ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA

LAUREA IN INGEGNERIA GESTIONALE

TESI DI LAUREA

In
Sistemi Informativi L

Interrogazione visuale per basi di dati a grafo: il sistema GraphVista

CANDIDATO

Andrea Marchese

RELATORE

Chiar.ma Prof. Wilma Penzo

Anno Accademico 2015/16

Sessione I

A Claudia.
Ai miei genitori.
Ai miei veri amici.
Alle persone che mi amano
e mi hanno amato davvero.

“Non siamo fatti per stare da soli
ma nemmeno per stare con chiunque”.

Indice

Indice delle figure

Introduzione

Oggi i sistemi informativi delle aziende o di qualsiasi altra realtà si trovano a gestire quantità di informazioni sempre più elevate e complesse.
L'insieme delle informazioni, sotto forma di dati, vengono immagazzinate all'interno di basi di dati o più comunemente database[1]: archivi di dati le cui informazioni sono strutturate e correlate fra loro seguendo un modello logico.

Oggi vi è sempre più la necessità di utilizzare e sfruttare database più efficienti che siano in grado di gestire la complessità della natura dei dati, che sempre più spesso sono molto eterogenei nel formato e nei contenuti e presentano numerose relazioni fra di essi. A tal proposito una soluzione molto promettente è quella che fa uso di database a grafo.
Fra le varie alternative di basi di dati esistenti emergono sempre vari limiti fra cui:

- Elevata tempistica di interrogazione del database.
- Scarsa interazione dell'utente nell'interrogazione.
- Richiesta di competenze di alto livello per formulare interrogazioni complesse.

Al fine di superare questi e tanti altri limiti relativi alla formulazione delle interrogazioni è stato sviluppato il sistema GraphVista, il quale si basa su interrogazioni visuali e più interattive di basi di dati a grafo.

La tesi è cosi organizzata:

Primo capitolo: definizione di grafo e teoria dei grafi. Tale capitolo ha l'obiettivo di fornire al lettore le conoscenze necessarie per poter comprendere al meglio l'intero elaborato.

Secondo capitolo: introduzione dei database a grafo. In questo capitolo vengono descritte le caratteristiche cardine delle basi di dati a grafo e descritti i sistemi che gestiscono tali database.

Terzo capitolo: è il capitolo cardine della tesi. In esso viene descritto il sistema GraphVista, ne vengono analizzate le caratteristiche e il funzionamento.

Quarto capitolo: in questo capitolo viene effettuato un confronto tra la modalità di formulazione di interrogazioni di tipo grafico e la modalità che fa uso di un linguaggio. In particolare viene presa in esame una query di esempio e ne vengono mostrate le forme, evidenziandone le differenze e le diverse modalità di esecuzione.

Infine vi è la fase conclusiva della tesi in cui vengono riportate le riflessioni sul lavoro svolto.

Capitolo I

Grafi

In questo capitolo verranno definiti i grafi e ne verranno illustrate le caratteristiche principali. Seguirà una veloce classificazione e un'introduzione alla teoria dei grafi.

1.1 I grafi

I grafi sono strutture matematiche la cui applicazione risiede su una vasta gamma di aree fra cui la topologia e la combinatoria.
Questi, inoltre, sono elementi cardine su cui si basano modelli sviluppati in campo ingegneristico, chimico (rappresentazione molecolare), matematico (ricerca operativa e programmazione lineare).
Infine, grazie alla loro ottima capacità di schematizzazione, vengono applicati anche in campo informatico.

1.1.1 Definizione

Viene definito Grafo[1] una coppia ordinata *G=(V,E)* di insiemi, con V insieme di nodi ed E insieme di archi, tali che gli elementi di E siano coppie di elementi di V.
Come è possibile vedere in **figura 1.1** sostanzialmente un grafo risulta essere un insieme di elementi, detti NODI, collegati tra loro, mediante l'utilizzo di linee dette ARCHI, da delle relazioni.

1.2 Teoria dei Grafi

Dopo aver definito un grafo, è necessario capire in che modo questi possono tornare utili nei campi applicativi citati in precedenza.
Per far ciò diventa fondamentale saper analizzare, quindi scomporre, i grafi.
A tal proposito ci può aiutare la **Teoria dei Grafi** [1], ovvero la disciplina che si occupa del loro studio.

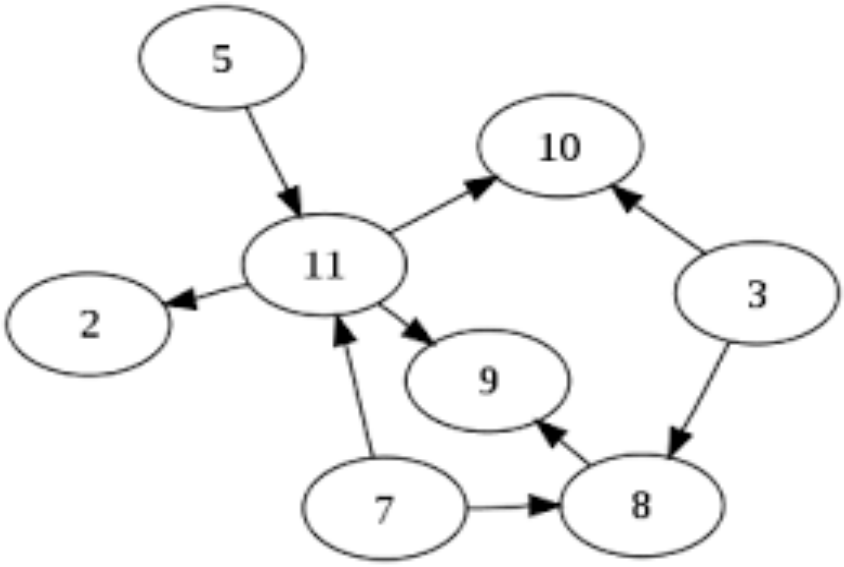


Figura 1.1 : Grafo formato da 8 Nodi e 9 Archi

1.2.1 Cenni storici

Le prime applicazioni dei grafi risalgono alla fine del XVIII secolo e riguardano principalmente rappresentazioni grafiche di scenari atti a risolvere banali problematiche.
Successivamente i grafi vennero presi in considerazione come entità matematiche e utilizzati nella risoluzione di problemi inerenti la geometria topologica, ma è a partire dalla seconda metà del XX secolo, grazie anche all'introduzione dei primi computer e allo sviluppo della combinatoria, che la teoria dei grafi viene presa seriamente in considerazione e utilizzata come supporto alla formulazione di modelli e/o algoritmi volti ad applicativi sempre più complessi.

1.2.2 Terminologia

Come già detto in precedenza, i grafi sono strutture costituite da oggetti (vertici o nodi) e collegamenti fra essi (archi).
Per capire al meglio le diverse proprietà che ogni vertice può assumere o le peculiarità che caratterizzano tipi di grafi differenti, è necessario introdurre diversi termini con i quali poter distinguere i vari grafi associando ad ognuno di essi caratteristiche e funzionalità diverse.

Chiamando **estremi** due vertici connessi da un arco, distinguiamo:

- Cappio: arco con due estremi coincidenti.
- Multiarco: più archi collegati agli stessi estremi.

Tale distinzione è importante per arrivare alla prima macro classificazione dei grafi.
Chiamiamo **grafo semplice** ogni grafo non avente né cappi né multiarchi, mentre la configurazione avente almeno uno dei due dà origine ad un **multigrafo** [1].
In un grafo è importante individuare le varie cardinalità che lo caratterizzano in quanto gli insiemi, nella maggior parte dei casi, risultano finiti. Individuiamo cosi:

- **Ordine** di un grafo: numero di vertici
- **Dimensione** di un grafo: numero di archi
- **Grado** del vertice: numero di archi che si connettono ad un vertice

Una ulteriore distinzione va fatta fra **grafi nulli** e **grafi completi**. I primi sono grafi il cui insieme degli archi risulta essere vuoto, mentre un grafo viene detto completo se presi due suoi nodi questi risultino connessi da un arco.
Fra tutte, la distinzione più nota è quella che divide i grafi orientati da quelli semplici.
Per distinguerli è necessario introdurre la definizione di **arco orientato**[1] **.**

Un arco orientato è un arco caratterizzato da una direzione, ovvero è un arco avente una coda e una testa, la coda è la parte di arco che segue il nodo da cui si parte mentre la testa, solitamente rappresentata da una freccia, raggiunge il nodo in cui si entra.
Viene quindi definito **grafo orientato** un grafo i cui nodi sono collegati da archi orientati.
La configurazione di grafo che non presenta nodi collegati da archi orientati viene chiamata **grafo non orientato** o indiretto.
Presi due nodi A e B di un grafo indiretto, il collegamento a-b presenterà le stesse caratteristiche del collegamento b-a.

1.3 Classificazione

Oggi è possibile riconoscere varie tipologie di grafi, le quali vengono distinte in base a particolari proprietà o configurazioni assunte.
Eccone alcune [1]:

- **Grafo Arricchito:**

  Un grafo viene definito **arricchito** quando può essere visto come 'arricchimento' delle semplici configurazioni dei grafi diretti e grafi indiretti. Per esempio grafi aventi un nodo colorato piuttosto che vertici caratterizzati da valori numerici.

- **Grafo Bipartito**

  È un grafo avente l'insieme dei vertici scomponibile in due sottoinsiemi tali che, come si può vedere in **figura 1.2**, ogni nodo dei due sottoinsiemi è collegato a nodi non appartenenti al suo sottoinsieme.

- **Grafo Aleatorio**

  È definito come una variabile aleatoria rappresentabile attraverso dei grafi.
  Di tali grafi viene analizzato il comportamento asintotico, ovvero viene analizzata una successione di grafi aleatori avente il numero di vertici tendente ad infinito.

- **Grafo Convesso e Grafo Biconvesso**

  Per capire con esattezza quando un grafo viene definito convesso o biconvesso è necessario introdurre la definizione di **Proprietà di Adiacenza.**
  Preso un grafo Bipartito, quindi avente due sottoinsiemi di nodi che chiamiamo A e B, un ordinamento del sottoinsieme A ha la proprietà di adiacenza, se per ogni nodo appartenente a B, i vicini di tale nodo in A sono consecutivi nell'ordinamento stesso.

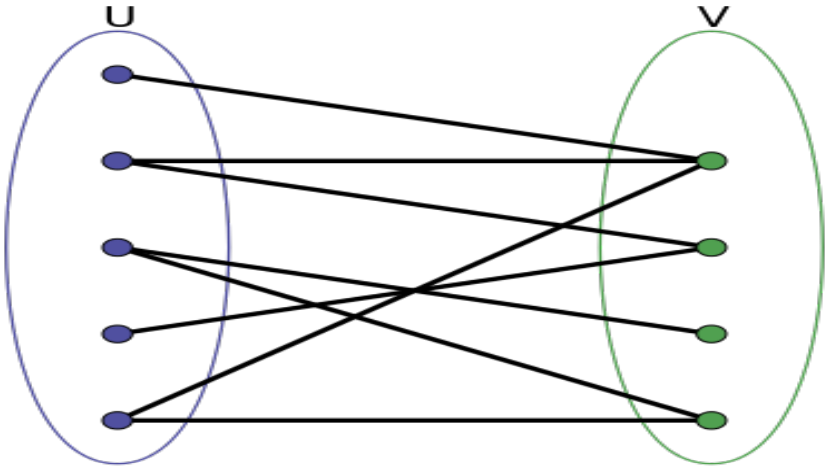


Figura 1.2: grafo bipartito con U e V sottoinsiemi di Nodi.

Enunciata la proprietà, possiamo definire che un grafo bipartito è convesso se presenta un ordinamento di uno dei due sottoinsiemi di nodi che rispetta la proprietà suddetta, qualora in entrambi i sottoinsiemi vi fosse un ordinamento che rispetti la proprietà di adiacenza, il grafo sarebbe definito biconvesso.

- **Grafo Regolare**

  Un grafo risulta essere regolare quando tutti i vertici di cui è composto presentano lo stesso grado, ovvero lo stesso numero di archi.

- **Grafo Cubico**

  I Grafi in cui tutti i nodi sono caratterizzati da un grado pari a 3, viene definito grafo cubico.
  All'interno di questa categoria possiamo individuare i **grafi bicubici**, ovvero i grafi bipartiti cubici.

- **Grafo Planare**

  Viene definito planare quel grafo che, raffigurato in un piano, non presenta archi intersecati.

- **Grafo Duale**

  Possiamo definire un grafo duale soltanto in relazione ad un altro grafo planare o comunque raffigurabile su uno spazio euclideo.
  Preso un grafo planare Z, il rispettivo duale è un grafo avente un nodo per ogni regione di Z e un arco passante per ogni arco di Z.

1.4 Connettività e Accoppiamento

Dopo aver definito un grafo, averne studiato le caratteristiche e in base ad esse averne individuato diverse categorie, bisogna cominciare ad analizzarli da un punto di vista più pratico.
A cosa servono? Come possono tornarci utili?
Per rispondere con esattezza a tali domande dovremo aspettare il prossimo capitolo.
Adesso possiamo soltanto vedere i grafi come grandi contenitori, entro cui poter memorizzare, leggere e spostare dati.

1.4.1 Connettività

Proprio per far in modo che tale contenuto possa essere spostato è necessario capire in che modo possiamo "viaggiare" all'interno di un grafo e quindi in che modo è possibile visionare i vari nodi.
Per poter accedere ai vari nodi è inevitabile seguire un **percorso**, ovvero una sequenza di nodi e rispettivi archi che li collegano.
Se quest'ultimo presenta i lati a coppia distinti tra loro viene chiamato **cammino**.
L'ultimo concetto è legato alla definizione di **connettività**, infatti due vertici qualsiasi appartenenti ad un grafo sono **connessi** se e solo se fra i due vi è un cammino che li collega, in caso contrario i due vertici risultano **sconnessi.**
Viene quindi definita la connettività [1] di un grafo come la cardinalità di un insieme A non vuoto (sottoinsieme dei nodi del grafo in questione) tale che il grafo, a cui sono stati tolti i nodi appartenenti all'insieme A, risulti sconnesso.

1.4.2 Accoppiamento e Ricoprimento

Per individuare, all'interno di un database ad esempio, la correlazione fra dati e quindi fra i contenitori di dati (nodi) è necessario studiarne l'accoppiamento e il ricoprimento.
Preso in considerazione un grafo semplice, viene definito **accoppiamento** [1] o matching un insieme *M=(A,B),* con A insieme di vertici e B insieme di archi, composto da coppie di vertici presi a due a due e dai rispettivi archi che collegano tali vertici se ogni vertice appartenente ad A ha grado O oppure 1,ovvero è un insieme di archi senza vertici comuni come è possibile vedere in **figura 1.3**.
Definiamo inoltre il **ricoprimento** come quell'insieme H non vuoto di nodi (sottoinsieme dei nodi del grafo) tale che ogni arco del grafo è incidente in almeno un nodo appartenente ad H.

