UNIVERSITA’ DEGLI STUDI MILANO - BICOCCA

**Scuola di Scienze**

**Dipartimento di Informatica, Sistemistica e Comunicazione**

**Corso di laurea in Informatica**

**Sviluppo di una piattaforma per la gestione di dati RDF in formato Property Graph**

**Relatore**: *Maurino Andrea*

**Relazione della prova finale di:**

*Tommaso Sancini*

*Matricola 833304*

**Anno Accademico 2019-2020**

***1***

***Introduzione***

Negli ultimi anni una nuova tecnologia è stata introdotta da Google per potenziare il suo famoso motore di ricerca. Questa tecnologia consiste nel Knowledge Graph, fondamentalmente un grafico contenente tutte le principali entità nominative del mondo e le relazioni tra loro, in modo che il motore di ricerca sia in grado di identificare l'oggetto della query e suggerire tutti gli oggetti o le caratteristiche ad esso correlati. il Knowledge Graph non viene utilizzato come tecnologia per codificare la conoscenza generale e codificare attività indipendenti dal dominio come la PNL, al contrario, viene utilizzato per scopi commerciali specifici, in questo caso le raccomandazioni.

Vista l’efficacia di questo strumento si è assistito ad un fenomeno di aggregazione di conoscenze, da parte di molteplici aziende, in fonti sia strutturate che non strutturate in un unico e completo Knowledge Graph.

L’utilizzo massiccio dei Knowledge Graph ha portato ad immagazzinare dati molto sensibili generati ad esempio dal settore medico/sanitario, militare, governativo e di cybersecurity. La presenza di dati delicati genera dunque nuove problematiche rispetto al controllo delle informazioni da accessi non autorizzati insieme alla necessità di scalare ad un livello superiore. Ad ora non esiste nessuno standard su come rappresentare il Knowledge Graph, RDF potrebbe essere una soluzione ma utilizzando questa implementazione emergerebbero limiti nella scalabilità, e nella flessibilità, ma soprattutto i limiti dei sistemi software in grado di gestire RDF e il suo linguaggio di query SPARQL. Una soluzione a queste problematiche consiste nell’ utilizzo delle moderne tecnologie NoSQL, come ad esempio i nuovi formati come i Property Graph. Si tratta sempre di modelli a grafo ma con una rappresentazione più compatta.

Esistono molti prodotti di Property Graph pronti per l'industria, ma spesso offrono funzionalità di controllo degli accessi di base, inoltre causa del loro diverso modello, non hanno compatibilità nativa con RDF né supporto per l'inferenza.

In questo lavoro viene proposta una soluzione ai problemi di

interoperabilità tra il modello RDF e PG, in particolare viene sviluppato un aggiornamento, di un algoritmo già esistente, in grado di non sottrarre informazioni semantiche al modello RDF.

In questo capitolo verrà fornita una panoramica sul contesto dei database grafici e su gli obiettivi di questo elaborato mentre nel capitolo 2 andremo ad analizzare nel dettaglio i modelli grafici e i relativi GDBMS in modo tale da comprendere gli aspetti più critici della mappatura presentata nel capitolo 3. Infine, nel capitolo 4 vengono riassunti i risultati del lavoro e vengono proposti i lavori futuri.

**1.1 Contesto**

A metà degli anni '60, i database di navigazione come l'IMS di IBM supportavano strutture ad albero nel suo modello gerarchico, ma la rigida struttura ad albero poteva essere aggirata con record virtuali.

Le strutture a grafo potrebbero essere rappresentate nei database dei modelli di rete dalla fine degli anni '60 come il *Network Database Language* definito da CODASY nel 1969.

I grafi etichettati (labeled graph) potrebbero essere rappresentati in database di grafi dalla metà degli anni '80, come il *Logical Data Model.*

I miglioramenti ai database a grafo sono apparsi durante tutto il decennio degli anni ’90, dove vi era la necessità di indicizzare le pagine web.

Tra la metà e la fine degli anni 2000, sono diventati disponibili database di grafi commerciali con garanzie ACID come *Neo4j* e *Oracle Spatial and Graph.*

