che descrivono i collegamenti di proprietà e controllo tra di esse. Successivamente, il dataset viene inserito in entrambi i database, Neo4j e BaseX, utilizzando script automatizzati scritti in Python.

Vengono quindi sviluppate cinque query, ciascuna con un livello di complessità crescente. Queste query sono progettate per testare vari aspetti delle capacità dei database, inclusa la capacità di eseguire operazioni su più livelli di relazioni e di gestire grandi volumi di dati. L'esecuzione di queste query su entrambi i database fornisce dati di performance che vengono poi analizzati per trarre conclusioni sulle capacità comparative di Neo4j e BaseX in questo contesto.

Il progetto si conclude con un'analisi dettagliata dei risultati, evidenziando i punti di forza e le debolezze di ciascun database e fornendo raccomandazioni su quale potrebbe essere la scelta migliore per applicazioni simili.

Descrizione dei Database

Introduzione ai Database Utilizzati

Nel contesto dell'attuale progetto, abbiamo scelto di analizzare e confrontare due distinti sistemi di gestione dei database: **Neo4j**, un database a grafo, e **BaseX**, un database **XML** nativo. Queste tecnologie rappresentano due paradigmi differenti nella gestione dei dati e offrono vantaggi unici a seconda del tipo di dati e delle esigenze applicative.

Neo4j

Neo4j è uno dei più diffusi database a grafo disponibili sul mercato, progettato specificamente per gestire e interrogare dati con relazioni complesse. Questa caratteristica lo rende particolarmente adatto per casi d'uso che coinvolgono connessioni intricate tra entità, come quelle tipiche delle strutture di proprietà aziendale necessarie per l'identificazione degli UBO.

Architettura e Modello dei Dati

Neo4j adotta un modello di dati basato sui grafi, dove le informazioni sono rappresentate tramite nodi, relazioni e proprietà:

- **Nodi**: Rappresentano entità o oggetti nel dominio dell'applicazione, come persone, aziende o transazioni.
- **Relazioni**: Definiscono le connessioni tra i nodi. Ogni relazione ha una direzione e un tipo che definisce il significato della connessione, come "possiede" o "gestisce".
- **Proprietà**: Sia i nodi che le relazioni possono avere attributi chiave-valore che forniscono informazioni aggiuntive, come nomi, date o valori monetari.

Il modello a grafo di Neo4j permette di eseguire query che riflettono direttamente la struttura relazionale dei dati, facilitando l'esecuzione di operazioni complesse che coinvolgono percorsi di connessione tra le entità.

Cypher: Il Linguaggio di Query

Neo4j utilizza Cypher come linguaggio di query, un linguaggio dichiarativo che consente di esprimere in modo conciso e leggibile le operazioni sui grafi. Cypher si ispira a SQL ma è ottimizzato per l'interazione con i grafi. Le sue caratteristiche includono:

- **Pattern Matching**: Permette di descrivere i percorsi nei grafi usando una sintassi visiva che rende facile capire le query a colpo d'occhio.
- **Aggregazione e Funzioni di Analisi**: Supporta operazioni di aggregazione sui nodi e le relazioni, oltre a funzioni di analisi avanzate.
- **Facilità d'Uso**: La sintassi di Cypher è intuitiva e consente agli utenti di esprimere query complesse con meno codice rispetto ad altri linguaggi di programmazione.

BaseX

BaseX è un database **XML nativo** che gestisce dati strutturati utilizzando il formato XML e sfrutta **XQuery** per l'interrogazione. È particolarmente adatto per la gestione di dati gerarchici e strutturati, consentendo di rappresentare facilmente relazioni complesse tipiche di scenari come l'identificazione degli UBO.

Architettura e Modello dei Dati

BaseX utilizza **XML** (Extensible Markup Language) come formato principale per la memorizzazione dei dati. XML è un formato di dati strutturato, particolarmente utile quando i dati sono gerarchici o hanno molteplici livelli di relazioni. In BaseX, i dati vengono organizzati in:

- **Documenti XML**: Ogni documento rappresenta una struttura di dati gerarchica, simile a un albero, con elementi, attributi e valori di testo.
- **Collezioni di Documenti**: I documenti XML possono essere raggruppati in collezioni, il che permette di organizzare e interrogare insiemi di dati correlati in modo efficiente.

Il modello XML di BaseX offre una struttura flessibile e ricca per rappresentare dati multilivello, ideale per modellare entità come aziende, persone e relazioni di proprietà.