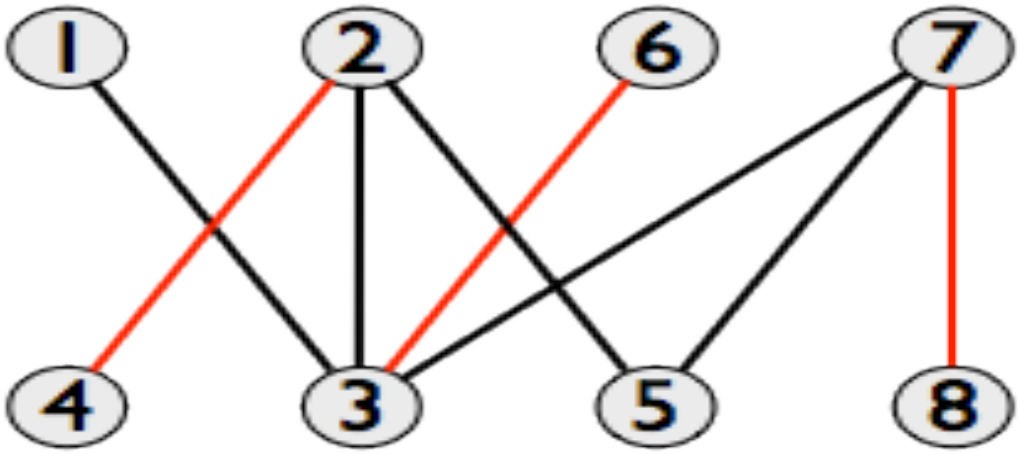


Figura 1.3 Grafo in cui è possibile individuare l'insieme di accoppiamento

1.5 Visita e Tecniche di esplorazione

I grafi possono contenere una notevole quantità di vertici e di archi. Motivo per cui diventa necessario applicare una metodologia accurata per visitarli.
Per **visita** [2] ci si riferisce alla possibilità di esaminare i nodi e gli archi di un grafo in maniera sistematica.
Vi sono due tipologie di visita:

- **Visita in ampiezza (BFS)**

  Definito un nodo sorgente all'interno di un grafo, con tale visita verranno esplorati gli archi del grafo a partire dal livello del nodo sorgente.
  Viene calcolato il numero minimo di archi tra il nodo sorgente e tutti gli altri e l'esplorazione sarà impostata partendo dai grafi con la distanza più bassa dal nodo sorgente.

- **Visita in profondità (DFS)**

  Anche con tale visita è necessario definire un nodo sorgente. La visita verrà eseguita andando il più possibile in profondità.
  In pratica, partendo dalla sorgente, dato un nodo v appena scoperto si prosegue, a partire da v, sui suoi archi ancora sconosciuti. Esplorati tutti gli archi si ritorna al nodo da cui è stato scoperto v al fine di esplorare, se vi sono, gli ulteriori suoi archi.
  Le esplorazioni procedono fino a che non si sono esplorati tutti i nodi raggiungibili dalla sorgente. Qualora vi fossero nodi non adiacenti la visita continuerebbe andando a scegliere, fra i nodi non ancora esplorati, una nuova sorgente.
  Naturalmente la visita avrà fino quando tutti gli archi risulteranno esplorati.

Entrambe le visite, per mantenere traccia delle esplorazioni eseguite, fanno uso di "flag" ai nodi, ovvero questi assumono colori diversi, come è evidente in **figura 1.4,** in base allo stato dell'esplorazione.

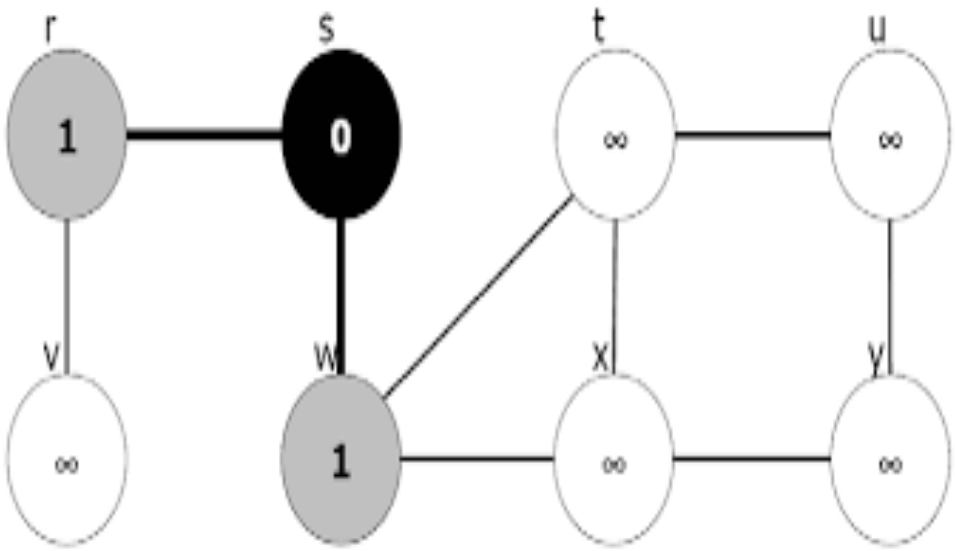


Figura 1.4 Grafo in cui è possibile individuare i diversi "flag" utilizzati durante la visita.

In particolare:

- ❖ Bianchi: non ancora scoperti.
- ❖ Grigi: scoperto ma l'esplorazione di uno dei suoi archi non è terminata.
- ❖ Neri: l'esplorazione lungo i suoi archi è completa.

Più grande è un grafo, in termini di nodi e correlazioni, ovvero archi, più sarà onerosa l'intera visita e quindi l'esplorazione.
Oggi gli istituti di ricerca focalizzano la loro attenzione soprattutto sulle tecniche di esplorazione al fine di ottimizzarle e renderle meno dispendiose in termini di 'tempo-visita'.
In particolare dal 2000 sono stati fatti notevoli passi avanti nell'esplorazione dei grafi.
Sono state proposte, per grafi che sono troppo grandi, le visualizzazioni in contemporanea, ovvero l'esplorazione si fonda sul principio del raggruppamento.
Vengono formati "gruppi" di nodi al fine di poterli esplorare insieme.

Fra le varie tecniche possiamo citare [3]:

- **Graph Sampling**

  Con tale tecnica viene evitato di raffigurare per intero il grafo. Si tende ad individuare campioni o filtrare sottoinsiemi più o meno rappresentativi ed analizzarli.
  Il problema principale è quale strada seguire per determinare il campione o sottoinsieme che sia.
  Vi si possono scegliere approcci sia deterministici che stocastici al fine di risolvere tale problema. Il focus principale è quello di scegliere in maniera ottimale il campione in modo da rappresentare nel modo migliore possibile le caratteristiche del grafo di partenza.

- **Graph Filtering**

  Tale tecnica si basa sul filtraggio approssimativo del grafo al fine di ridurlo, senza però alterare le proprietà e mantenendo la sua struttura essenziale.
  In particolare, semplicemente disponendo gli archi in maniera diversa si avrebbe una visione più chiara dell'intero grafo.

- **Graph Partioning**

  Un ulteriore semplificazione visiva del grafo è possibile applicando metodi di partizionamento su di esso al fine di visualizzare separatamente le diverse partizioni.
  Un elevato partizionamento può risultare costoso da un punto di vista computazionale e il numero di partizioni non può essere dettato dal caso in quanto partizioni troppo piccole potrebbero risultare povere dal punto di vista informativo.
  Tali tecniche però richiedono un elevato tempo di attuazione, motivo per cui spesso vengono utilizzate partizioni precalcolate.

- **Graph Clustering**

  Un ulteriore approccio si basa sulla creazione di cluster.
  Vengono raggruppati i nodi in base agli attributi simili anche con l'ausilio di software di elaborazione analitica online.
  Tali software permettono di combinare informazioni strutturali e produrre una versione ridotta del grafo formata da più insieme di nodi legati da attributi comuni.
  Spesso questi software intervengono nella matrice di incidenza associata al grafo in cui individuare i cluster è molto più semplice.

È importante puntualizzare che ognuna di queste tecniche non esclude l'altra.
Tutte o alcune di esse possono essere applicate sequenzialmente arrivando a risultati ottimali, ovvero la riduzione di grafi inizialmente di difficile lettura.

1.6 Implementazione

Dopo aver visto varie rappresentazioni e le tecniche con cui è possibile esplorare un grafo è possibile analizzarlo e utilizzarlo per la lettura/scrittura di dati.
Nei casi di grafi con pochi elementi le operazioni suddette potrebbero essere svolte dall'uomo ma per grafi contenenti milioni di dati e quindi con elevati numeri di archi e nodi la lettura, eseguita dall'uomo, potrebbe essere molto complicata.
Per tale motivo oggi vi sono dei software in grado di leggere molto rapidamente grafi, quindi informazioni, molto ampi.
Tali software però non leggono direttamente il grafo come noi l'abbiamo definito ma sfruttano rappresentazioni diverse, infatti le informazioni dei grafi possono essere inserite all'interno di particolari strumenti chiamati [1]:

- Lista di adiacenza

- Matrice delle adiacenze

1.6.1 Lista di adiacenza

Viene associata ad ogni vertice del grafo una lista, come è possibile vedere nella **figura 1.5**, che elenchi tutti i vertici con cui lo stesso è connesso.
Memorizzando tutte le liste per ogni nodo del grafo possiamo ottenere una descrizione univoca del grafo stesso.
È una rappresentazione molto immediata e facile da realizzare però richiede uno spazio in memoria abbastanza elevato e soprattutto il tempo impiegato risulta direttamente proporzionale al numero dei nodi del grafo. Per tale motivo non conviene utilizzarla per grafi di grandi dimensioni.
Inoltre nel caso di grafi orientati l'efficienza risulterebbe dimezzata in quanto andrebbero individuate due liste per ogni nodo: una per gli archi entranti e l'altra per gli archi uscenti.

1.6.2 Matrice delle adiacenze

È una particolare struttura spesso utilizzata per rappresentare grafi più o meno complessi. È di gran lunga la rappresentazione più utilizzata in campo informatico poiché è più vicina ai linguaggi utilizzati negli algoritmi e nei modelli che fanno uso di grafi.
Consiste in una matrice binaria quadrata avente come indici di righe e colonne i 'nomi' dei nodi del grafo.
La matrice verrà riempita, come da **figura 1.5**, con 0 o 1. In particolare nel posto (i,j) della matrice troveremo 0 quando fra il nodo i e il nodo j non vi è connessione, nel caso in cui fra i due nodi vi è collegamento la matrice viene riempita con 1.
A volte al posto degli uni in matrice possono essere presenti altri valori. In questi casi i valori corrispondono al peso, ad esempio la distanza nel caso di nodi rappresentanti luoghi, che viene dato al collegamento fra i nodi, tali matrici prendono il nome di Matrici di Markov [1].

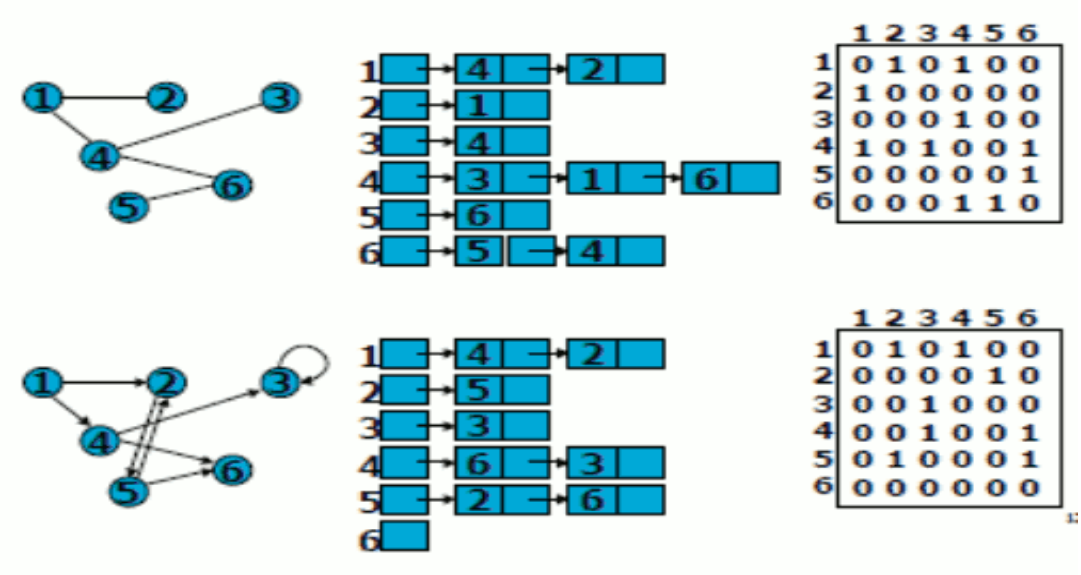


Figura 1.5 Semplice grafo con la rispettivamente la lista di adiacenza e matrice delle adiacenze

Le due rappresentazioni occupano quantità di memoria significativamente diversa. Infatti la memoria occupata dalla lista di adiacenza risulta essere una funzione di un valore dato dalla somma della cardinalità dei nodi e quella degli archi, mentre la memoria occupata dalla matrice risulta essere funzione del quadrato della cardinalità dell'insieme dei vertici.
La lista quindi risulta essere adatta alla rappresentazione di grafi sparsi o comunque con un numero di archi limitato mentre la matrice risulta idonea per grafi con un numero notevole di collegamenti.

1.7 Applicazioni

La rappresentazione mediante grafi è molto efficiente, motivo per cui può essere utilizzata per descrivere situazione facenti parte dei più svariati campi applicativi.
Per esempio sono alquanto implementati negli ipertesti (ad esempio Wikipedia) ove vi si trovano parecchi dati messi in relazione fra loro mediante l'uso di parole chiave.
In questo caso i nodi del grafo rappresenterebbero i vari documenti mentre gli archi corrisponderebbero ai link e quindi ai collegamenti fra i diversi documenti.

La struttura dei grafi orientati viene molto utilizzata nel campo delle scienze e dell'ingegneria per rappresentare modelli utili a descrivere i comportamenti di molti sistemi, inoltre hanno trovato successo nel campo della ricerca operativa, mediante la raffigurazione di diagrammi di flusso utili alla simulazione degli scenari più svariati.
La loro facilità di schematizzare i dati e la facilità con cui è possibile individuare i diversi collegamenti fra più basi di dati li rende molto utilizzati nel campo dell'informatica, in particolare negli schemi E-R (entità-relazione).
L'esigenza di organizzare in maniera ottimale il flusso di dati nel mondo del web ha fatto si che gli istituti di ricerca informatica si ponessero l'obiettivo di implementare paradigmi gestionali di basi di dati più efficienti.
Come vedremo nei prossimi capitoli, i paradigmi su cui sono focalizzate le maggiori ricerche implementano database a grafo.
Vi è però la certezza che tali strumenti possano essere migliorati attraverso lo sviluppo di ulteriori algoritmi finalizzati alla manipolazione più efficiente di tali oggetti.

Capitolo II

Base di dati a Grafo

In questo capitolo verranno introdotti i Database, soffermandosi in particolare sulle basi di dati organizzate a grafo.
Verranno classificati ed enunciati diversi modelli di riferimento, il linguaggio utilizzato nelle interrogazioni e il sistema attraverso cui vengono gestiti i database.
Infine verranno descritte e comparate alcune basi di dati a grafo.