Negli anni 2010 sono diventati disponibili database commerciali di grafici ACID che potevano essere scalati orizzontalmente. Inoltre, SAP HANA ha portato le tecnologie in-memory e colonnari ai database grafici. Sempre negli anni 2010 sono diventati disponibili database multi-modello che supportavano modelli a grafo (e altri modelli come database relazionali o database orientati ai documenti), come OrientDB, ArangoDB e MarkLogic (a partire dalla sua versione 7.0). Durante questo periodo, i database di grafici di vario tipo sono diventati particolarmente popolari con l'analisi dei social network con l'avvento delle società di social media. Se i graph database fino a qualche anno fa erano poco conosciuti o visti come una tecnologia non ancora matura, oggi sono quindi visti come strettamente necessari per le aziende. Non a caso i graph database e le tecnologie correlate sono state inserite tra i [dieci trend](file:///C:UsersalsomAppDataLocalMicrosoftWindowsINetCacheContent.Outlook2J5AU5I7(https:www.digital4.bizexecutivedigital-transformationgartner-trend-analytics-ai)) che stanno cambiando il volto della gestione dei dati e degli analytics

**1.2 Obiettivi**

L'obiettivo principale di questo lavoro è fornire un GDMBS in grado di gestire Knowledge Graph di grandi dimensioni con informazioni sensibili, fornendo compatibilità con l'origine dati RDF. Questo può essere rifiutato in diversi aspetti:

• Scalabilità: questo è il primo e più importante aspetto. Deve essere fornita una soluzione che consenta la scalabilità orizzontale. Ciò significa che lo stesso software può essere in grado di gestire dati aggiuntivi se viene fornito più hardware, ad es. più server.

• Controllo degli accessi: poiché la soluzione deve gestire dati sensibili, è necessario fornire un modello di controllo degli accessi. Questo modello CA deve essere adatto a diversi casi d'uso

• Sicurezza: la sicurezza del sistema ha implicazioni anche sulla scelta delle tecnologie. In effetti, deve essere seguito il principio di Kerckhoffs che afferma che la sicurezza attraverso l'oscurità è una cattiva pratica. Quindi, deve essere preferito il codice open source. Inoltre, poiché grandi aziende ed enti pubblici non vogliono condividere le loro soluzioni, è necessario utilizzare software con licenza Apache 2.

***2***

***Graph database***

**2.1 Definizione**

La rappresentazione dei dati mediante grafi offre un’ alternativa al modello relazionale, ai [database orientati al documento](https://it.wikipedia.org/wiki/Base_di_dati_orientata_al_documento), o altri, come i sistemi ad [archivi strutturati](https://it.wikipedia.org/w/index.php?title=Archivio_strutturato&action=edit&redlink=1) basati su colonne.

Un grafo G(V, E) è una struttura composta essenzialmente da un insieme V di nodi e un insieme E di archi. In questa implementazione i concetti più importanti, le entità del modello ER, vengono modellate come nodi e il modo con cui le entità sono semanticamente associate generano degli archi. Le relazioni consentono di collegare direttamente i dati nell'archivio, più precisamente i nodi che hanno correlazioni sono fisicamente collegati nel database dunque, in molti casi, è possibile recuperarli con un'unica operazione. Le relazioni in un database a grafo sono una priorità quindi l’ accesso a tali relazioni è immediato quanto l’ accesso ai dati stessi. In altre parole, invece di calcolare la relazione come devono fare i database relazionali, i database a grafo leggono semplicemente la relazione dello storage.

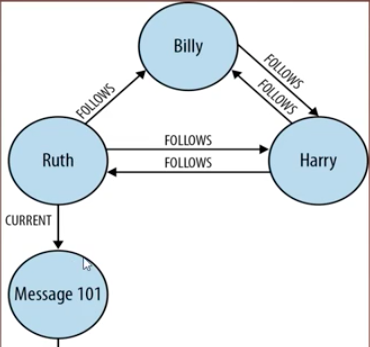
Le applicazioni di questo modello sono molteplici, in particolare troviamo i social, le raccomandazioni, gli ambiti geografici oppure il mondo finanziario.

**2.2 Proprietà**

Utilizzando i database a grafo è possibile sfruttare la conoscenza matematica su questo tipo di strutture, come ad esempio algoritmi ben consolidati in grado di ricavare risultati in tempi brevissimi.

Dato che il modello a grafo differisce, come vedremo, su molti aspetti rispetto alla classica struttura relazionale, esso viene considerato come un modello NoSQL.