XQuery: Il Linguaggio di Query

BaseX utilizza **XQuery**, uno standard per interrogare e manipolare documenti XML. XQuery consente di eseguire operazioni sofisticate su dati strutturati e gerarchici. Le caratteristiche principali di XQuery includono:

- **Query su Dati Gerarchici**: Permette di navigare e interrogare i dati XML come un albero, accedendo agli elementi e ai loro sottoelementi con una sintassi specifica.
- **Filtraggio e Trasformazione dei Dati**: Supporta operazioni di filtraggio e trasformazione dei dati XML, consentendo l'estrazione e la modifica di parti specifiche dei documenti.
- **Aggregazione e Funzioni Avanzate**: XQuery offre una vasta gamma di funzioni per l'aggregazione e l'elaborazione dei dati, simile a quanto offerto nei tradizionali linguaggi di query.

Struttura del Database

La struttura principale dei nostri dati per entrambi i database si base in questo modo:

Schema dei Dati

Collezioni e Nodi

1. **Aziende (Companies)**:
    - **Descrizione**: Rappresentano le aziende e contengono informazioni dettagliate su ciascuna entità aziendale.
    - **Campi**: Name, address, legal_form, registration_details e financial_data.

    - **Riferimenti**: IDs of administrators, shareholders, ubo, transactions.
2. **Amministratori (Administrators)**:
    - **Descrizione**: Rappresentano gli individui che amministrano le aziende.
    - **Campi**: Name, address, birthdate, nationality.
3. **Azionisti (Shareholders)**:
    - **Descrizione**: Rappresentano le persone o le entità che possiedono azioni nelle aziende.
    - **Campi**: Name, type, ownership_percentage, address, birthdate e nationality
4. **Beneficiari Effettivi (UBO - Ultimate Beneficial Owners)**:
    - **Descrizione**: Rappresentano i proprietari ultimi delle aziende, con un focus specifico su chi beneficia effettivamente delle proprietà.
    - **Campi**: Name, address, birthdate, nationality, ownership_percentage e type.
5. **Transazioni (Transactions)**:
    - **Descrizione**: Rappresentano le transazioni finanziarie effettuate dalle aziende.
    - **Campi**: Type, amount, date, currency.
6. **Controlli KYC/AML (KYC_AML_Checks)**:
    - **Descrizione**: Rappresentano i controlli di conformità per la verifica dei beneficiari effettivi.
    - **Campi**: Type, result, date e notes.
    - **Riferimenti**: ubo_id.

BaseX

Ecco come sono state strutturate le relazioni in BaseX:

1. Relazioni Implicite:

Abbiamo considerato l'entità **companies** come entità principale nel contesto del nostro database BaseX. Possiamo descrivere le relazioni tra l'entità **companies** e le altre entità del database attraverso riferimenti ID. In questo modello, le relazioni sono gestite attraverso elementi XML all'interno dei documenti di **companies** che contengono ID di documenti di altre collezioni.

Tutte le relazioni nel nostro database sono di tipo uno-a-molti. Questo modello di relazioni è stato scelto per la flessibilità e la scalabilità nella gestione dei dati. In questo contesto, un'azienda può essere associata a molte altre entità subordinate (come amministratori, azionisti, transazioni, ecc.). Questo tipo di relazione permette di rappresentare in modo efficiente e naturale le connessioni tra le diverse entità e di adattarsi facilmente a nuove esigenze e modifiche.

Relazione con Administrators:

- **Descrizione**: Ogni azienda ha uno o più amministratori che la gestiscono.
- **Dettagli della Relazione**: Un documento **company** può includere un array di ID che fanno riferimento a documenti **administrators**. Questo tipo di relazione consente di gestire facilmente l'elenco di amministratori di ogni azienda e di mantenere aggiornate le informazioni relative.

Relazione con Shareholders:

- **Descrizione**: Le aziende hanno azionisti che possiedono quote della società.
- **Dettagli della Relazione**: L'azienda può avere un array di ID che puntano a documenti **shareholders**. Questo approccio consente di rappresentare con precisione le proprietà azionarie delle aziende.

Relazione con UBO (Ultimate Beneficial Owners):

- **Descrizione**: I beneficiari effettivi sono coloro che, direttamente o indirettamente, traggono vantaggio dalla proprietà dell'azienda.
- **Dettagli della Relazione**: L'azienda tiene traccia degli ID dei beneficiari effettivi nei suoi documenti. La struttura consente di mappare chiaramente chi beneficia dalla proprietà di un'azienda.

Relazione con Transactions:

- **Descrizione**: Le aziende effettuano transazioni finanziarie che devono essere registrate.
- **Dettagli della Relazione**: L'azienda può avere numerose transazioni finanziarie. Questo permette di tracciare tutte le attività finanziarie di un'azienda in modo dettagliato.

Relazione con KYC AML Checks:

- **Descrizione**: Le aziende devono effettuare controlli di conformità KYC/AML per prevenire il riciclaggio di denaro e il finanziamento del terrorismo.
- **Dettagli della Relazione**: L'azienda può avere molti controlli KYC/AML associati. Questo modello di relazione supporta il monitoraggio e la gestione efficiente delle attività di conformità.