2.1 Base di dati (Database)

Per capire come funziona una base di dati gestita mediante i grafi è necessario innanzitutto avere chiaro in mente cosa si intende per base di dati o database.
Il database è definito come insieme di strutture, come è raffigurato in **figura 2.1**, contenenti svariate informazioni sotto forma di dati [1].
Tali dati sono gestiti in modo tale da essere sempre disponibili ai vari processi che hanno bisogno di particolari informazioni come input.
Risulta quindi importante saper sfruttare tali "collezioni" di dati in modo da aver a disposizione nel minor tempo possibile le informazioni di cui abbiamo bisogno.
È necessario quindi saper, come si dice in gergo, interrogare un database, interfacciarsi con esso.
Ciò viene svolto attraverso l'implementazione di particolari linguaggi definiti **query language** e l'utilizzo di specifici software adibiti alla gestione del database e dell'interazione con l'utente e/o altre applicazioni.
Quindi generalmente, col termine database, si può indicare allo stesso modo:

- **Archivio fisico**: sistema hardware contenente i dati e il processore utilizzato per l'elaborazione degli stessi.

Figura 2.1 Stereotipo di una base di dati

- **Archivio logico**: insieme dei dati e dei software utilizzati per la loro gestione.

Ci si riferisce quindi ad un insieme di dati e al modo con cui questi vengono organizzati.
I software adibiti alla gestione dei dati svolgono diverse funzioni, fra cui [1] :

- **Definizione dati**: viene definita l'organizzazione dei dati all'interno del database.

- **Aggiornamento**: funzione attraverso cui vengono modificati o cancellati vari dati.

- **Recupero**: vengono forniti i dati in una forma utilizzabile dai software applicativi.

- **Amministrazione**: funzione cardine che si occupa della registrazione dei vari utenti e della sicurezza dei dati all'interno del database

Grazie alla loro funzione di "contenitore", i database hanno riscontrato maggiore successo nel campo amministrativo e in tutti gli applicativi in cui è richiesta la memorizzazione e l'utilizzo di una molteplicità di dati differenti come ad esempio un sistema di prenotazione dei voli, sistemi bibliotecari o sistemi di gestione dei contenuti dei siti web.

2.2 Base di dati a grafo

Introdotte le basi di dati è arrivato il momento di soffermarci sui database organizzati con i grafi.
Tali tipi di database sfruttano i grafi per svolgere le loro funzioni.
L'utilizzo dei nodi e degli archi per memorizzare e rappresentare le informazioni è una valida alternativa ad altre strutture organizzative dei database, come ad esempio quella relazionale che si serve di tabelle, quelle basate su documenti o archivi strutturati.
Generalmente i database a grafo presentano diversi vantaggi rispetto al database relazionale [1]:

- Maggiore velocità nell'associazione dei dati grazie allo sfruttamento diretto dei collegamenti (archi) senza dover passare dalle operazioni di join.

- Minore dipendenza da rigidi schemi E-R.

- Maggiore adeguatezza nella gestione di dati mutevoli in quanto l'individuazione dei dati stessi risulta meno onerosa.

Oltre ai suddetti vantaggi, è giusto sottolineare come tali basi di dati possano incorrere in piccole difficoltà quando la mole di dati risulta essere assai elevata. In questi casi la struttura relazionale risulta essere più efficiente.
Dopo esserci introdotti nel mondo dei database a grafo è bene scendere nel dettaglio. Adesso infatti, ci concentreremo sugli aspetti più applicativi di tali database.

Generalmente i modelli a cui si fa riferimento per la loro implementazione sono due:

- Property graph model
- Resource description framework graph

Il primo fa riferimento ai grafi i cui collegamenti sono legati a particolari proprietà, ovvero ai vari archi viene assegnato un peso che caratterizza quel collegamento.
Il secondo invece è il modello implementato nel *web semantico* ed è più conosciuto con la sigla RDF. Tale modello permette la codifica e quindi il riutilizzo più efficiente di dati attraverso un'identificazione standard. In pratica tutte le diverse tipologie di dati vengono convertite in un formato standard adatto all'interrogazione (per esempio attraverso motori di ricerca) e all'elaborazione automatica.
Un'applicazione del modello RDF è riportata in **figura 2.2,** ove risalta il vantaggio di utilizzare una codifica standard.
La figura esemplifica il collegamento di più dati di un utente all'interno di un social network. In particolare viene evidenziata la possibilità di accedere, mediante link di collegamento, a varie pagine contenenti dati di diversa tipologia (video, immagini, pagine HTML).
Col passare degli anni sono stati affinati i modelli già esistenti e si è tentato di svilupparne di nuovi.
Oggi distinguiamo [1]:

- **Modello dei dati di grafo basilare**: modello semplice in cui archi e nodi sono descritti in una legenda apposita.
- **Modello dei dati di grafo a ipernodo**: modello costituito da molti sottoinsiemi di archi e nodi, chiamati appunto ipernodi, attraverso cui è possibile rappresentare oggetti più complicati ed eredità complesse.
- **Modello dei dati RDF**: utile nella codifica dei dati nel web mediante la modellazione dei grafi in insiemi di tuple contenenti soggetto, predicato e oggetto. È raccomandato da una importante organizzazione internazionale come il *World Wide Web Consortium.*
- **Modello dei dati di grafo di proprietà** (Property graph model): come descritto precedentemente, si basa sull'assegnazione di particolari proprietà ai collegamenti dei vari nodi.

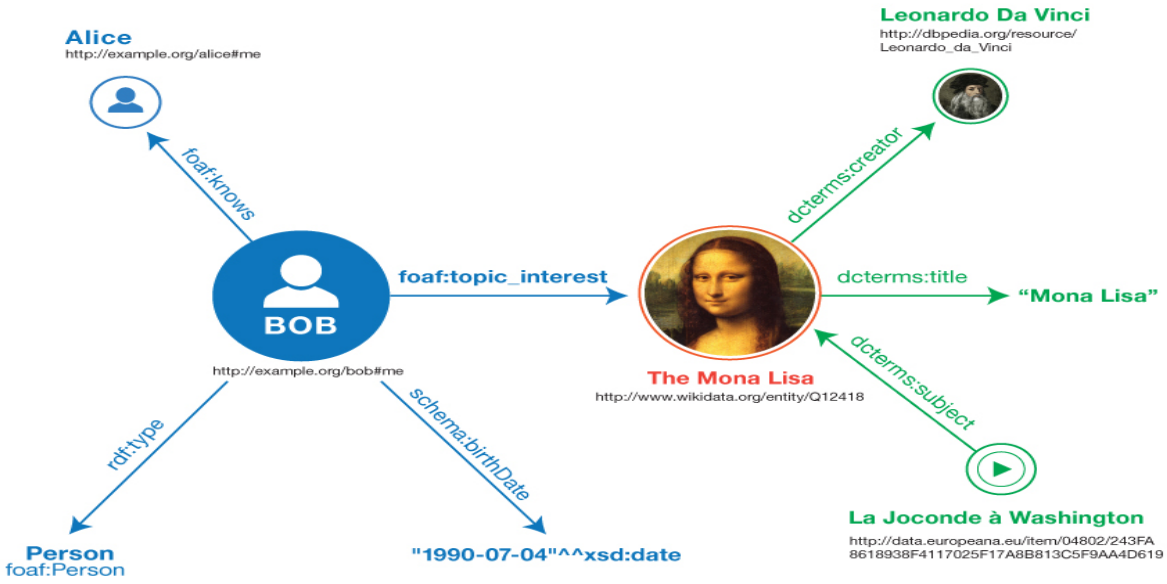


Figura 2.2 esemplificazione struttura dati organizzata mediante modello RDF

2.3 Interrogazioni di base di dati a grafo

Una volta implementato il database bisogna utilizzarlo, ovvero bisogna interrogarlo in modo che possa restituire i dati richiesti.
Ciò è possibile attraverso particolari linguaggi definiti *Query language*.
Uno dei primi linguaggi adottati è stato il **GraphQL**[4], il quale è stato progettato per eseguire interrogazioni di tipo *pattern matching*, cioè interrogazioni che restituiscono dati che soddisfano determinate proprietà. Quest'ultime vengono inserite dall'utente nella fase di querying.
Tale linguaggio però risulta poco efficace nelle interrogazioni richiedenti l'esecuzione di un particolare percorso all'interno del grafo, motivo per cui varie aziende (Facebook) partendo dal GraphQL hanno introdotto linguaggi utilizzabili soltanto per l'interrogazione del proprio database.

Tali linguaggi utilizzano interfacce di tipo SQL e offrono una serie semplice e limitata di query.
Il linguaggio più diffuso è sicuramente **SPARQL** (*SPARQL Protocol and RDF Query Language).*

È stato definito standard ufficiale dal *W3C (World Wide Web Consortium)* per l'interrogazione e l'estrazione delle informazioni dei database gestiti mediante i modelli a grafo RDF.
Questo linguaggio descrive i collegamenti fra i dati presenti nel database attraverso tuple formate da 3 elementi: soggetto, predicato, oggetto.
Questi elementi permettono la formazione di interrogazioni basate su connettivi logici e/o algoritmi ricorsivi.

Un buon linguaggio di interrogazione deve rispondere alle esigenze degli utenti in modo chiaro e conciso. Per tale motivo sono state apportate migliorie al linguaggio SPARQL fondando il linguaggio **G-SPARQL** [4] **.**
Questa versione risulta avere una sintassi molto più semplice e inoltre permette l'esecuzione di interrogazioni che richiedono una visita del grafo molto profonda in quanto include nuovi costrutti non supportati dalla versione standard.
È possibile ancora eseguire interrogazioni le cui condizioni sono riscontrabili in attributi presenti su vertici diversi.

In **figura 2.3** sono riportati alcuni esempi di un'interrogazione di un database mediante l'utilizzo di tre linguaggi: SPARQL, G-SPARQL e Cypher. In particolare è possibile notare, a prescindere dalla diversa sintassi dei vari linguaggi, l'utilizzo di parole chiave come FROM, SELECT e WHERE. Con la prima vengono indicati al sistema gli attributi da restituire, la seconda indica il "luogo" all'interno del database entro cui focalizzare la ricerca e l'ultima riporta la proprietà da soddisfare durante la ricerca.

```
## SPARQL 1.0 and SPARQL 1.1
SELECT ?X
FROM <http://www.socialnetwork.org>
WHERE { ?X sn:firstName "James" }
## G-SPARQL
SELECT ?X
WHERE { ?X @firstName "James" }
## CYPHER
MATCH (person:Person)
WHERE person.firstName="
```

Figura 2.3 Esempio di interrogazione con vari linguaggi: verranno restituite le persone che si chiamano James.

Un ulteriore linguaggio utilizzato nelle interrogazioni di basi di dato a grafo è **Cypher** [5]: si basa sullo sfruttamento di query precompilate e memorizzate nella cache della base di dati. Tali query possono contenere attributi quali numeri, stringhe o array.
Il risultato delle interrogazioni può essere sotto forma di nodi completi di attributi, selezione di particolari attributi o dati aggregati, ovvero valori ricavabili dall'applicazione di operazioni (media, massimo, minimo) ad un range di dati.

I linguaggi suddetti vengono utilizzati dall'utente per estrarre dalla base di dati le più svariate informazioni.
Tali dati vengono, quindi, estratti mediante interrogazioni.
Queste possono essere di quattro tipi [7]:

- **Interrogazioni per il riscontro di una condizione:** sono interrogazioni pattern matching: focalizzano la loro ricerca su dati che soddisfano proprietà richieste dall'utente.

- **Interrogazioni di adiacenza:** sono delle interrogazioni che sfruttano la proprietà di adiacenza dei nodi di cui è formato il database. Queste possono verificare l'adiacenza fra due nodi o restituire il numero e il contenuto dei nodi adiacenti rispetto ad un nodo k-esimo. Vengono utilizzate spesso nei database contenenti dati inerenti alla chimica o alla biologia.

- **Interrogazioni di raggiungibilità:** queste query si occupano della raggiungibilità dei nodi all'interno del database. Esse restituiscono valori booleani o un elenco di vertici indicante i percorsi possibili fra due nodi.

- **Interrogazione analitica del grafo:** interrogazioni eseguite mediante una serie di algoritmi di analisi che restituiscono le caratteristiche del grafo.

2.4 Sistemi di Gestione

Per far in modo che le query vadano a buon fine e che l'intero sistemi funzioni in modo adeguato è necessario che la base di dati sia gestita in modo univoco da un insieme di software, ovvero un sistema in grado di operare e controllare l'intero database: il DBMS.
Il **DBMS-*Database Management System*** è un sistema di programmi avente le seguenti funzioni [1]:

- **Memorizzazione e reperimento dati:** in pratica tale sistema è in grado di tradurre le richieste di dati pervenute da una applicazione e "prepara" il sistema operativo, ovvero l'interfaccia attraverso cui avviene l'interazione con l'utente, al recupero dei dati.

- **Controllo sicurezza e integrità database:** ulteriore funzione molto importante è quella di impedire la visita e/o la modifica del database o di una parte di esso agli utenti non autorizzati e controlla l'inserimento dei dati evitando duplicazioni o bloccandone l'accesso durante un'eventuale modifica.

Quindi tale sistema fa da mediatore tra il database e l'utente. In **figura 2.4** è raffigurato un classico uso del sistema DBMS: i vari soggetti presenti all'interno dell'impresa per svariati motivi hanno bisogno di accedere, anche in contemporanea, al database aziendale. Sarà compito, quindi, del DBMS gestire gli accessi in maniera ottimale alla base di dati in questione.
Generalmente per capire come è strutturato un DBMS si può far riferimento all'insieme di tali sezioni:

- **Gestione delle interrogazioni**: si occupa delle richieste pervenute dagli utenti, ovvero della traduzione di un linguaggio dichiarativo in un insieme di procedure da

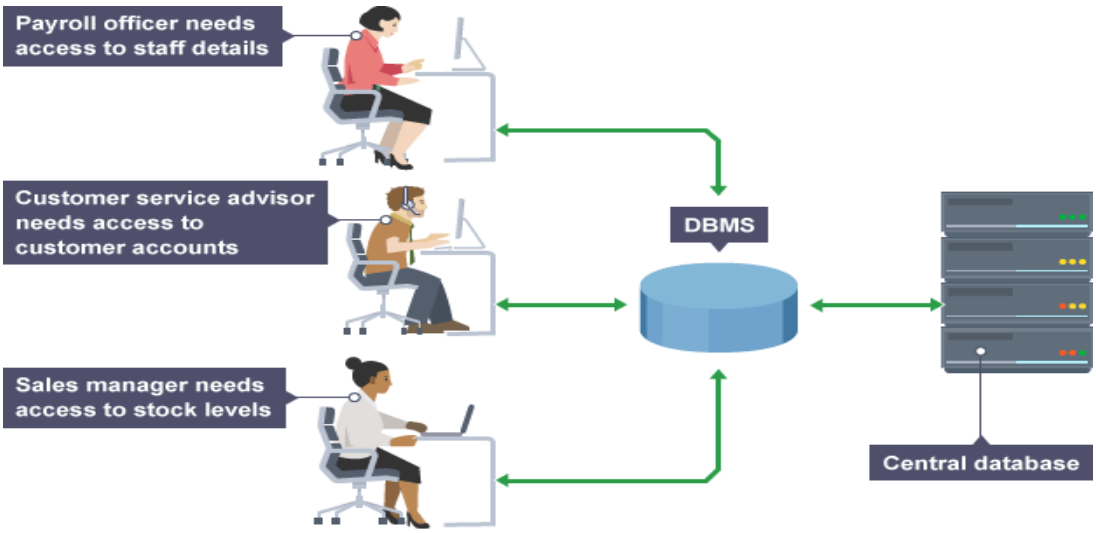


Figura 2. 4 esempio di gestione di interazioni fra utenti e database all'interno dell'azienda mediante il DBMS.

eseguire. Compito di tale sezione è inoltre ottimizzare il tempo di tali procedure attraverso la scelta di più alternative.