**2.2.1 Schemaless**

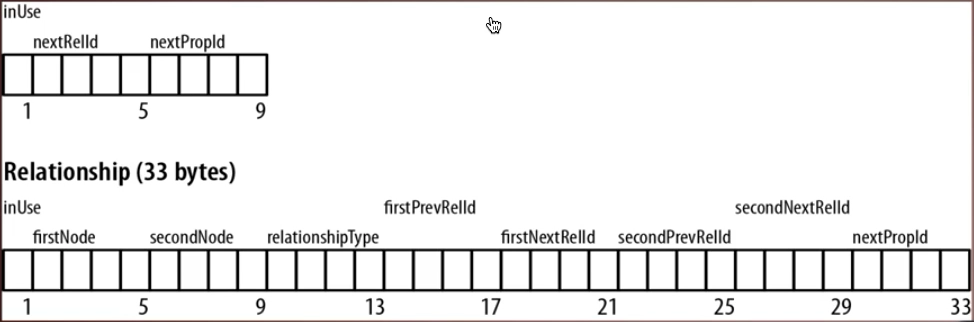
L utilizzo dei nodi è molto flessibile e ci permette di rappresentare con la stessa struttura diverse entità. Ad esempio, una persona ed i messaggi da lui inviati vengono rappresentati nello stesso modo, dunque non esiste una descrizione intensionale dei dati, il cosiddetto schema. Questo è uno dei motivi per la quale i database a grafo risulta molto semplice e apprezzato.

**2.2.2 Storage**

Nei database a grafo il meccanismo con la quale i dati vengono fisicamente memorizzati può variare.

* + - 1. ***Native Graph storage***

Sono ottimizzati e progettati per la gestione dei grafi, più precisamente alla base di questa implementazione risiede il principio di “*index free adjacency*” , dove i nodi e la relazione che li lega sono salvati uno dopo l'altro. Se volessimo utilizzare termine semplificativo possiamo dire che “il grafo viene srotolato”.



* + - 1. ***Non-Native graph storage***

Dipendono da un motore di un altro modello di basi di dati, come ad esempio relazionale, documentale. Alcuni memorizzano i dati del grafico in una tabella e svolgono le relative interrogazioni utilizzando il linguaggio SQL, trasformando il risultato secondo una modalità di visualizzazione a grafo con nodi e archi.

Altri utilizzano un archivio chiave-valore o un database orientato ai documenti per l'archiviazione, rendendoli intrinsecamente strutture NoSQL. Un nodo verrebbe rappresentato come qualsiasi altro archivio di documenti, ma i bordi che collegano due nodi diversi contengono attributi speciali all'interno del suo documento:

attributi \_from e \_to.

Ovviamente questo approccio impone un ulteriore livello di astrazione tra database, graph engine e il dispositivo su cui è memorizzato.

**2.2.3 Index free adjacency**

L'adiacenza senza indice impone ai nodi di avere indirizzi RAM fisici diretti che puntano fisicamente ad altri nodi adiacenti, ottenendo cosi una lettura molto rapida. Un sistema a grafo nativo con adiacenza senza indice non deve spostarsi attraverso nessun altro tipo di strutture di dati per trovare collegamenti tra i nodi. I nodi direttamente correlati inoltre vengono memorizzati nella cache in modo tale che una volta recuperato il nodo, la ricerca dei dati risulti ancora più veloce rispetto alla prima volta. Tuttavia, tale vantaggio ha un costo, l'adiacenza senza indice sacrifica l'efficienza delle query che non utilizzano attraversamenti di grafi. I database a grafo nativi utilizzano l'adiacenza senza indice per elaborare le operazioni CRUD sui dati archiviati.

**2.2.4 Diverse gestioni del grafo**

Con i database a grafo si introduce una modalità innovativa per gestire il grafo.

**2.2.4. T1: Graph Database OLTP**

Database che memorizzano i dati, implementano il modello e utilizzano un grafo per gestirlo. Consentono le cosiddette operazioni fondamentali CRUD(Create, Read, Update, Delede).

**2.2.4 T2: Graph Compute engine OLAP**

Insieme di programmi che utilizzando algoritmi su grafi svolgendo su di essi analisi.

### **2.3 Casi d’uso dei database a grafo**

I database a grafo funzionano meglio quando i dati **sono altamente connessi** dunque le informazioni da ricavare sono legate più alle relazioni che vivono tra le risorse che le risorse stesse. Ad esempio:

* *Social Network :* **I** database a grafo riducono la quantità di lavoro necessario per costruire le visualizzazioni di dati presenti nei social network come i feed di attività, o per determinare se è possibile conoscere una determinata persona**a seconda della sua vicinanza ad altri amici che vivono nella rete.**
* *Raccomandazioni:* Le risorse nel web negli ultimi anni vengono classificate, dunque l utilizzo di relazioni facilita i motori di ricerca o le pagine di e-commerce a proporre, consigliare articoli coerenti con gli interessi dell’ utente.
* *Geographic:* Il cosiddetto “grafo viario”, dunque la rete viaria costituita dall’ insieme dei segmenti stradali è appunto rappresentato come un grafo, sulla quale è possibile effettuare operazioni logistiche come la ricerca della strada più veloce oppure l instradamento dei pacchetti.
* *Financial:* Utile per implementare sistemi di “*Fraud detection*”, modellando ogni transazione finanziaria come un nodo e i collegamenti temporali tra una transazione e un'altra come relazioni, è possibile identificare dei pattern, dunque delle sequenze di operazioni che possono portare alla luce situazioni di frode fiscale.