Neo4j

Ecco come sono abbiamo strutturato le relazioni in Neo4j:

2. Relazioni:

- **AZIENDA_HA_AMMINISTRATORE (COMPANY_HAS_ADMINISTRATOR):** Relazione tra un'azienda e i suoi amministratori, includendo il ruolo e le date di inizio e fine.
- **AZIENDA_HA_AZIONISTA (COMPANY_HAS_SHAREHOLDER):** Relazione tra un'azienda e i suoi azionisti, con proprietà come la percentuale di partecipazione e la data di acquisto.
- **AZIENDA_HA_UBO (COMPANY_HAS_UBO):** Relazione tra un'azienda e i suoi beneficiari effettivi, simile a quella con gli azionisti.
- **AZIENDA_HA_TRANSAZIONE (COMPANY_HAS_TRANSACTION):** Relazione tra un'azienda e le sue transazioni.
- **UBO_HA_CONTROLLI (UBO_HAS_CHECKS):** Relazione tra un beneficiario effettivo e i controlli KYC/AML associati.

Processi di Popolamento dei Database

I dati per entrambi i database sono stati generati utilizzando la libreria Python `Faker`, che ha permesso di creare dati realistici ma fittizi, come nomi, indirizzi e date.

Generazione dei Dati

- **Administrators:** Sono stati generati 5000 record, inclusivi di dettagli come nome, indirizzo, data di nascita e nazionalità.
- **Shareholders:** Sono stati creati 5000 record con informazioni simili, ma includendo anche il tipo di azionista e la percentuale di proprietà.
- **Ubo (Ultimate Beneficial Owners):** 10.000 record sono stati generati, comprendendo dettagli simili agli azionisti.
- **Transactions:** 400.000 record di transazioni sono stati creati con dettagli come tipo, importo, data e valuta.
- **KYC_AML_Checks:** 30.000 record sono stati generati per i controlli di verifica e conformità.
- **Companies:** 50.000 record sono stati creati con dettagli come nome, indirizzo, forma legale, dati finanziari e riferimenti ad amministratori, azionisti, UBO, transazioni e controlli KYC/AML.

I dati generati sono stati esportati in file CSV, che sono poi stati letti e importati nei rispettivi database utilizzando script Python.

Esempio script generazione dati **`ADMINISTRATORS:`**

```
import random
import csv
import json
from faker import Faker

# Crea un'istanza del generatore di dati falsi
fake = Faker()

# Definisci il numero di record da generare per ciascun tipo di entità
NUM_ADMINISTRATORS = 5000

# Genera i dati per gli amministratori
administrators = []
for administrator_id in range(1, NUM_ADMINISTRATORS + 1):
    name = fake.name()
    address = fake.address()
    birthdate = fake.date_of_birth(minimum_age=25,
maximum_age=70).strftime('%Y-%m-%d')
    nationality = fake.country()
    administrators.append({
        'id': administrator_id,
        'name': name,
        'address': address,
```

```
        'birthdate': birthdate,
        'nationality': nationality
    })

# Scrivi i dati degli amministratori in un file CSV
with open('Dataset/File/administrators.csv', 'w', newline='') as csvfile:
    fieldnames = ['id', 'name', 'address', 'birthdate', 'nationality']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(administrators)

print("CSV file 'administrators.csv' successfully created.")
```

Formati di Dati
I dati (come descritto in precedenza con la dimostrazione dello script) sono stati salvati in file CSV, con ciascun file contenente informazioni per una specifica entità. I file CSV generati sono:

- *`administrators.csv`*
- *`shareholders.csv`*
- *`ubo.csv`*
- *`transactions.csv`*
- *`kyc_aml_checks.csv`*
- *`companies.csv`*

Motivazione della Scelta di Python

Python è stato selezionato per questo progetto grazie alle sue numerose caratteristiche che lo rendono ideale per lo sviluppo di applicazioni nel contesto dei database e dell'analisi dati:

Sintassi Intuitiva e Leggibilità: Python offre una sintassi chiara e leggibile che semplifica lo sviluppo del codice e facilita la collaborazione tra membri del team con diversi livelli di competenza tecnica.

Ecosistema di Librerie:
Python dispone di una vasta gamma di librerie, tra le quali abbiamo utilizzato nel progetto alcune delle più importanti. Tra queste troviamo:

- **pandas** per la manipolazione avanzata dei dati.
- **py2neo** per lavorare con Neo4j e gestire dati grafici.
- **faker** per la generazione di dati fittizi.
- **matplotlib** per la creazione di grafici.

Queste librerie accelerano il processo di sviluppo e riducono la necessità di codice personalizzato.