- **Gestione dei metodi di accesso**: sezione che si occupa di individuare il blocco entro cui vi si trovano i dati richiesti dall'utente. Inoltre viene controllato l'accesso a tali blocchi in modo da non permettere la modifica contemporanea dei dati.

- **Gestione dei buffer**: solitamente, nel corso delle normali operazioni sui dati, la memoria richiesta per i blocchi è maggiore di quella realmente disponibile. Nasce, quindi, la necessità di occupare un'area di memoria "d'appoggio" in cui poter elaborare i vari blocchi. Tale sezione si occupa delle operazioni eseguite in questa area.

- **Gestione dei guasti**: è la sezione che garantisce il normale funzionamento dell'intero sistema e che consente di ricostruirne lo stato qualora si verificassero particolari guasti al sistema stesso o ad un dispositivo di memorizzazione.

Relativamente ai sistemi di gestione (Graph-DBMS) delle basi di dati a grafo possono essere individuate due macro-classi [1]:

- Basi di dato a grafo
- Framework elaboratori di grafi

In particolare i primi applicano i concetti suddetti a database strutturati a grafo e si pongono l'obiettivo di dedicare un'area di memoria in maniera persistente ai dati in modo da facilitare anche le operazioni di recupero degli stessi.
I secondi invece pongono la loro attenzione su vari processi di raggruppamento e di analisi di grafi di grandi dimensioni.

2.5 Le Transazioni

Una transazione risulta essere una serie di operazioni eseguite in sequenza che permettono al database di passare da uno stato corretto ad un altro stato corretto.
Tale serie di operazioni può concludersi con un successo o un insuccesso. Nel caso di successo di tutte le operazioni il risultato scaturito da queste deve essere inserito, con un'operazione definita commit work, in maniera permanente all'interno del database, mentre nel caso in cui una o più operazioni non riuscissero a terminare in maniera positiva si verificherebbe lo stato di abort, ovvero il sistema verrà riportato allo stato precedente alla transazione attraverso un'operazione di rollback.
Una transazione per essere definita tale deve godere delle proprietà **ACID [1]**:

- **Atomicità**: le operazioni previste dalle transazioni devono risultare indivisibili, ovvero vi è il bisogno che vengano eseguite nella loro interezza. Grazie a questa proprietà non è possibile eseguire solamente una parte della transazione, garantendo più affidabilità.

- **Consistenza**: la transazione in atto deve garantire consistenza a tutti i dati presenti nel database, ovvero non possono essere inseriti o modificati dati che non rispettano vincoli di integrità. Se lo stato iniziale del database risulta essere corretto, deve risultare tale anche lo stato successivo alla transazione.
- **Isolamento:** questa proprietà permette alle transazioni di essere eseguite in modo indipendente l'una dall'altra. Cosi facendo se una

transazione dovesse fallire non influenzerebbe il buon fine di un'altra transazione, verrebbe evitato l'effetto domino.

- **Durabilità:** tale proprietà garantisce che tutte le modifiche apportate da transazioni andate a buon fine, vengano apportate al database in maniera permanente. Per assicurarsi da questo punto di vista e quindi evitare la perdita dei dati, vengono tenute dei registri (log) ove si registrano tutte le operazioni sul database.

Generalmente, all'interno del database, le transazioni seguono questi step:

a) **Inizio transazione**: vengono individuate le aree di memoria entro cui vi si trova l'informazione su cui svolgere le operazioni.

b) **Esecuzione operazioni:** i dati presenti nelle aree selezioni vengono elaborati e quindi modificati. Durante tali elaborazioni, i dati vengono bloccati (mediante delle procedure chiamate **lock**) in modo da non permettere ad altre transazioni la modifica.

c) **Verifica del successo o insuccesso della transazione**: dopo aver eseguito l'elaborazione dei dati, il sistema di controllo verificherà l'esito della transazione. In particolare se quest'ultima risulta essere idonea verrà eseguito il commit work, ovvero le modifiche verranno apportate al database e rese permanenti.
   Invece se la transazione non dovesse ottenere il successo, si attuerebbe il rollback work, cioè il sistema verrebbe ripristinato allo stato precedente alla transazione.

d) **Fine della transazione**

È importante che in entrambi i casi, successo o insuccesso, la transazione venga portata a termine, poiché fino a quando la transazione risulta essere attiva non verranno rilasciati gli oggetti del database che servono alla transazione e non verrà liberata l'area di memoria utilizzata dal sistema di traduzione del linguaggio query in input operativi, ovvero in linguaggio.
Di notevole importanza nelle transazioni è il compito del *transaction log:* processo atto ad annotare su un giornale (chiamato log) tutte le operazioni, ovvero le modifiche, eseguite dalle transazioni sul database. Tale processo è fondamentale poiché garantisce la proprietà di persistenza dei dati e, inoltre, risulta utile nella fase di commit.

2.6 Alcune basi di dati

Oggi una parte del lavoro dei maggiori istituti di ricerca informatica si focalizza su configurazioni di basi di dati alternative a quelle esistenti.
Tale ricerca di nuove configurazioni è dettata dall'esigenza di avere database sempre più efficienti, in termini di tempo di prelievo delle informazioni, e più semplici da gestire.
A tal proposito nel prossimo capitolo, entrando nel cuore dell'elaborato, verrà descritto un progetto di database innovativo fondato su un diverso paradigma di interrogazione: il sistema **GraphVista**.
Per capire meglio le innovazioni apportate da tale sistema è utile conoscere le caratteristiche dei database oggi in uso.
In particolare è necessario focalizzarsi sui limiti e individuare in che modo questi siano stati superati o si provi a farlo attraverso la ricerca.
In questa sezione, quindi, verranno descritte alcune basi di dati a grafo oggi utilizzate.

2.6.1 Neo4j

Uno dei primi database che verranno descritti è **Neo4j** [1].
È disponibile dal 2007 ed è stato sviluppato dalla *Neo Technology.* È un database a grafo, scritto utilizzando il linguaggio java, il cui codice è stato reso pubblico fin dalla prima versione in modo che qualsiasi programmatore potesse favorirne lo sviluppo.
Tale condizione e la possibilità di far uso del web come strumento di condivisione hanno permesso in poco tempo al sistema di integrarsi al meglio con diversi sistemi operativi (Linux, Windows, Macintosh).

Alla base del reperimento e della modifica dei dati del database Neo4j vi è il meccanismo delle transazioni descritto precedentemente.
In aggiunta al meccanismo di transazione classico, Neo4j permette l'esecuzione di transazioni annidate.
Ogni transazione quindi presenta una serie di sotto-transazioni. In questo caso qualora una solamente delle sotto-transazioni non dovesse andare a buon fine, verrebbe eseguito l'abort dell'intera transazione madre.

Questo sistema può essere utilizzato in due modalità [7]:

- **Modalità Embedded**: In questa modalità il database viene caricato direttamente all'interno dell'applicazione di cui si sta usufruendo, in pratica il database viene integrato all'interno dell'applicazione e ne vengono sfruttate le funzionalità come se facessero parte dell'applicazione in cui è contenuto.
  L'incorporazione del database nelle applicazioni può avvenire attraverso particolari software finalizzati al prelievo dei file JAR (java Archive), librerie contenenti funzionalità utili alla rappresentazione e alla manipolazione di strutture dati.
  Neo4j Embedded, essendo parte integrante dell'applicazione, risulta avere tempi di risposta brevissimi. Inoltre, permette di eseguire transazioni più complesse.
  Un esempio di applicazione di tale modalità è riportato in **figura 2.5**, ove è possibile notare il caricamento, attraverso la parola chiave "import", di alcune librerie di un database Neo4j all'interno dell'applicativo ECLIPSE IDE.

```
package com.neo4j.chapter1;

importorg.neo4j.graphdb.GraphDatabaseService;
import org.neo4j.graphdb.Node;
import org.neo4j.graphdb.Direction;
import org.neo4j.graphdb.Relationship;
import org.neo4j.graphdb.Transaction;
import org.neo4j.graphdb.RelationshipType;
import org.neo4j.graphdb.factory.GraphDatabaseFactory;
```

Figura 2.5 parte introduttiva di un progetto sviluppato su Eclipse in cui è possibile notare l'importazione, in modalità Embedded, di un database neo4j.

- **Modalità Server**: In questa modalità il database viene utilizzato come se fosse un'applicazione a sé stante.
  Al database si accede attraverso la formulazione di interrogazioni delle quali si riceveranno i risultati in remoto. Tale modalità permette a più applicazioni (client) di utilizzare il database simultaneamente.
  I vari client formulano le richieste al server in formato JSON (formato, basato sul linguaggio JavaScript, adatto allo scambio di dati fra applicazioni Client-Server) utilizzando il protocollo HTTP.
  Tale caratteristica permette alle varie applicazioni client di essere costruite su diversi tipi di piattaforma a patto che queste posseggano librerie HTTP.
  Neo4j Server è la modalità più comunemente utilizzata.

Una caratteristica che contraddistingue Neo4j è la possibilità di utilizzare indici al fine di reperire più velocemente i dati.
Ciò è possibile grazie ad un servizio di indicizzazione che sfrutta un insieme di procedure (chiamate Lucene) che permettono di memorizzare dei nodi, quindi delle aree di memoria, assegnandovi "etichette" aventi specifici valori.
Questo stratagemma permette di velocizzare l'accesso ai nodi. Infatti conoscendo il valore dell'etichetta sarà possibile ridurre al massimo il tempo di ricerca dell'informazione desiderata.
Tale funzione può essere automatizzata dal sistema assegnando un valore indice casuale ogni qualvolta viene inserito un nuovo nodo o modificato uno già esistente.
Per la manipolazione e l'accesso ai dati inseriti possono essere sfruttate le funzioni implementate in Neo4j. Solitamente le applicazioni utilizzano diversi linguaggi di interrogazione, alcuni di essi risultano essere direttamente supportati da Neo4j altri non lo sono.
Per tale motivo nasce il bisogno di tradurre tali linguaggi in un codice standard, univoco.
Per assecondare tale esigenza il sistema è dotato di più software di compilazione che hanno il compito di trasformare i vari linguaggi di interrogazione in bytecode. Il bytecode viene letto ed interpretato dalla JVM (java virtual machine). La JVM è fondamentale in quanto è in grado di tradurre il bytecode in linguaggio macchina.
Successivamente il linguaggio macchina viene fornito alla macchina fisica e da essa viene eseguito.

Neo4j risultata essere perfettamente compatibile con sistemi aventi strutture ad albero, ovvero sistemi descrivibili attraverso grafi come file XML e reti, in quanto può sfruttare la sua normale predisposizione e rendere più veloce l'accesso ai dati.
Inoltre esso mette a disposizione dell'utente varie implementazioni riguardanti operazioni comuni (cammino minimo tra nodi, ricerca cicli etc.) in modo da facilitarne l'utilizzo e renderlo più efficiente.

Vanno però sottolineati i limiti di tale sistema:

- Neo4j è assai sconveniente nei casi in cui vengano ricercate informazioni richiedenti confronti fra dati presenti su nodi diversi e quindi risulta difficile anche la modifica avente per oggetto tali dati.

- Il sistema risulta non idoneo alla memorizzazione di dati in formato audio o video, per i quali risulta necessario l'utilizzo di un ulteriore database.

- Impossibilità di partizionare i dati che rende il sistema non adatto per i lavori di ETL (*estrazione, trasformazione, caricamento)* cioè risultano poco eseguibili operazioni di consolidamento e raggruppamento dei dati più significativi.

2.6.2 OQGRAPH (Open Query GRAPH)

Questa sistema viene descritto come libreria per determinati database relazionali.
In realtà la mansione di OQGRAPH è quella di trasformare la navigazione tabellare di tali database in una navigazione ad albero e quindi mediante grafi.
Per far ciò è necessario che le tabelle utilizzate siano organizzate tutte allo stesso modo. È quindi necessario che vi sia uno standard da seguire quando esse vengono create.
La **figura 2.6** riporta lo standard sintattico che viene seguito durante la loro creazione.
Vengono associati particolari significati alle colonne delle tabelle, in particolare [1]:

- **origid**: vertice di origine
- **destid**: vertice di destinazione
- **weight**: valore assegnato all'arco che collega i due nodi
- **latch**: specifica delle informazioni desiderate
- **linkid**: sono indicati gli identificativi dei vertici che collegano due nodi

I dati presenti nelle tabelle vengono assegnati ai vari nodi o agli archi a seconda della colonna in cui sono contenuti.
I nodi vengono identificati da numeri progressivi in modo da facilitarvi l'accesso.

Lettura e modifica dati:

La lettura avviene attraverso delle query. In esse è importante che vi sia indicato la specifica delle informazioni da ricercare, ovvero il contenuto della colonna latch. Inoltre è possibile indicare delle proprietà su cui basare la ricerca all'interno dei grafi, cioè può essere indicato il progressivo del nodo di origine e/o del nodo di destinazione in modo da scremare i nodi entro cui ricercare l'informazione.

```
CREATE TABLE db.tblname (
     latch    SMALLINT   UNSIGNED NULL,
     origid   BIGINT     UNSIGNED NULL,
     destid   BIGINT     UNSIGNED NULL,
     weight   DOUBLE     NULL,
     seq      BIGINT     UNSIGNED NULL,
     linkid   BIGINT     UNSIGNED NULL,
     KEY (latch, origid, destid)
USING HASH,
     KEY (latch, destid, origid)
USING HASH
   ) ENGINE=OQGRAPH;
```

Figura 2. 6 creazione di una tabella mediante linguaggio SQL.È possibile osservare l'imprescindibilità della tabella dalle colonne latch, origid, destid e weight.

I risultati della ricerca verranno restituiti nelle colonne weight e linkid, in particolare nella prima verranno restituiti i valori dei pesi (proprietà) presenti negli archi e nella colonna linkid verranno restituiti i progressivi dei nodi che costituiscono il percorso necessario per leggere l'informazione desiderata.
La modifica risulta essere molto più diretta: avviene attraverso query SQL in cui sono specificate i dati da modificare o aggiungere nelle clausole INSERT, UPDATE.
È importante precisare che possono essere modificati solo i dati presenti nelle colonne origid, destid e weight.

Tale sistema rende più efficiente la ricerca entro database relazionali sfruttandone la navigazione ad albero. La sua diffusione, però, risulta essere molto limitata poiché è implementabile soltanto nei database MySQL
Inoltre è molto oneroso, in termini di memoria, in quanto richiede un'ingente quantità di tabelle di supporto entro cui registrare e modificare i dati.

2.6.3 DEX

È un sistema implementato in base di dati a grafo scritto con il linguaggio C++.
Si basa su un modello di database sviluppato inseguendo tre caratteristiche principali [1]:

- Dati organizzati mediante grafi o strutture simili
- Modifica dei dati basata su operazioni graph-oriented
- Presenza di vincoli specifici che garantiscano le integrità dei dati e le interazioni fra essi.