Allo stesso modo, i database a grafo **sono adatti per applicazioni che gestiscono le relazioni o le interdipendenze tra entità.** Si trovano spesso questi database dietro sistemi di gestione dei contenuti e dei beni, sistemi di gestione delle identità e degli accessi e soluzioni di conformità normativa e gestione del rischio.

**2.4 Query**

Nel mondo dei graph database, ad ora, nessun linguaggio universale di query è stato adottato allo stesso modo di SQL per i database relazionali. Questo ha portato alla nascita di linguaggi di query completamente diversi tra loro, molto spesso strettamente legati a un prodotto come Gremlin, SPARQL o Cypher.

Dato che uno dei punti di forza del database a grafo consiste nel fatto che le **relazioni sono memorizzate nel database stesso** e pertanto non devono essere calcolate prima con la query, ne risulta una velocità di elevata performance anche in query complesse.

Mostriamo ora un esempio di come la stessa query verrebbe implementata su SQL, Cypher e SPARQL.

* SQL :

**SELECT** rel.per\_name

**FROM** people per

**JOIN** relatives **ON** (per.per\_id = relatives.per\_id)

**JOIN** people rel **ON** (rel.per\_id = rel.rel\_id)

**WHERE** per.per\_name = 'Mimmo';

* Cyper :

**MATCH** (per:person)-[:RELATIVE-**WITH**]-(rel:person)

**WHERE** per.name = "Mimmo"

**RETURN** rel.name

* SPARQL :
* Long Form

**PREFIX** **foaf**: <http://xmlns.com/foaf/0.1/>

**SELECT** ?name

**WHERE** { ?p **a** **foaf**:**Person** .

?p **foaf**:**name** "Mimmo" .

?p **foaf**:**relative\_of** ?r .

?r **foaf**:**name** ?name .

}

* Short form

**PREFIX** **foaf**: <http://xmlns.com/foaf/0.1/>

**SELECT** ?name

**WHERE** { ?p **foaf**:**name** "Mimmo" ;

**foaf**:**relative\_of** ?r .

?r **foaf**:**name** ?name .

}

* Hybrid SPARQL *(versione SQL implementata con SPARQL)*

**SELECT** people.name

**FROM** (

SPARQL **PREFIX** **foaf**: <http://xmlns.com/foaf/0.1/>

**SELECT** ?name

**WHERE** { ?p **foaf**:**name** "Mimmo" ;

**foaf**:**relative\_of** ?r .

?p­ **foaf**:**name** ?name .

}

) **AS** people ;

* Gremlin

**g**.**V**().**has**(**”name”,“Mimmo”**).**out**(**“relatives\_of”**).

**2.4.1 Algoritmi su grafi**

La mancanza di standard ha portato anche all’ utilizzo di **speciali algoritmi** in grado semplificare e snellire le complesse interrogazioni di dati. Tra i più importanti algoritmi figurano:

**2.4.1 A1: *DFS* *(depth first search)***

DFS è un algoritmo di visita di un grafo che effettua una visita in profondità di un grafo. Alla fine dell’esecuzione dell’algoritmo siamo in grado di determinare quante sono le componenti connesse del grafo ed a quale componente connessa appartiene ogni nodo. Quando la componente connessa contenente il nodo sorgente è stata del tutto esplorata, la prima chiamata di DFS-VISIT termina. A questo punto viene chiamata nuovamente sul primo nodo libero (WHITE) del grafo, che appartiene ad una componente connessa diversa da quella precedente. Quindi il numero di chiamate, all’interno di DFS, della procedura DFS-VISIT corrisponde al numero effettivo di componenti connesse.

**2.4.1 A2:** ***BFS* *(breadth first search****)*

BFS è un algoritmo di visita di un grafo che effettua una visita in ampiezza. Lo scopo di BFS è quello di scoprire tutti i vertici raggiungibili, partendo dal vertice sorgente . La scoperta avviene in ampiezza, nel senso che inizialmente vengono scoperti tutti i vertici a distanza di 1, poi a 2 e così via. Vengono scoperti sempre i cammini di lunghezza minima . L’algoritmo visita solo una delle componenti connesse del grafo.