Relazione sulla Progettazione e Inserimento dei Dati

Nel contesto del nostro progetto, abbiamo creato e configurato il database **Neo4j** con un unico database denominato **UBO**. All'interno di questo database, sono stati configurati quattro dataset distinti per rappresentare le diverse percentuali di campioni: **100%**, **75%**, **50%** e **25%**. Questa suddivisione consente di gestire e analizzare i dati in base a diverse dimensioni campionarie all'interno di un singolo ambiente Neo4j.

Parallelamente, per **BaseX**, i dati sono stati organizzati in documenti XML, mantenendo le stesse percentuali di campioni. Per ciascuna percentuale, i dati sono stati generati casualmente utilizzando un generatore di dati, assicurando che il contenuto fosse coerente e pertinente all'analisi degli UBO.

Documenti : Administrators, Companies, KYC_AML_Checks, Shareholders, Transactions, UBO

- **Database per Percentuale (che abbiamo nominato UBO_%):**
  - `UBO_100`, `UBO_75`, `UBO_50`, `UBO_ 25`

Implementazione

In questa sezione vengono presentate le implementazioni delle query utilizzate per interagire con il nostro database. Le query sono progettate per estrarre, manipolare e analizzare i dati relativi alle aziende e alle loro relazioni con amministratori, azionisti, beneficiari effettivi, transazioni finanziarie e controlli di conformità KYC/AML.

Ogni query è strutturata per fornire risultati significativi che supportano l'analisi delle strutture proprietarie e delle attività aziendali. Le implementazioni sono ottimizzate per garantire efficienza e chiarezza, sfruttando appieno le capacità del linguaggio scelto e dell'architettura del database. I risultati ottenuti vengono presentati in un formato che facilita l'interpretazione e l'applicazione pratica degli insight derivati.

QUERY 1

Descrizione: Questa query è progettata per estrarre i dettagli di un'azienda specifica utilizzando il nome dell'azienda come criterio di ricerca. La query cerca il documento corrispondente nella collezione delle aziende e restituisce tutte le informazioni disponibili per l'azienda con il nome specificato

Script - Tempi di Esecuzioni

- **GitHub: Link_Query_1**

QUERY 2

Descrizione: Questa query recupera informazioni dettagliate su un'azienda specifica e sui suoi amministratori associati. Inizia cercando il documento dell'azienda in base al suo ID e, successivamente, raccoglie i dettagli degli amministratori elencati nel documento dell'azienda.

Script - Tempi di Esecuzioni

- **GitHub: Link_Query_2**

QUERY 3

Descrizione: Questa query estende la Query 2 includendo anche i dettagli degli UBO (Ultimate Beneficial Owners) che possiedono più del 25% dell'azienda. Inizia con il recupero delle informazioni aziendali e dei suoi amministratori, e poi cerca i dettagli degli UBO con una quota di partecipazione significativa.

Script - Tempi di Esecuzioni

- **GitHub: Link_Query_3**

QUERY 4

Descrizione: Questa query fornisce una vista dettagliata dell'azienda, inclusi amministratori, UBO con più del 25% di proprietà e un riepilogo delle transazioni finanziarie effettuate dall'azienda in un intervallo di date specificato. Oltre a recuperare i dettagli aziendali, degli amministratori e degli UBO, la query esegue un aggregazione per sommare l'importo delle transazioni nel periodo specificato.

Script - Tempi di Esecuzioni

- **GitHub: Link_Query_4**

QUERY 5

Descrizione: Questa query recupera i dettagli di un'azienda specifica, compresi gli amministratori associati, i beneficiari effettivi (UBO) con una partecipazione maggiore al 25%, le transazioni effettuate in una specifica valuta e data, e i suoi azionisti. L'obiettivo è confrontare le prestazioni di BaseX e Neo4j nell'esecuzione di query per il recupero di dati aziendali e delle relative entità correlate..

Script - Tempi di Esecuzioni

- **GitHub: Link_Query_5**

Conclusione

Il benchmark ha confrontato le prestazioni di Neo4j e BaseX nell'esecuzione di 5 query con complessità crescente, in cui la complessità aumentava aggiungendo progressivamente una "*join*" per ciascuna query. Le query sono state eseguite su 4 dataset con dimensioni diverse: 100%, 75%, 50% e 25%, dove il dataset al 100% conteneva 500.000 dati ed era identico sia su Neo4j sia su BaseX. Gli altri dataset contenevano una quantità di dati ridotta in base alla rispettiva percentuale. I risultati hanno evidenziato che Neo4j ha sempre superato BaseX in termini di velocità di esecuzione, indipendentemente dalla dimensione del dataset. La differenza di prestazioni è risultata ancora più marcata all'aumentare della complessità delle query, sottolineando la maggiore efficienza di Neo4j nella gestione di operazioni di "*join*" complesse, grazie alla sua architettura orientata ai grafi. Pertanto, Neo4j si dimostra più adatto a gestire query con molti collegamenti tra i dati, anche su dataset di dimensioni variabili.