Questo sistema viene definito diretto, etichettato, "Attributed" e multigrafo: diretto poiché fa uso di grafi diretti, etichettato perché sia i nodi che gli archi vengono contraddistinti da una certa classe (definita da un'etichetta appunto), può presentare attributi sia nei nodi che negli archi, mentre viene detto anche multigrafo a causa della possibilità di avere più archi che collegano gli stessi nodi.
La caratteristica cardine di questi sistemi è la facilità con cui si possono immagazzinare e recuperare dati in grafi che presentano un numero notevole di nodi.
Anche questo sistema però presenta vari limiti come ad esempio la ridotta interazione con l'utente

2.7 Conclusioni

Le basi di dati suddette oggi risultano obsolete o meglio vi è la tendenza e la necessità di implementare database più efficienti.
Dalla descrizione di queste basi di dati sono emersi i seguenti limiti:

- Difficile interazione utente-database
- Difficoltà nella gestione di grafi estesi
- Richiesta elevata di memoria

- Incompatibilità con alcuni applicativi

- Dipendenza da un linguaggio specifico di interrogazione

Inoltre la quantità dei dati da gestire risulta essere sempre più ampia e vi è l'esigenza di un sistema di memorizzazione sempre più veloce.
Oltre alla velocità, è richiesta una maggiore interazione dell'utente con la base di dati stessa.
La ricerca informatica si sta muovendo in questa direzione studiando e sviluppando nuovi progetti di database.
Uno di questi progetti, come detto in precedenza, è Graph Vista e verrà discusso nel capitolo successivo.

Capitolo III

Il sistema GraphVista

In questo capitolo verranno illustrate le caratteristiche di questo nuovo sistema di interrogazione dei database.
Verranno descritte le principali funzioni e le migliorie più significative rispetto ai tradizionali sistemi.

3.1 Paradigmi tradizionali

L'analisi di un database viene eseguita mediante l'uso di interrogazioni i cui risultati vengono visualizzati su console.
Viene quindi utilizzato il paradigma "query ad-hoc", ovvero vengono eseguite delle interrogazioni specifiche, finalizzate all'ottenimento delle informazioni richieste.
Tale paradigma col passare del tempo sta risultando sempre più obsoleto in quanto presenta vari limiti:

- Richiede più iterazioni: il risultato non è detto che venga ottenuto alla prima interrogazione.
- Durante l'esecuzione della query non è possibile interagire col sistema.
- È necessario conoscere un linguaggio di interrogazione specifico, a volte troppo complesso.

Tali limiti possono essere ovviati applicando determinati stratagemmi.
Per esempio, al fine di rendere più semplice l'esplorazione del database, si potrebbe lavorare sulla quantità dei dati presenti nel database applicandovi algoritmi (Clustering) utili alla scrematura degli stessi.
Tale approccio però faciliterebbe si l'esplorazione dei vertici all'interno della base di dati ma non apporterebbe nessuna miglioria nella formulazione delle query e quindi nell'interazione con l'utente.
Vi è il bisogno, quindi, di intervenire sul linguaggio di interrogazione.

Oggi gli istituti di ricerca hanno fatto notevoli passi avanti a riguardo, offrendo un sistema di interrogazione visiva basato sull'utilizzo di sottografi [8].
Uno di questi sistemi è QUBLE. Esso sfrutta i sottografi per la formulazione di query ma non permette l'esplorazione interattiva del database e soprattutto risulta essere assai inefficace per database aventi un elevato numero di grafi.
Molti database, fra cui quelli citati nel capitolo precedente, pur ovviando ai limiti suddetti implementando sistemi di interrogazione più efficienti, risultano essere parzialmente efficaci. Essi difficilmente permettono all'utente di partecipare attivamente all'esplorazione del database stesso.
Ci si pone, quindi, la sfida di offrire all'utente un paradigma di interrogazione basato sulla combinazione di query specifiche (ad-hoc) ed esplorazioni grafiche.
In particolare tale combinazione dovrebbe essere cosi strutturata:

- Sunto generale della query e individuazione di particolari "punti di ingresso" nel sistema.

- Esplorazione interattiva del grafo a partire dai punti di ingresso sopra citati.

La prima parte rappresenta la query ad-hoc. Viene chiesta all'utente la formulazione di una query, detta **query driver**.
Viene chiamata in questo modo poiché rappresenta un input da cui poter iniziare l'interrogazione del database.
Attraverso la query driver è possibile avvicinarsi al risultato finale. In essa verranno specificati quindi alcuni attributi che i risultati dovranno necessariamente avere, cosi facendo verrebbero automaticamente esclusi dalla successiva fase tutti quei dati non necessari.
Questa fase infatti non restituirà il risultato finale ma verrà interrotta ogni qualvolta si verifica una corrispondenza tra la clausola di ricerca immessa nella query e i dati presenti nei nodi del database. Inoltre, tali punti di interruzione potrebbero essere salvati ed utilizzati come punti di ingresso nel database per le successive interrogazioni.
Per esempio, se si volessero conoscere i dati degli studenti di Ingegneria che hanno sostenuto l'esame di informatica, una possibile query driver con la quale iniziare l'esplorazione del database sarebbe quella che va a selezionare tutti gli studenti di ingegneria o ancor meglio gli studenti di ingegneria con informatica nella lista degli esami sostenuti

Ottenuti i risultati della query ad-hoc, mediante esplorazione interattiva, si potrebbero estrapolare dal database i dati richiesti.
La secondo parte, quindi, porterà a termine l'interrogazione a partire dai punti di ingresso precedentemente definiti.
Il risultato ottimale della sfida suddetta, ad oggi, sembra essere il sistema ***GraphVista*** **[8].**
Tale sistema per ricercare i dati all'interno del database, seguendo le direttive inserite nella query ad-hoc, si affida ad un elaboratore grafico chiamato ***GRAPHITE***.

3.2 GRAPHITE

Gli utenti moderni chiedono maggiore flessibilità e facilità nell'operazione di ricerca e interrogazione di un database.
Oggi ascoltando i malumori degli utenti, in particolare quelli che interagiscono con delle basi di dati a grafo, emergono le seguenti richieste:

- Necessità di sviluppare un modo più "comodo" con cui eseguire delle query.
- Maggiore velocità nei processi di matching.
- Bisogno di trovare maggiori corrispondenze (totali o parziali) all'interno di un database col minor "sforzo".

Al fine di risolvere queste e altre problematiche è stato progettato GRAPHITE[9]: un sistema di ricerca ed elaborazione dei grafi basato sulla costruzione grafica di modelli di query.
In particolare l'utente può disegnare la struttura del modello (sottografo) da ricercare nel database e, attraverso un'analisi comparativa, ottenere le corrispondenze desiderate in pochi secondi.
Tale sistema, grazie alla possibilità di interrogazione visiva che offre, viene utilizzato per l'esecuzione della query driver all'interno di GraphVista.
GRAPHITE permette di interrogare database aventi migliaia di nodi ed estrarre da essi solo la parte di interesse semplicemente attraverso un modello di partenza, ma cosa si intende per modello?
Il modello è un sottografo avente le caratteristiche dell'informazione che si vuole ricercare. Il sistema quindi non farà altro che individuare le corrispondenze, totali o solamente parziali, tra il modello proposto e il database.

Per esempio dato un grafo di dati come quello in **figura 3.1 (a)** in cui:

- Nodi: rappresentano persone aventi ruoli differenti indicati mediante attributi (colori e forme diverse) all'interno di un'azienda.

- Archi: rappresentano le interazioni fra le varie persone all'interno del contesto aziendale.

Un eventuale modello, ovvero una configurazione di nodi e archi da trovare all'interno del database, inserito dall'utente potrebbe essere quello riportato in **figura 3.1 (b)**.
Prima di cominciare l'analisi viene chiesto all'utente il numero (k) di corrispondenze da trovare all'interno del database da esaminare.

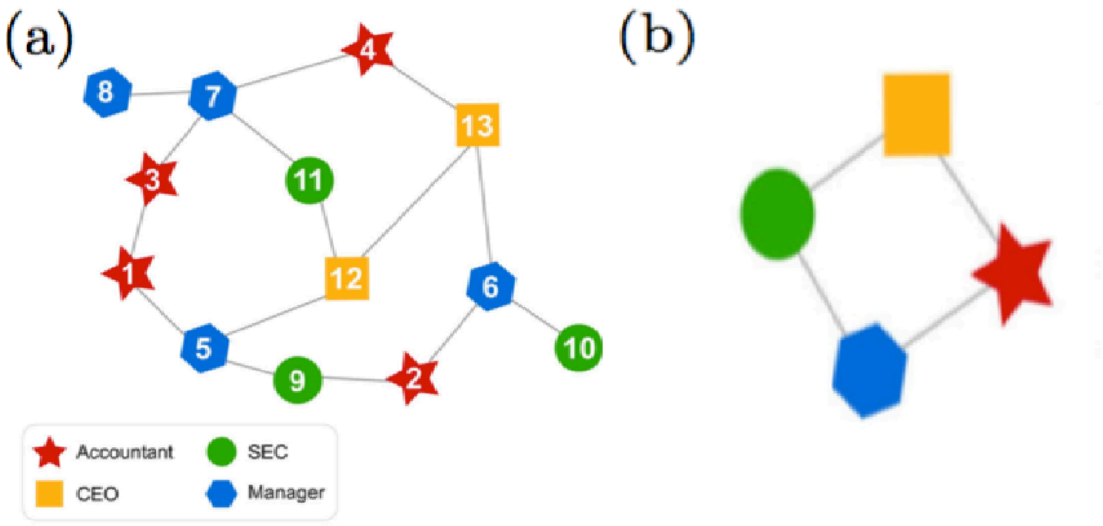


Figura 3.1 Rete aziendale descritta mediante un grafo (a) e relativo sottografo modello (b) da inserire in GRAPHITE

3.2.1 Attraversamento Database

Una volta inseriti tutti gli input GRAPHITE potrà quindi elaborare la richiesta e restituire i risultati richiesti, ma come avviene ciò?
Per rendere più veloce l'esplorazione del database e restituire in fretta i risultati il sistema utilizza un algoritmo chiamato G-Ray.
Per permettere l'applicazione di tale algoritmo vengono mappate le proprietà del grafo da analizzare, ovvero vengono costruite due tabelle: la prima conterrà le informazioni relative ai vertici, mentre nella seconda verranno registrate tutte le connessioni fra i vertici e gli eventuali attributi relativi a tali connessioni.
Inoltre, ad ogni vertice viene assegnato un identificatore col quale verrà distinto all'interno del sistema.
Mediante l'uso di queste tabelle è possibile attraversare i grafi ed effettuare una veloce analisi comparativa.
Un attraversamento$_{[10]}$ f è definito come una tupla f=(S,ø,c,r,d) in cui:

- S rappresenta un insieme contenente i vertici da cui partirà l'attraversamento.
- Ø rappresenta uno o più predicati da verificare sugli archi.
- c rappresenta il numero minimo di archi verificabili.
- r rappresenta il numero massimo di archi verificabili.
- d descrive la direzione di attraversamento.

Per ogni arco, l'algoritmo verificherà il predicato ø e nel caso in cui dovesse esservi corrispondenza inserirà i relativi nodi nel risultato.

Il sistema GRAPHITE riceve una tupla f ed elabora un attraversamento. Solitamente questo è composto da tre fasi:

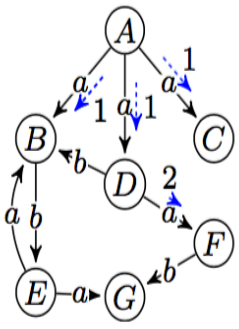
1) **Fase di preparazione**: vengono sostanzialmente preparati i dati da analizzare, in particolare viene codificato e segnalato il cammino da seguire a partire dai nodi contenuti nell'insieme S. Per la codifica si utilizzano le tabelle contenenti i nodi descritte in precedenza. Ad ogni nodo viene assegnato un identificatore interno, diverso da quello presente in tabella, di cui si tiene traccia lungo tutte le operazioni future.

2) **Fase di attraversamento:** in questa fase, a partire dai nodi segnalati nella fase precedente, vengono verificati i predicati ø sui vari archi e rispettando i limiti dei livelli di percorrenza si procede con l'attraversamento del grafo.

3) **Fase di decodifica:** vengono tradotte le rappresentazioni risultate dall'attraversamento, ovvero mediante gli identificatori interni si risale alla tipologia del nodo di partenza e se ne restituisce l'ID iniziale.

Per esempio se si volesse eseguire un attraversamento del grafo in **figura 3.2**, data una tupla *f*= ({A}, 'type = a', 1, 1, →) si otterrebbero come risultato i nodi B, C, D, in quanto:

- tutti e tre i nodi risultato essere collegati al nodo A con un collegamento di tipo a, ovvero tutti e tre risultato verificare il predicato ø.
- Tutti e tre i nodi risultano essere entro i limiti di archi verificabili dettati dalla tupla.
- Tutti e tre i nodi sono attraversabili secondo la direzione
- →, ovvero uscente dal nodo A.



Traversal Configuration	Result
$(\{ A \}, \text{'}type{=}a\text{'}, 0, 1, \rightarrow)$	$\{ A, B, C, D \}$
$(\{ A \}, \text{'}type{=}a\text{'}, 0, 1, \rightarrow)$	$\{ A, B, C, D \}$
$(\{ A \}, \text{'}type{=}a\text{'}, 1, 1, \rightarrow)$	$\{ B, C, D \}$
$(\{ A \}, \text{'}type{=}a\text{'}, 2, 2, \rightarrow)$	$\{ F \}$
$(\{ A \}, \text{'}type{=}a\text{'}, 1, \infty, \rightarrow)$	$\{ B, C, D, F \}$
$(\{ E \}, \text{'}type{=}b\text{'}, 2, 2, \leftarrow)$	$\{ D \}$
$(\{ A \}, \text{'}type{=}a \vee type{=}b\text{'}, 2, 2, \rightarrow)$	$\{ E, F \}$

Figura 3.2 grafo di esempio e tabella di attraversamento in cui sono indicate le tuple e i rispettivi risultati

Se avessimo avuto una tupla del tipo f=({B}, 'type = a', 1, 1, →) l'insieme dei nodi risultato sarebbe stato vuoto in quanto non vengono rispettati i vincoli imposti dalla tupla.
Il sistema, quindi, è in grado di immagazzinare tutte le possibili configurazioni di attraversamento di un database e ogni qualvolta arrivi un modello di grafo, questo verrà convertito in una tupla f che sarà eseguita e in pochi secondi verranno restituiti tutti gli insiemi che verificano la tupla in oggetto.
Come abbiamo visto, però, il sistema non restituirà una tupla ma la trasformerà nuovamente in un grafo da cui poi l'utente potrà continuare l'esplorazione sulla piattaforma GRAPHVISTA.

L'algoritmo utilizzato per l'attraversamento viene definito intelligente poiché:

- Quando non si verificano corrispondenze esatte, vengono restituite automaticamente quelle "best-effort", ovvero si prendono in considerazione percorsi indiretti e più lunghi.

- restituisce all'utente le k corrispondenze richieste ordinate in base alla bontà della corrispondenza stessa, ovvero l'output verrà classificato in ordine decrescente a partire dalla corrispondenza più vicina alle richieste dell'utente.

3.2.2 Interfaccia utente

Come è possibile vedere dalla **figura 3.3,** l'interfacciamento con l'utente nel sistema GRAPHITE è abbastanza semplice.
In particolare la pagina di lavoro viene divisa in due macro sezioni [10]:

- Area delle query (a): area dedicata all'utente per permettergli di disegnare i modelli-grafo che Graphite dovrà analizzare.

- Area dei risultati (b): area di sola lettura in cui il sistema restituirà all'utente i risultati.