**2.4.1 A3:** ***Algoritmo di Dijkstra***

L’algoritmo di Dijkstra risolve il problema dei cammini minimi da sorgente unica. Sostanzialmente vogliamo trovare un cammino minimo che va da un dato vertice sorgente s ∈ V a ciascun vertice v ∈ V in un grafo orientato pesato G = (V, E) nel caso in cui i pesi degli archi non siano negativi.

**2.4.1 A4:** ***Algoritmo di Bellman-Ford***

L’algoritmo di Bellman-Ford risolve il problema dei cammini minimi da sorgente unica nel caso generale in cui i pesi degli archi possono essere negativi . Svolge quindi lo stesso lavoro di Dijkstra, ma senza il vincolo che i pesi siano tutti non negativi.

**2.4.1 A5*: Algoritmo di Krusal e Prim***

Il problema della costruzione di alberi minimi riguarda solo i grafi G = (V, E) non orientati ,dotati di una funzione peso. L’obiettivo è trovare un albero di connessione minimo per il grafo, ovvero, dato un grafo con un determinato numero di archi, determinare un sottoinsieme di archi tale per cui i cammini tra i vari vertici sono minimi. I due algoritmi che analizzeremo, Kruskal e Prim, sono entrambi identificabili come greedy.

Gli algoritmi visti consentono di trovare modelli (graph patterns) e nodi limitrofi diretti e indiretti. Altri algoritmi permettono di calcolare il percorso più breve tra due nodi e di identificare cricche (sottoinsieme di nodi) e hotspot (informazioni altamente interconnesse).

**2.5. Modello a grafo vs modello relazionale**

I database a grafo sono un tipo di database NoSQL, creato per affrontare i limiti dei database relazionali. Mentre il modello a grafo definisce esplicitamente le dipendenze tra i nodi di dati, il modello relazionale collegai dati tramite connessioni implicite. In altre parole, le relazioni sono un cittadino di prima classe in un database a grafo e possono essere etichettate, dirette e assagnare proprietà. Possono scalare in modo più naturale a set di dati di grandi dimensioni poiché in genere non necessitano di operazioni di join, che spesso possono essere costose. Dipendendo meno da uno schema rigido, sono commercializzati come più adatti a gestire dati ad hoc e mutevoli con schemi in evoluzione. Questo è paragonato agli approcci relazionali in cui queste relazioni sono implicite e devono essere reificate in fase di esecuzione. I database dei grafici sono simili ai database dei modelli di rete degli anni '70 in quanto entrambi rappresentano grafici generali, ma i database dei modelli di rete operano a un livello inferiore di astrazione e mancano di un facile attraversamento su una catena di bordi. Al contrario, i sistemi di gestione dei database relazionali sono tipicamente più veloci nell'eseguire la stessa operazione su un gran numero di elementi di dati, consentendo la manipolazione dei dati nella loro struttura naturale. Nonostante i vantaggi dei database a grafo e la recente popolarità rispetto ai database relazionali, si raccomanda che il modello a grafo stesso non dovrebbe essere l'unica ragione per sostituire un database relazionale esistente. Un database a grafo può diventare rilevante se esiste una prova di miglioramento delle prestazioni per ordini di grandezza e latenza inferiore.

* 1. **Contesto rdf**

Nel 2001 Tim Berners-Lee ha proposto la sua visione [11] del web semantico. Da allora, molti dataset sono stati pubblicati nel formato grafico RDF secondo il principio Linked Open Data, quindi li chiamiamo Open Knowledge Graphs. Questo KG è stato spesso generato a partire da una fonte di testo strutturato come Wikipedia1 e contiene informazioni sulla conoscenza generale. Esempi di questo tipo di KG sono YAGO [54], DBPedia [8], Freebase [12].

1 wikipedia.org/

Knowldge Graph non rappresenta più solo la funzionalità offerta dal motore di ricerca di Google per aiutare gli utenti a cercare le informazioni più rapidamente e in modo più intuitivo. Knowledge Graph è diventata l’etichetta adottata da tutti i vendor di tecnologie del Web Semantico. Tecnologie e standard che hanno faticato ad emergere, come RDF, OWL, Sparql, per lungo tempo limitate al mondo accademico, vivono oggi un momento di fortissimo interesse, anche commerciale. Pensiamo alle specifiche RDF, Resource Description Framework, adottate come raccomandazione esattamente venti anni fa. Queste sono alla base del modello dati oggi etichettato come **Semantic Knowledge Graph** o più semplicemente come Knowledge Graph. Vendor come Stardog, Cambridge Semantics, Ontotext, Openlink hanno implementato questo modello che oggi è in competizione con il modello alternativo, più giovane di dieci anni e denominato **Property Graph** sposato da vendor come Neo4j, TigerGraph, ArangoDb, OrientDb, JanusGraph e altri.