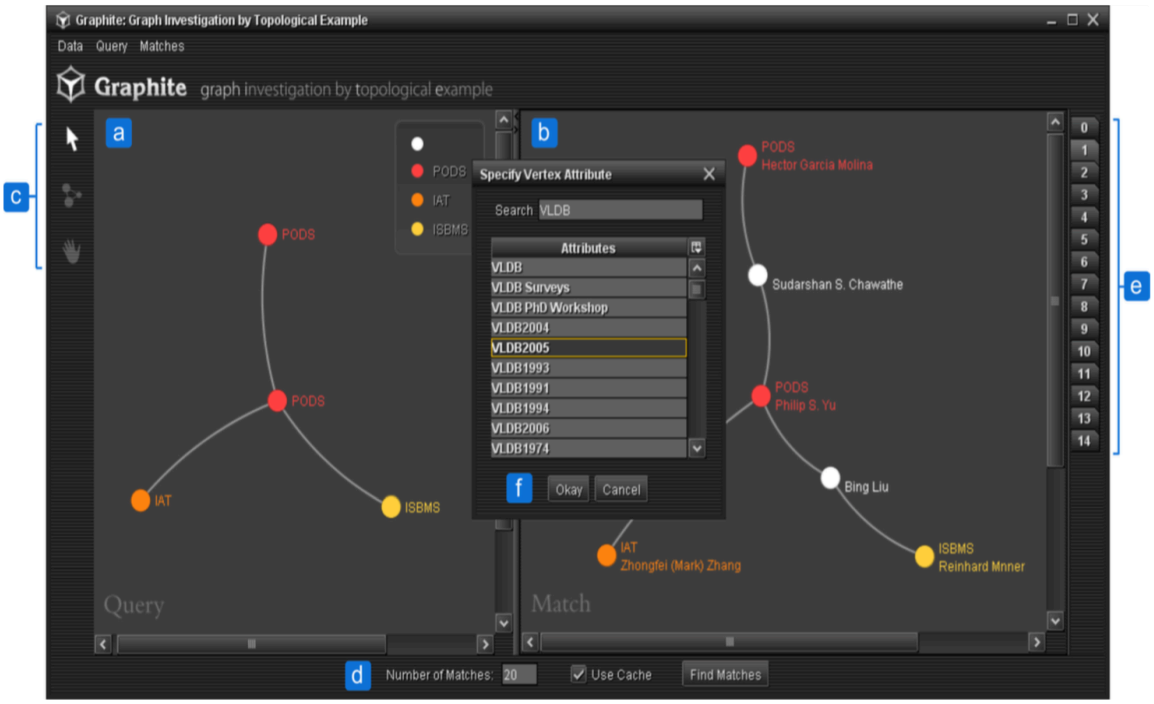


Figura 3.3 Interfaccia utente in GRAPHITE

La pagina di lavoro risulterà quindi suddivisa verticalmente, in cui: la parte di sinistra ospita l'area delle query, mentre la parte di destra è occupata dall'area
dei risultati.

L'utente, nell'area delle query, può inserire archi e nodi, modificarli o riposizionarli mediante strumenti situati in alto a sinistra della pagina (c). Può inoltre assegnare attributi/valori ai nodi o agli archi facendo doppio clic su di essi e scegliendo un valore da una finestra che comparirà appositamente (f).
Infine, all'utente viene chiesto di inserire il numero di corrispondenze che si vogliono trovare in una cella in fondo alla pagina (d).
Per quanto riguarda i risultati, quindi la parte destra della pagina, verranno visualizzati (sotto forma di grafi) e classificati in base alla bontà delle corrispondenze.
La classifica delle corrispondenze verrà riportata in alto a destra nell'area dei risultati (e).

3.3 GraphVista

È un paradigma di database a grafo basato sull'interrogazione visiva dei grafi. Come detto precedentemente, sfrutta la combinazione di query ad-hoc ed esplorative.
In particolare l'interrogazione di un database è suddivisa in due fasi.
Nella prima fase ci si avvicina al risultato mediante query specifiche (query driver), mentre nella seconda fase è richiesta la stretta interazione dell'utente in quanto esso attraverso la manipolazione visiva può estrapolare dal database il dato ricercato.
L'obiettivo di tale paradigma è quello di consentire all'utente l'analisi di grandi database a grafo senza sovraccaricare eccessivamente l'intero sistema.
Per iniziare l'esplorazione del database e quindi dei grafi, l'utente dovrà necessariamente formulare una query (query driver) sfruttando i vari comandi di cui è dotato GraphVista, in particolare viene scelto il tipo di query da eseguire e i predicati che devono essere verificati (breakpoint). Dopo aver lanciato la query driver il sistema restituirà all'utente, su un'ulteriore pagina, il grafo risultato. L'utente potrà quindi svolgere diverse operazioni (lettura attributi, esplorazione nodi/archi) su tale grafo ed attraverso di esse può arrivare al risultato desiderato.

Dopo questa breve descrizione è possibile individuare i primi vantaggi portati dall'utilizzo di questo modello:

- Non è richiesta all'utente la conoscenza di un linguaggio di interrogazione avanzato.

- È possibile ottenere risultati intermedi, memorizzarli ed utilizzarli in seguito.

- Ogni qualvolta un predicato di ricerca viene verificato, ovvero viene attivato un breakpoint, il sistema salva automaticamente tale punto come "punto di ingresso" nel database. Nelle successive interrogazioni sarà possibile sfruttare tali punti di ingresso.

- Le interrogazioni ad-hoc permettono di evitare l'analisi di parti del grafo non rilevanti ai fini del risultato.

3.3.1 Architettura

L'architettura del sistema GraphVista, illustrata in **figura 3.4**, è formata da tre componenti principali:

- **Generatore di query**: fornisce all'utente gli elementi necessari per formulare la query da cui partire ed individuare i possibili punti di interruzione o ingresso.

- **Pannello di visualizzazione grafo**: contiene degli elementi di controllo utili ad iniziare, continuare o interrompere l'esecuzione della query. Inoltre riporta un piccolo sunto della parte di grafo selezionata dalla query.

- **Archivio**: viene utilizzato per l'eventuale memorizzazione di una parte dei dati del grafo che possono risultare utili successivamente.

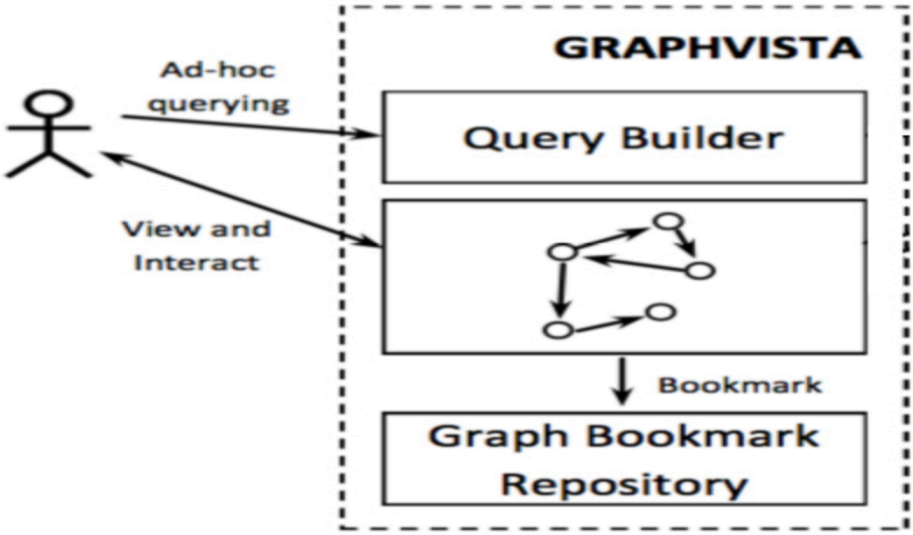


Figura 3.4 Architettura sistema GraphVista.

3.3.2 Flusso di lavoro

Dopo aver illustrato l'architettura del sistema è bene capire in che modo viene utilizzato. Il flusso di lavoro[8] che usualmente viene eseguito è illustrato in **figura 3.5**: consiste nell'inserimento, utilizzando il generatore di query, di una interrogazione di partenza (query driver) e più criteri (break-point) che se verificati memorizzano l'indirizzo del nodo. Fatto ciò viene eseguita la query. Ogni qualvolta viene attivato un break-points la query driver viene messa in pausa e a partire da quel punto viene eseguita l'esplorazione visuale interattiva da parte dell'utente.
Tale esplorazione consiste in un'analisi visiva delle informazioni situate nei nodi dei sottografi selezionati dai breakpoints al fine di estrapolare i dati ricercati. Inoltre l'utente, per svolgere al meglio l'attività di ricerca, può inserire durante l'esplorazione visiva ulteriori criteri selettivi.

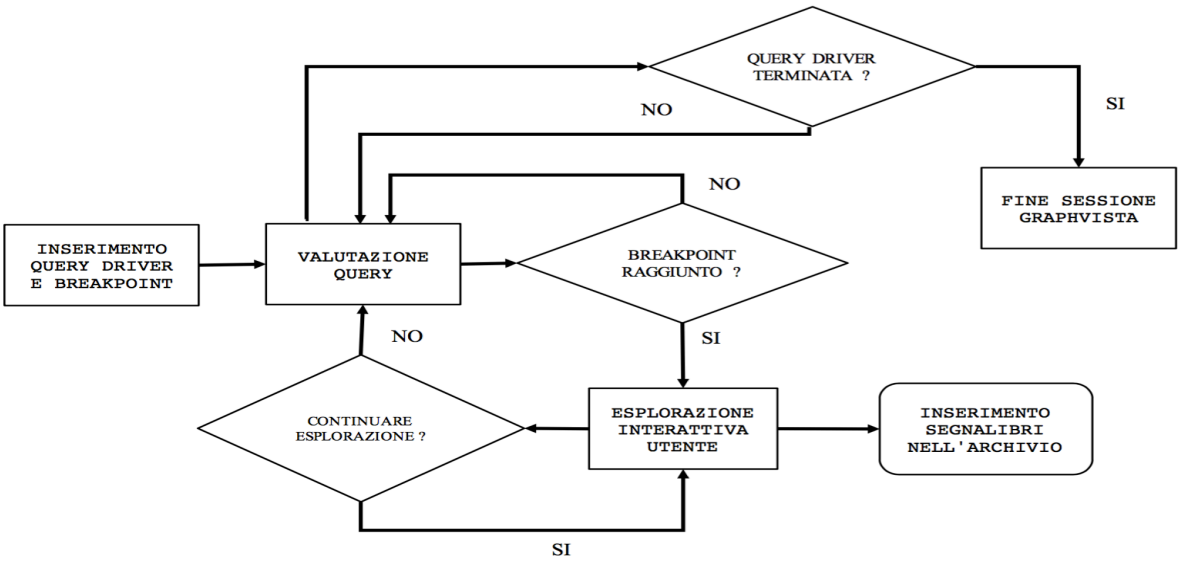


Figura 3.5 flusso di lavoro tipico con GRAPHVISTA

L'utente durante l'esplorazione, può inserire dei "segnalibri" al fine di salvare dei dati da cui poter ripartire nelle esplorazioni successive. Tali segnalibri identificano l'indirizzo del nodo all'interno del grafo e vengono salvati in un archivio interno al sistema.
Finita tale analisi l'utente può decidere se far continuare l'interrogazione del database, quindi rilasciare un'ulteriore query driver, o interromperla.

3.3.3 Esplorazione

L'esplorazione del database e quindi dei grafi, con il sistema GraphVista viene sostanzialmente divisa in due fasi:

1) Esplorazione automatica

2) Esplorazione Interattiva

La prima fase consiste nella formulazione della query driver con cui poter far partire l'esplorazione. Per tale operazione viene utilizzato un generatore di query che permette di eseguire la query driver utilizzando due diversi approcci esplorativi:

- Esplorazione di tutti i vertici e di tutti gli archi al fine di trovare una comparazione fra l'attributo di ricerca e quelli presenti nei nodi.

- Esplorazione mediante tecniche di visita in ampiezza e in profondità.

Dopo che l'utente ha scelto l'approccio di esplorazione viene chiamato in causa GRAPHITE.
L'elaboratore grafico esplorando il database esegue un'analisi comparativa e ogni qualvolta viene soddisfatto un predicato di ricerca, ovvero viene individuato un break-point precedentemente fissato, esso restituirà all'utente il nodo che ha attivato quel punto di interruzione.

Ad ogni attivazione dei breakpoints il motore di elaborazione Graphite viene messo in pausa e viene data la possibilità all'utente di scegliere il da farsi. Infatti arrivati a questo punto l'utente può scegliere di far terminare la sessione in Graphite e continuare con l'esplorazione interattiva sull'interfaccia di GraphVista, oppure far ripartire l'elaboratore grafico e aspettare l'attivazione del successivo breakpoint o la fine dell'esplorazione automatica.
Il sistema è dotato di meccanismi di "comeback", ovvero è permesso all'utente di tornare indietro ad un qualsiasi breakpoint precedentemente riscontrato. Per tale motivo viene assegnato un identificatore unico alla sessione di Graphite, la quale può gestire più sessioni GraphVista contemporaneamente (naturalmente le sessioni di GraphVista saranno inerenti tutte a operazioni su grafi appartenenti allo stesso database)

Uno dei vantaggi più importanti di GraphVista consiste proprio nell'esecuzione dell'esplorazione automatica prima di passare a quella interattiva. Cosi facendo l'utente avrà a disposizione, per l'interrogazione visiva, soltanto i dati realmente utili. L'operazione è quindi strettamente necessaria per elidere, o meglio, non coinvolgere nell'esplorazione successiva, la parte di grafo contenente i nodi e gli archi irrilevanti ai fini dell'obiettivo di ricerca dell'utente. Si procede quindi con l'esplorazione visiva e interattiva: una volta eseguita la query driver, Graphite restituisce all'utente un insieme di nodi e archi, ovvero viene restituito un sottografo del database.
L'utente, attraverso degli strumenti di elaborazione, può eseguire varie operazione sul sottografo, in particolare è possibile[9] :

- Recuperare gli archi in entrata/uscita da un certo nodo, ovvero è possibile risalire a tutti i collegamenti dei vari nodi.
- Recuperare i nodi che precedono o seguono un determinato arco.
- Recupero e visualizzazione di specifici valori di attributi relativi ad archi e/o nodi.

Per ogni operazione eseguita dall'utente, il sistema GraphVista formula automaticamente una richiesta di query a Graphite il quale la esegue e invia il risultato attraverso una connessione http al sistema GraphVista.
Arrivato il risultato, GraphVista decodificherà il messaggio e apporterà le modifiche al grafo visualizzato dall'utente. In pratica il sistema recupera il sottografo attuale e lo fonde con i nuovi dati ricevuti da Graphite.
L'esplorazione interattiva risulta essere, quindi, un processo iterativo che per ogni breakpoints permette all'utente di eseguire le operazioni suddette su di esso.
Un ulteriore vantaggio di GraphVista consiste nella possibilità che ha l'utente di salvare i vari risultati intermedi in archivi.
Per ogni salvataggio il sistema conserva la data e l'ora della creazione e un'eventuale descrizione testuale.
Tali salvataggi possono essere interpretati come dei "segnalibri" che permettono:

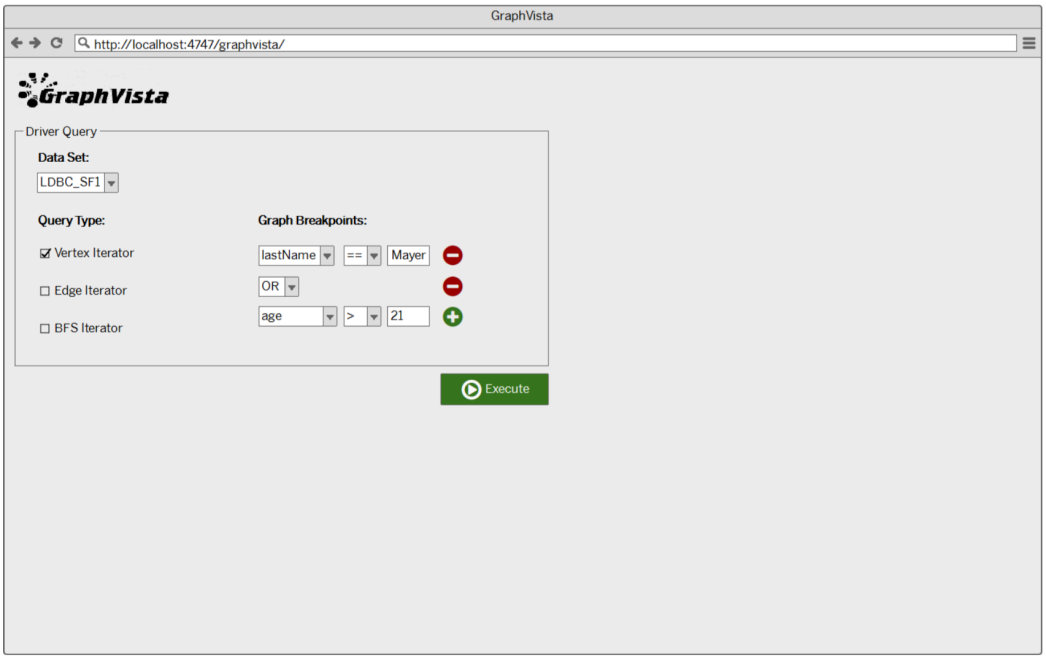
- Conservazione dei dati: salvando tutti i risultati intermedi non vi è più la necessità di reperire alcuni dati mediante il motore di ricerca. È possibile quindi reperire valori necessari all'esplorazione interattiva direttamente dai salvataggi.

- Recupero: è possibile recuperare risultati meno recenti e cominciare l'interrogazione del database a partire da essi. Grazie alla possibilità del salvataggio risulta, inoltre, possibile eseguire l'interrogazione in una sessione diversa di GraphVista rendendo le operazioni di interrogazione più flessibili.

3.3.4 Interfaccia utente

L'interfaccia utente, come tutto il sistema, è stata costruita al fine di essere abbastanza intuitiva e quindi facile da utilizzare senza il bisogno di richiedere all'utente conoscenze elevate dei linguaggi di interrogazione.
L'interfaccia presenta due pannelli distinti: uno relativo alle query driver e uno relativo all'esplorazione interattiva.

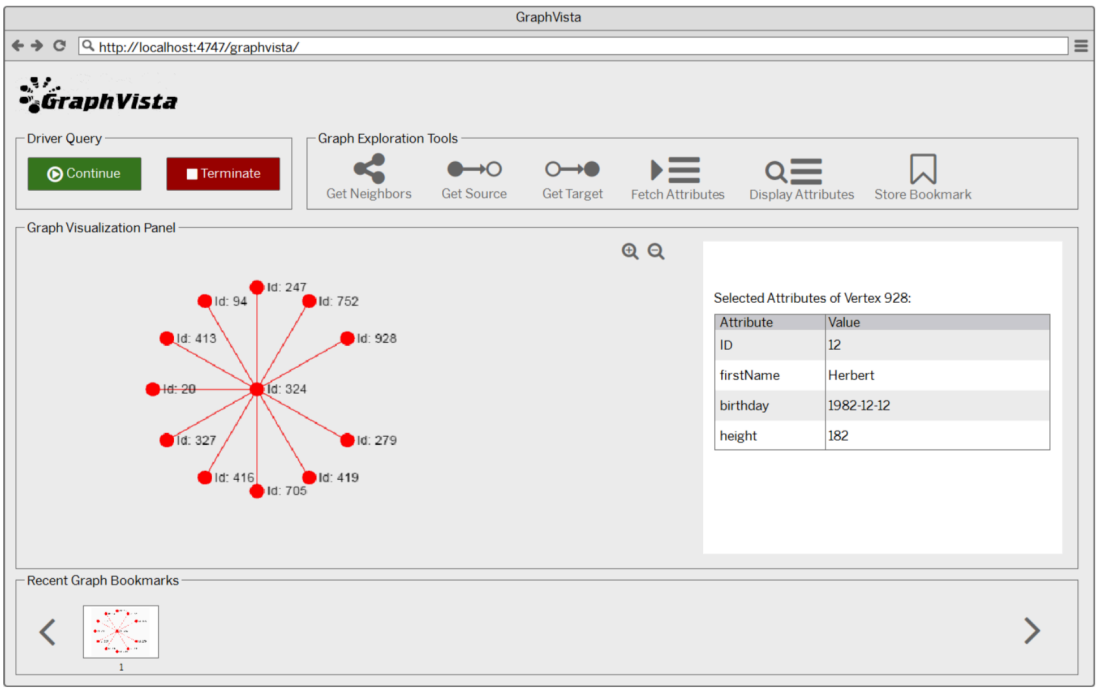
Il pannello entro cui gestire le query driver è visibile in **figura 3.6**: esso permette di lanciare la query con la quale iniziare l'esplorazione. Presenta una sezione entro cui inserire la tipologia di visita che si vuole eseguire e un'altra sezione entro cui specificare i predicati che attivano i vari punti di breakpoint.



Driver query panel.

Figura 3.6 Interfaccia utente GRAPHVISTA: pannello per la formulazione delle query driver

Il pannello inerente all'esplorazione interattiva, come è possibile vedere in **figura 3.7**, presenta varie sezioni.
In altro a sinistra vi è la sezione attraverso cui viene attivato o disattivato il motore di ricerca Graphite, ovvero attraverso cui viene gestita l'esecuzione della query driver. In altro a destra è presente la barra degli strumenti. Essa mette a disposizione dell'utente delle icone ad ognuna delle quali corrispondono varie operazioni di elaborazione del grafo.
Nella parte centrale del pannello viene visualizzato il grafo su cui l'utente può effettuare le esplorazioni interattive.
Infine nella parte bassa vengono visualizzati tutti i segnalibri, ovvero i salvataggi eseguiti dall'utente.



Interactive query panel.

Figura 3.7 Interfaccia utente GRAPHVISTA: pannello per l'esplorazione interattiva del grafo

Capitolo IV

Applicazione: GraphVista vs SPARQL

Dopo aver presentato il sistema GraphVista, in quest'ultimo capitolo verrà proposto un esempio di interrogazione attraverso cui poter, ancor meglio, evidenziare i vantaggi di tale sistema rispetto a sistemi di interrogazione non interattivi.

4.1 Applicazione

L'applicazione riguarda l'esecuzione di una query attraverso il sistema GraphVista e l'esecuzione della corrispondente query scritta con un linguaggio standard (SPARQL).

4.1.1 Dataset

Le interrogazioni si basano sullo schema di dati presente in **figura 4.1**.
In particolare lo schema in questione è un grafo raffigurante la struttura semplificata di un social network (Linkedin).
Lo schema è costituito da 10 nodi:

- **UTENTE**: questo nodo rappresenta i vari protagonisti del mondo social.
- **NOME e COGNOME**: ogni utente all'interno del social network viene identificato con un nome e un cognome.
- **DATA ISCRIZIONE**: in questo nodo viene registrata la data di iscrizione al social network di ogni utente.

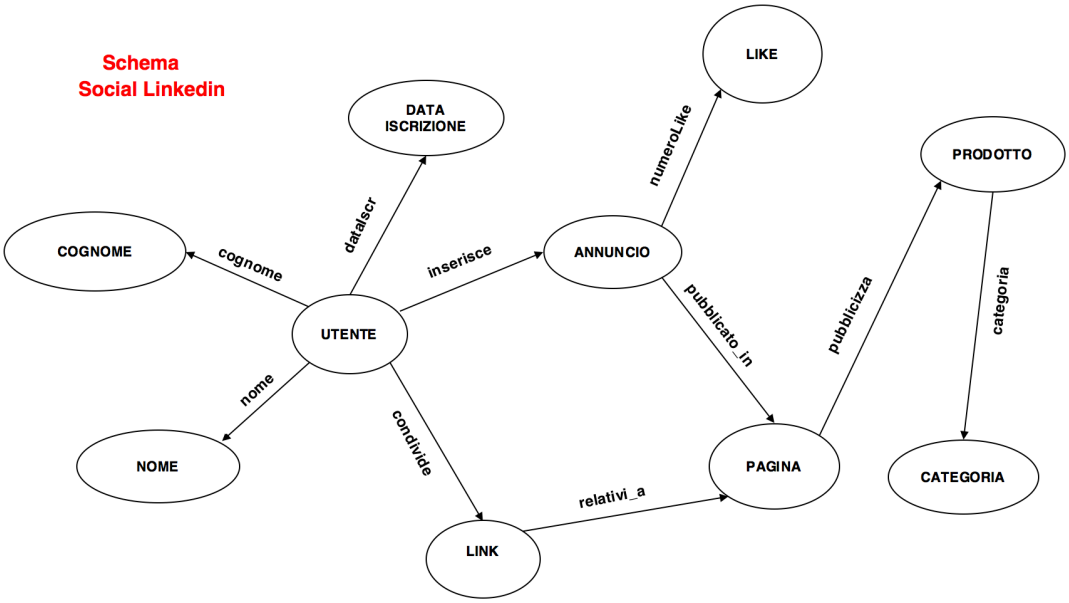


Figura 4.1 Grafo raffigurante la struttura del social Network Linkedin.

- **LINK**: Una delle operazioni principali del social network è la condivisione dei propri stati d'animo, delle emozioni, delle sensazioni dell'utente. Ciò è possibile grazie alla funzione dei link. Le informazioni relative a quest'ultimi sono memorizzate all'interno dell'omonimo nodo.

- **ANNUNCIO**: agli utenti è permesso di inserire annunci attraverso cui poter vendere/ricercare prodotti o servizi.

- **LIKE**: all'interno di questo nodo sono immagazzinate le informazioni relative al numero di like, ovvero apprezzamenti, ottenuti dagli annunci pubblicati dagli utenti.

- **PAGINA**: In questo nodo vi sono le informazioni relative alle pagine in cui gli annunci vengono pubblicati e alle pagine a cui si fa riferimento nei link.

- **PRODOTTO**: all' interno delle pagine è consentito pubblicizzare diversi prodotti. Le informazioni relative a quest'ultimi si trovano nel nodo prodotto.

- **CATEGORIA:** i vari prodotti pubblicizzati all'interno delle pagine vengono distinti per categoria (sport, auto/moto, articoli regalo etc.).

I nodi suddetti sono collegati fra loro mediante archi che ne raffigurano le relazioni.
In particolare il grafo è composto dai seguenti archi:

- **cognome, nome e datalscr**: mettono in relazione l'utente e con le caratteristiche che possiede: nome, cognome e la data in cui si è iscritto al social network.

- **inserisce**: relaziona l'utente con la possibilità di inserire annunci.

- **condivide**: relazione cardine del social in quanto evidenzia il legame dell'utente con i link.

- **numeroLike**: mette in relazione l'annuncio con il numero di apprezzamenti ricevuti.

- **pubblicato_in** e **relativi_a**: relazionano rispettivamente l'annuncio e il link con la relativa pagina.

- **pubblicizza**: collega le varie pagine agli eventuali prodotti in esse inseriti come inserzioni.

- **categoria**: mette in relazione il prodotto con la rispettiva categoria.

L'interrogazione che sarà oggetto del seguito dell'applicazione è la seguente.

QUERY:

"Restituire gli utenti iscritti nel 2016 che hanno inserito annunci relativi a prodotti sportivi e con più di 100 like. I dati da restituire sono il nome e il cognome dell'utente e il relativo annuncio".

4.1.2 Querying con SPARQL

Dopo aver descritto il dataset è il momento di cominciare ad interrogarlo. Inizialmente ci occupiamo dell'interrogazione eseguita con un linguaggio standard. Nell'applicazione è stato utilizzato SPARQL. La query è la seguente:

```
PREFIX social:<https://www.linkedin.com/uas856/submita>
SELECT ?nome ?cognome ?annuncio
FROM <https://www.linkedin.com/uas856/submita>
WHERE {

   ?utente social:nome ?nome.
   ?utente social:cognome ?cognome.
   ?utente social:dataIscr  "2016".
   ?utente social:inserisce ?annuncio.
   ?annuncio social:numeroLike ?like.
   ?annuncio social:relativo_a ?prodotto.
   ?prodotto social:categoria "sport".

   FILTER (?like > 100).
}
```

Tale interrogazione restituirà il nome, il cognome dell'utente e i rispettivi annunci inseriti. Naturalmente dovranno essere rispettati alcuni vincoli:

- la data di iscrizione dell'utente dovrà essere 2016.
- gli annunci avranno in oggetto un prodotto appartenente alla categoria sport.
- gli annunci dovranno aver più di 100 like.

Come è possibile notare, la formulazione di tale interrogazione richiede la conoscenza di un linguaggio specifico.
SPARQL segue una sintassi articolata e ciò rende la formulazione della query, quindi l'esplorazione del grafo, abbastanza difficile da esprimere, soprattutto se non si hanno buone conoscenze del linguaggio.

4.1.3 Querying con GraphVista

La query mediante il sistema GraphVista risulta essere più intuitiva e richiede una maggiore e continua collaborazione dell'utente.
Come detto precedentemente la query deve restituire, oltre al nome e al cognome dell'utente, l'elenco degli annunci pubblicati dall'utente, aventi un numero di like superiore a 100 e relativi a prodotti di categoria "sport".
La query grafica potrebbe partire proprio da quest'ultimo punto. Conoscendo più o meno il dataset, si può chiedere al sistema GraphVista di "scremare" il database applicando alcuni vincoli, ovvero si chiede di restituire tutti i prodotti della categoria "sport" inserendo il vincolo dal pannello di query driver del sistema. In **figura 4.2**, è ben visibile il vincolo inserito nella query driver: "*categoria == Sport* ".

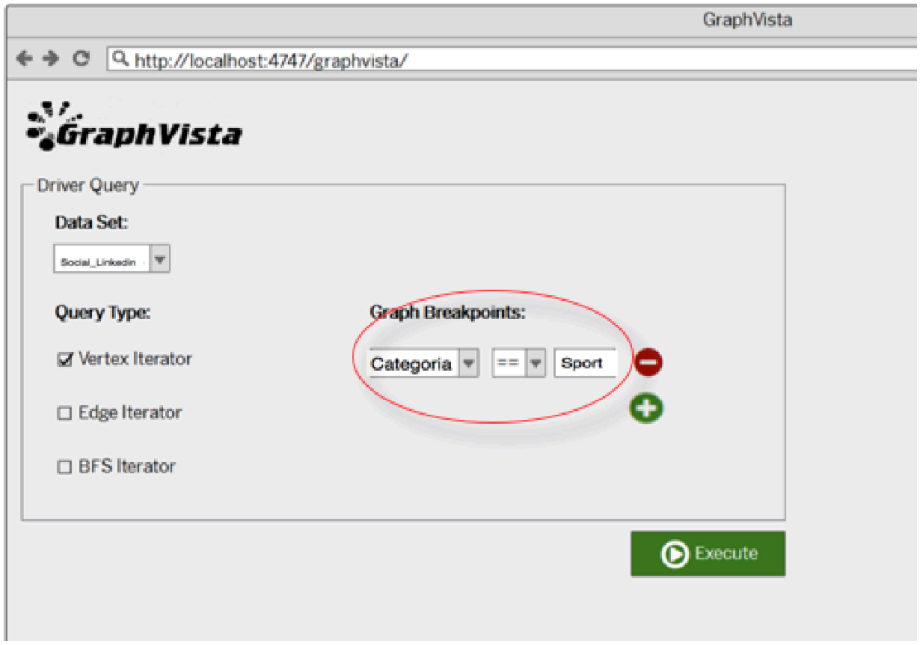


Figura 4.2 pannello query driver in cui viene inserito uno dei vincoli della query.

Sarebbe stato possibile lanciare una query driver più restrittiva in modo da avvicinarci fin da subito al risultato richiesto. In particolare si sarebbe potuto aggiungere il vincolo riguardante il numero di like ricevuti dall'annuncio come viene riportato in **figura 4.3**

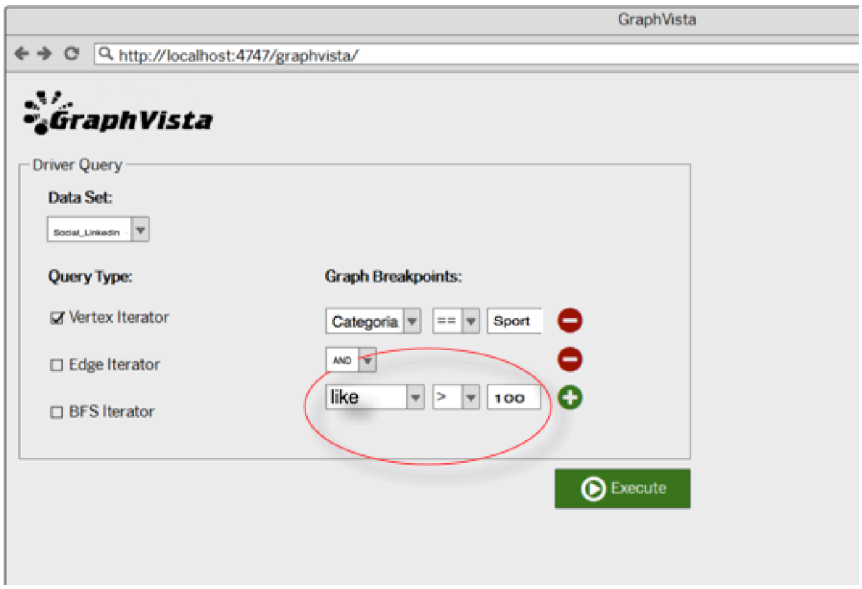


Figura 4.3 pannello query driver in cui si evidenzia l'inserimento dell'ulteriore vincolo.

Una volta inseriti i vincoli desiderati e avviata l'esplorazione del grafo, entra in gioco Graphite.
Come è raffigurato in **figura 4.4**, nella parte sinistra dell'interfaccia di Graphite viene inserita la query grafica corrispondente avente il vincolo immesso in GraphVista (categoria == sport), nella parte destra invece è possibile individuare i risultati della query. In particolare all'interno del social network considerato sono presenti 15 annunci relativi a prodotti facenti parte della categoria sport. L'elenco di tali annunci è riportato accanto alla barra laterale destra dell'interfaccia. In figura è riportato il match relativo all'utente numero 34, il quale ha inserito un annuncio nella pagina denominata "subito" relativo al prodotto "completo pallacanestro". Come richiesto il prodotto appartiene alla categoria "sport".

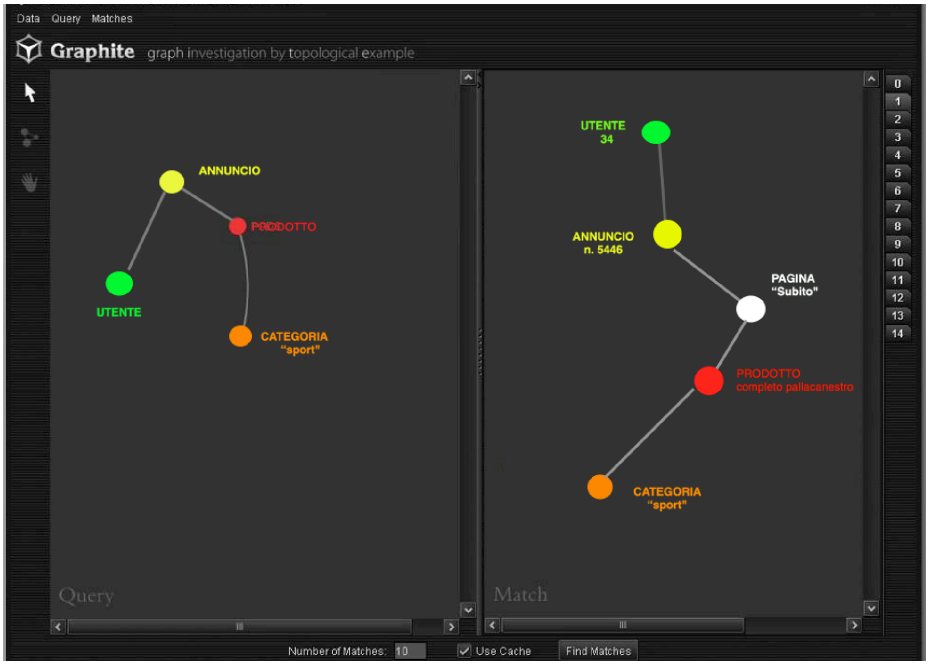


Figura 4.4 interfaccia Graphite raffigurante la query utilizzata per iniziare l'esplorazione

Come si nota dalla figura, il grafo inserito nella parte sinistra non coincide perfettamente col grafo della parte destra (è presente il nodo bianco in più). Ciò accade perché la base di dati interrogata non contiene sottografi identici a quello richiesto. In questi casi Graphite restituirà un sottografo che più si avvicina alla richiesta. I nodi che non erano presenti nella query sono subito riconoscibili poiché vengono segnati in bianco.

La query proposta in **figura 4.4**, però, non corrisponde a quella precedentemente descritta con SPARQL. Essa restituisce si tutti gli utenti che hanno inserito annunci per prodotti di tipo sportivo, ma non soltanto quelli iscritti nel 2016 e i cui annunci hanno ottenuto più di 100 like.
Ciò è stato fatto appositamente per evidenziare le caratteristiche del sistema.
Infatti partendo da tale query, l'utente attraverso l'esplorazione interattiva può comunque ottenere i risultati richiesti.
In pratica ad ogni breakpoint, ovvero ogni qualvolta il sistema "trova" un prodotto appartenente alla categoria sportiva, a partire dal rispettivo sottografo lanciato in output dal sistema l'utente può risalire agli archi in uscita dal nodo UTENTE e constatare da sé la data di iscrizione al social network. Nel caso in cui la data fosse 2016 l'utente continuerebbe ad esplorare il grafo risalendo al numero di like degli annunci, in caso contrario riavvierebbe l'esecuzione automatica e aspetterebbe un ulteriore breakpoint.

Ritornando alla nostra applicazione, la corrispondente query visiva completa risulta essere quella presente in **figura 4.5** a sinistra. Un possibile grafo corrispondente nei dati è quello mostrato a destra. Nel risultato restituito dalla query verrà quindi riportato:

Alessandro Reina n.8632

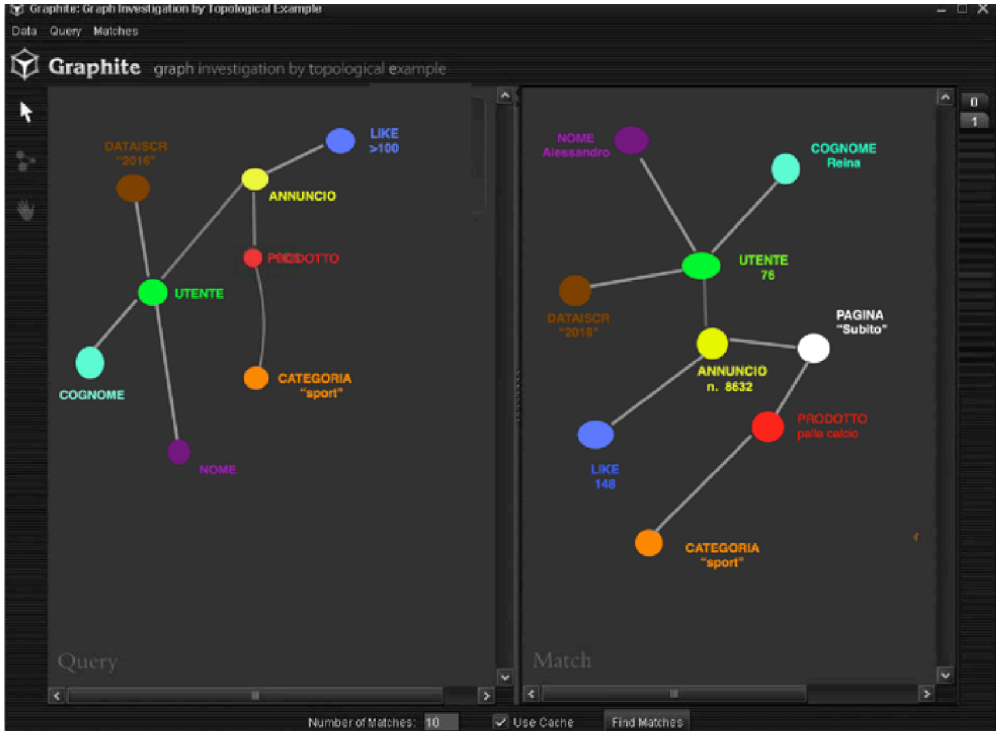


Figura 4.5 Interfaccia di Graphite in cui è eseguita la query relativa all'applicazione esemplificativa

4.2 Comparazione e Valutazioni

Le due alternative di query sono estremamente differenti in termini di complessità, di competenze richieste e soprattutto di partecipazione.
La query SPARQL è sintatticamente complessa. Per formularla vi è la necessità di conoscere il rispettivo linguaggio di interrogazione. Risulta evidente che più è complessa l'interrogazione da formulare, più sarà necessario possedere competenze elevate inerenti al linguaggio SPARQL.
Pertanto l'esecuzione di una query mediante tale linguaggio risulta assai sconveniente per gli utenti poco esperti.
Inoltre l'interrogazione risulta statica, in quanto non è possibile introdurre ulteriori vincoli prima della fine dell'esecuzione della query né tantomeno interagire con essa.

L'esecuzione della stessa query attraverso il sistema GraphVista risulta essere molto più congeniale agli utenti poco esperti di linguaggi di interrogazione.
Infatti, grazie alla possibilità di formulare la query in maniera visiva, il tutto risulta essere molto più semplice e soprattutto intuitivo. La possibilità di inserire una query driver permette di fare a meno di analizzare parti del grafo non interessanti nel risultato finale.
Successivamente è l'utente che ha la possibilità di esplorare i vari nodi e gli archi del grafo alla ricerca delle informazioni interessate. Con tale sistema, quindi, l'utente partecipa in modo attivo all'interrogazione stessa.
GraphVista è preferibile anche per il dinamismo che viene offerto all'utente:
quest'ultimo ha la possibilità di interrompere l'interrogazione quando vuole senza il bisogno che essa abbia finito di analizzare tutti i dati presenti nel database.
Ad ogni breakpoint all'utente viene chiesto se terminare l'esecuzione della query o continuare nella ricerca del prossimo breakpoint.

Dalla descrizione del sistema GraphVista nei capitoli precedenti e dall'applicazione oggetto di questo capitolo, risultano evidenti i seguenti vantaggi:

- Partecipazione attiva dell'utente all'interrogazione.
- Elaborazione query dinamica.
- Formulazione della query semplice ed intuitiva grazie al modello a grafo.
- Minor tempo necessario alla ricerca dei risultati, seppur parziali, grazie all'utilizzo dei breakpoint.
- Minore mole di dati da analizzare grazie all'uso delle query driver.
- Possibilità di salvare all'interno di appositi archivi parte del lavoro svolto, ovvero parte dell'esplorazione, per poi riprenderlo in un secondo momento.
- Nessuna richiesta di competenze tecniche inerenti a linguaggi di interrogazione.

Conclusioni

Nel capitolo precedente si sono ampiamente discusse e dimostrate le migliorie apportate dal sistema innovativo di interrogazione di basi di dati a grafo oggetto della tesi.
Oltre alle caratteristiche tecniche, ampiamente descritte in precedenza e decisamente più efficienti rispetto ai sistemi tradizionali, il punto di forza di questo sistema risiede nella possibilità offerta all'utente di interagire e quindi partecipare attivamente all'interrogazione stessa.
Questa caratteristica rende GraphVista davvero innovativo in quanto riesce a motivare e responsabilizzare ancor più l'utente.

Tale sistema però è ancora in fase sperimentale motivo per cui è poco diffuso rispetto alle potenzialità, ma le caratteristiche che offre saranno indubbiamente alla base dei sistemi di interrogazione di basi di dati a grafo del futuro prossimo.

Bibliografia

[1] Wikipedia, https://it.wikipedia.org

[2] Ilaria Castelli. Algoritmi di visita di un grafo, Università degli studi Siena, pp 2-30, 2010.

[3] Robert Pienta, James Abello, Minsuk Kahng, Duen Horng Chau. “Scalable Graph Exploration and Visualization: sensemaking challenges and opportunities”. International Conference on Big Data and Smart Computing (BIGCOMP), pp 271-278, 2015.

[4] Sherif Sakr, Sameh Elnikety, Yuxing He. ”G-SPARQL: A Hybrid Engine for Querying Large Attributed Graphs”. International conference on Information and knowledge management. New York, pp 335-344, 2012.

[5] Florian Holzschuher, René Peinl. “Querying a graph database – language selection and performance considerations”. Journal of Computer and System Sciences 82.1, pp 45-68, 2016.

[6] Renzo Angles, Claudio Gutierrez. “An introduction to Graph Data Management”, pp 12-15.

[7] Sonal Raj. “Neo4j High Performance”, Packt Publishing Ltd, Birmingham, pp 18-26, 2015.

[8] Marcus Paradies, Michael Rudolf, Wolfgang Lehner. “GraphVista: Interactive exploration of large graphs”, arXiv preprint arXiv:1506.00394, 2015.

[9] Duen Horng Chau, Christos Faloutsos, Hanghang Tong, Jason I. Hong, Brian Gallagher, Tina Eliassi-Rad. “GRAPHITE: A Visual Query System for Large Graphs”. International Conference on Data Mining Workshops, Pisa, pp 963-966, 2008.

[10] Marcus Paradies, Wolfgang Lehner, Christof Bornhövd. “GRAPHITE: An Extensible Graph Traversal Framework for Relational Database Management Systems”. International Conference on Scientific and Statistical Database Management, New York, pp 29, 2015.